CC20A
Modelos y Herramientas en Tecnología de la Información

10 UD
Requisitos: CC10A

Objetivos

El estudiante deberá ser capaz de modelar problemas y soluciones utilizando herramientas computacionales de alto nivel. Los problemas serán referidos a organizaciones y sistemas que puedan modelarse con servidores y clientes, colas, almacenamientos y procesos.

Las soluciones suponen modelamiento de los datos del problema y la utilización de herramientas computacionales para su procesamiento.

Metodología

En las clases de cátedra se expondrán las técnicas de modelamiento, mientras que en las clases auxiliares se introducirá a los alumnos en las herramientas para apoyar estos procesos. Ejemplos de estas herramientas son SIMSCRIPT (simulación) y miniSQL (Bases de datos).

Temario

1.4. Modelos más eficientes. Modelos de eventos discretos

1.4.1. Ejemplo 8. Supermercado

1.4.2. Instantes de ocurrencia.

Ejercicios.

Bibliografía: "Theory of Modelling and Simulation", Bernard P. Zeigler

2. Modelamiento de datos

2.1. Introducción.

2.2. Modelo entidad-relación.

2.2.1. Tipos de relaciones.

2.2.2. Tipos de atributos.

2.3. El modelo relacional

2.3.1. Base de datos relacional.

2.4. Lenguaje de consulta SQL

2.4.1. Consultas en SQL.

2.4.2. Consultas en SQL (continuación).

Bibliografía: "Fundamentals of Database Systems", R. Elmasri y S.B. Navathe

3. Modelamiento de redes

3.1. Grafos

3.2. Árboles

3.3. Centralidad en árboles

3.4. Recorrido de árboles

Bibliografía:

F. Havary, "Graph Theory". Addison-Wesley, Reading, 1972.
A. Rosenthal, J.A.Pino, "A Generalized Algorithm for Centrality Problems on Trees". Journal of the ACM, 36, No.2, April 1989, 349-361.

[anterior] [home] [siguiente]

1.1. Modelamiento y simulación (introducción)

Entenderemos por "modelamiento y simulación" a las actividades asociadas con la construcción de modelos de sistemas del mundo real, y su simulación en un computador.

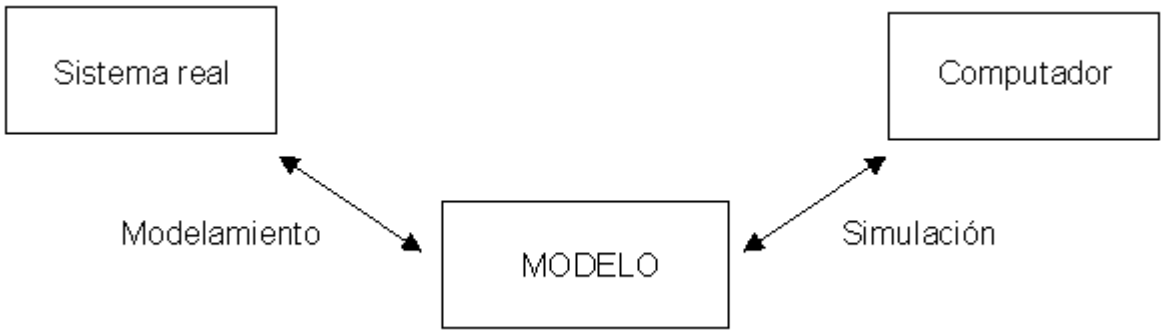
Los modelos son útiles para predecir y/o estudiar el comportamiento de un sistema real, que puede servir para corroborar algunas hipótesis. Por ejemplo, para predecir el movimiento de objetos en la superficie de la Tierra, sometidos a la fuerza de gravedad, las ecuaciones de Newton son un modelo suficientemente bueno. Con ellas se podría predecir cuál sería el movimiento (la posición segundo a segundo) de un objeto que se suelta desde una altura determinada. Sin embargo, si los objetos se mueven a velocidades comparables con la velocidad de la luz, las ecuaciones de Einstein describen mejor su comportamiento.

Muchas veces se usan modelos de sistemas (que incluso puede que no existan todavía) para ver cómo funcionan estos sistemas bajo distintas condiciones (con distintos parámetros) y ver cuáles son las condiciones necesarias para que el sistema sirva o trabaje en forma óptima. Hay muchas razones por las cuales es conveniente experimentar en un modelo y no en la vida real: costos, tiempo, peligro o simplemente imposibilidad. Los experimentos son repetibles. Algunos ejemplos de sistemas de la vida real que pueden ser modelados son: supermercados, hospitales, redes de caminos, represas, redes de computadores y modelos económicos.

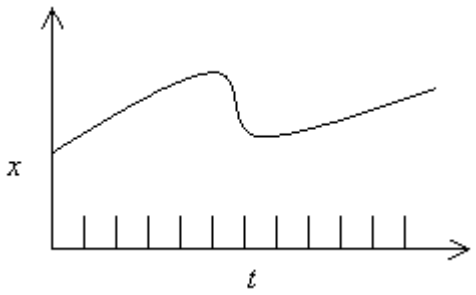
En general, los modelos son una simplificación de la vida real. Esto porque el sistema real generalmente es muy complicado, o porque sólo se pretente estudiar una parte del sistema real.

El proceso de definición del modelo de un sistema (real o no real) se llama *modelamiento*. La *simulación* consiste en usar el modelo para generar datos acerca del comportamiento del sistema para ver cómo se comportaría, bajo el supuesto de que el modelo está bien hecho.

En general, cualquiera sea la forma que adopte, el modelo debe ser capaz de proveer instrucciones a alguien o algo, de modo que pueda generar datos que describan el comportamiento del sistema modelado. Así entonces, se tiene un *sistema real*, cuyo *modelamiento* genera un *modelo* que puede ser representado en un *computador* a través de una *simulación*.



El "sistema real" es la parte del mundo real de nuestro interés. Como regla general, podemos decir que el sistema real es una *fuente de datos conductuales*, los cuales consisten en formas primarias de gráficos *x* versus *t*, donde *x* puede ser cualquier variable de interés, tal como la temperatura de un cuarto, el número de pétalos de una flor o el Producto Nacional Bruto, y *t* es el tiempo, medido en unidades como segundos, días o años. La siguiente figura muestra un ejemplo.



Un *modelo* es básicamente un conjunto de instrucciones para generar datos conductuales (con cierto comportamiento) de la forma de la figura anterior. Los modelos son algunas veces expresados en forma de ecuaciones diferenciales, notación teórica de autómatas o en formalismo de eventos discretos.

¿Cómo se sabe si un modelo es bueno? Para saber si un modelo es bueno, basta comparar los resultados que arroja con los del sistema real que se quiere estudiar. La validez del modelo depende de "cuan bien el modelo representa al sistema real" en términos de los datos arrojados por el modelo versus los datos del sistema real. El problema es que muchas veces el sistema no existe, pues puede ser un modelo de algo que se quiere construir. En todo caso, lo principal de un modelo es que los resultados que arroje reflejen de alguna manera lo que se quiere estudiar acerca del sistema que se está modelando.

No existe ninguna manera de saber cuál es el mejor modelo para un sistema. Es posible comparar dos modelos y decidir cuál de ellos es mejor bajo algún punto de vista particular, pero en ningún caso se puede saber exactamente cuándo se está frente al mejor modelo para una situación dada.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.2.1. Ejemplo 1 - Sistema de tiempo compartido

Un sistema computacional de tiempo compartido está compuesto por un computador y cinco usuarios en terminales conectadas al computador, como se muestra en la siguiente figura.

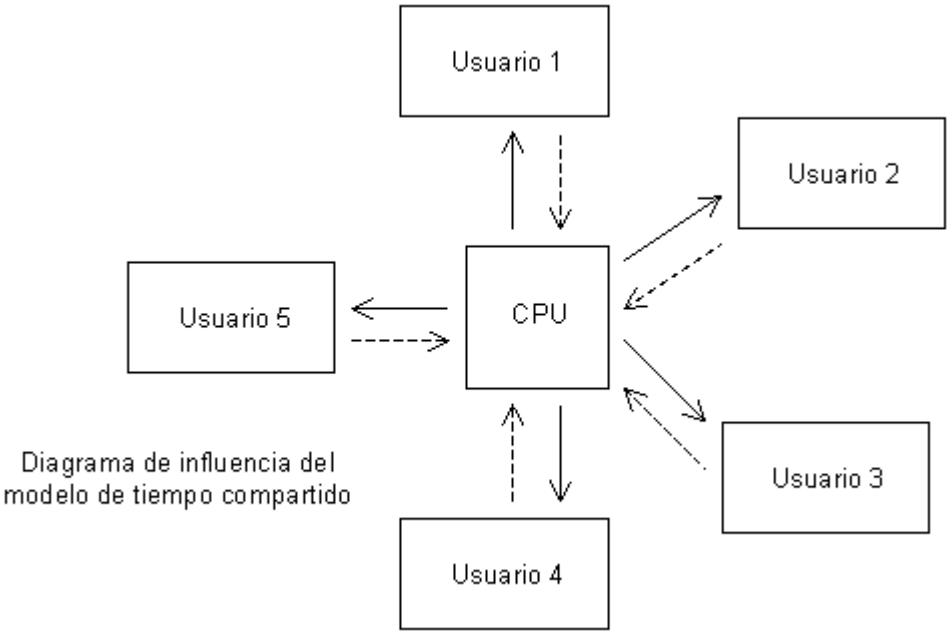


Diagrama de influencia del modelo de tiempo compartido

El computador sirve a cada usuario por turnos, moviéndose en el sentido de las manecillas del reloj. Cuando un usuario tiene su turno, transmite sus datos a la CPU y espera una respuesta. Cuando recibe su respuesta, empieza a preparar los datos para la próxima entrega. El interés del modelo es estudiar qué tan rápido un usuario completa el desarrollo de su programa.

Descripción

Componentes

CPU, USUARIO1, USUARIO2, ..., USUARIO5.

Variables descriptivas

CPU

QUIEN·AHORA - con rango {1,2,3,4,5}; QUIEN·AHORA = i indica que USUARIOi está siendo atendido por el CPU.

USUARIOi (i = 1,2,3,4,5)

ESTADO - con rango [0,1]; ESTADO = s indica que un usuario ha progresado una fracción de tiempo s en completar su programa (cero significa empezando, 1/2 es la mitad, 1 significa que terminó).

PARÁMETROS

ai - con rango [0,1]. Tasa de trabajo realizado por USUARIOi.

Interacción entre componentes

1. El CPU sirve a cada usuario por turnos, con una tasa fija. De este modo, QUIEN·AHORA sigue el ciclo 1,2,3,4,5,1,2,...

2. Cuando USUARIOi tiene su turno (es decir, cuando QUIEN·AHORA toma el valor de i), el usuario completa una fracción ai del trabajo que le falta, es decir, si su ESTADO es s, éste se convierte en s + ai(1 - s).

Supuestos

1. El tiempo de servicio dado a un usuario se asume fijo. Es por esto que las flechas de líneas no continuas en el diagrama de estado de la figura anterior, indica que en este modelo no hay influencia de USUARIO sobre CPU.

2. El progreso del USERi en la terminación de su programa, sigue una tasa exponencial, determinada por su parámetro individual ai.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.2.2. Ejemplo 2 - Dinámica de las relaciones Gobierno-pueblo

Considere el modelamiento de un país consistente de un gobierno y la gente. El gobierno es dirigido por un PARTIDO en el poder, y éste puede ser LIBERAL o CONSERVADOR, lo que determina la POLITICA interna, siendo ésta PERMISIVA o COERCITIVA. La gente reacciona a las acciones del gobierno, y en un momento determinado, estará en un estado de CONMOCION·CIVIL que puede ser ALTO o BAJO. Muchas propuestas pueden hacerse sobre cómo la gente racciona a los cambios en las políticas del gobierno, y cómo, en respuesta, el gobierno determina su política en respuesta al comportamiento del pueblo. Cuando un conjunto de propuestas no es satisfactorio o completo, éste determina una interacción particular entre la gente y el gobierno. El presente modelo ilustra esta situación.

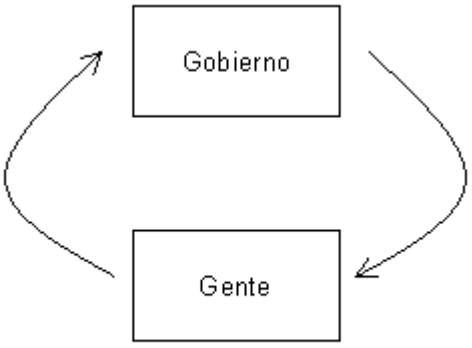


Diagrama de influencia gobierno-gente

Descripción

Componentes

GOBIERNO, PUEBLO.

Variables descriptivas

GOBIERNO

PARTIDO - con rango {CONSERVADOR, LIBERAL}; indica la tendencia política (ideología) del GOBIERNO.

POLITICA - con rango {PERMISIVA, COERCITIVA}; indica el tipo de política que el GOBIERNO está siguiendo.

PUEBLO

CONMOCION·CIVIL - con rango {BAJA, ALTA}; indica el estado general de malestar del PUEBLO.

Interacción entre componentes

1. Una política gubernamental COERCITIVA es invariablemente seguida en el siguiente año por un ALTO grado de CONMOCION·CIVIL.

2. Por el contrario, un gobierno PERMISIVO siempre es capaz de producir y/o mantener un BAJO nivel de malestar civil durante un año.

3. Un PARTIDO permanece en el poder tanto como la CONMOCION·CIVIL sea BAJA, siendo reemplazado al término de un año si el malestar se vuelve ALTO.

4. Una vez en el poder, un gobierno CONSERVADOR nunca cambia su POLITICA, ni tampoco cambia la POLITICA de su predecesor cuando recien asume el poder.

5. Un gobierno LIBERAL reacciona a un ALTO grado de CONMOCION·CIVIL mediante una legislación PERMISIVA, pero un año después de que la quietud ha regresado, invariablemente toma una actitud COERCITIVA.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.2.3. Ejemplo 3 - Sistema mundial

El siguiente ejemplo modela la interacción entre la industria, la población y la contaminación de un país o ciudad. El diagrama de influencia del modelo se muestra en la siguiente figura.

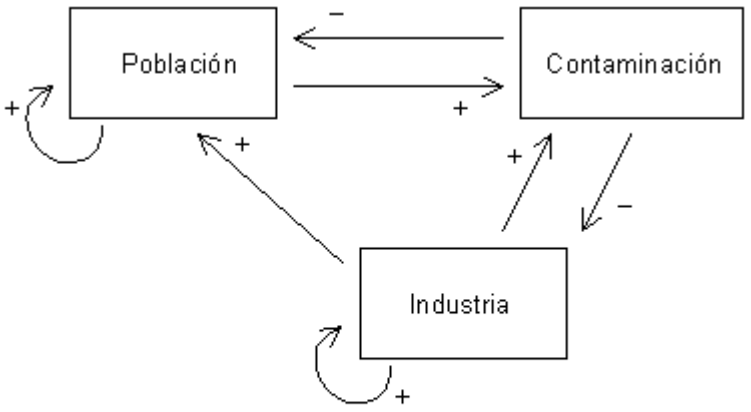


Diagrama de influencia del modelo mundial
Los signos + y - indican incrementos o decrementos en la influencia

Descripción

Componentes

POBLACION, CONTAMINACION, INDUSTRIA.

Variables descriptivas

POBLACION

DENSIDAD·POBLACION - con rango en los números reales positivos;
DENSIDAD·POBLACION = x indica que actualmente hay x personas por metro cuadrado habitando el mundo.

CONTAMINACION

NIVEL·CONTAMINACION - con rango en los números reales positivos;
NIVEL·CONTAMINACION = y indica que el actual nivel de contaminación del ambiente es y unidades de alguna escala no especificada.

INDUSTRIA

CAPITAL·INDUSTRIAL - con rango en los números reales positivos;
CAPITAL·INDUSTRIAL = z indica que el mundo industrial total está actualmente valorado en z dólares.

Interacción entre componentes

1. La tasa de crecimiento de DENSIDAD·POBLACION se incrementa linealmente con el incremento en DENSIDAD·POBLACION y CAPITAL·INDUSTRIAL. Ésta decrece linealmente con el incremento en NIVEL·CONTAMINACION.

2. La tasa de crecimiento del NIVEL·CONTAMINACION se incrementa linealmente con el incremento en DENSIDAD·POBLACION y CAPITAL·INDUSTRIAL.

3. La tasa de crecimiento de CAPITAL·INDUSTRIAL se incrementa linealmente con el incremento del CAPITAL·INDUSTRIAL y se decrementa linealmente con el incremento del

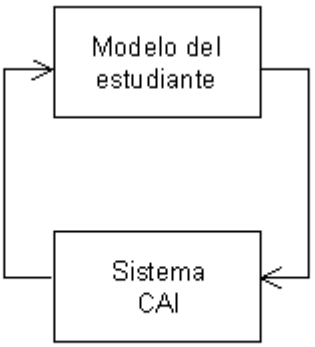
NIVEL·CONTAMINACION.

[anterior] [home] [siguiente]

1.2.4. Ejemplo 4 - Modelo del estudiante

En el diseño de un sistema CAI ("Computer-Aided Instruction") se construyó un modelo de estudiante para probar la operación del sistema. El modelo incluye parámetros tales como conocimiento inicial, habilidad organizacional, retención de memoria y capacidad para resolver problemas. Distintos rangos en estos parámetros determinan distintas categorías de estudiantes. Se presenta aquí una versión simplificada del modelo.

El modelo del estudiante fue diseñado para tomar el lugar de un estudiante real trabajando en una sesión CAI (ver la figura). El modelo de estudiante recibe un número de documento (pregunta) del sistema CAI, y decide si el nivel de conceptos relevantes en este documento es suficientemente alto. En caso afirmativo el modelo reporta que "entendió" el documento, y en caso contrario reporta que "no entendió" el documento. Dependiendo de su respuesta, el sistema CAI selecciona otro número de documento para su presentación al modelo de estudiante. El modelo considera el nuevo documento, y así sucesivamente.



Interacción entre el sistema CAI y el modelo del estudiante

Descripción

Componentes

REGISTRO·ENTRADA - representa el número de documento que va a ser presentado al modelo.

CONCEPTO·1, CONCEPTO·2, ..., CONCEPTO·Ncon - Los conceptos que caracterizan el curso CAI que actualmente esta siendo "enseñado" al modelo. Hay Ncon conceptos.

REGISTRO·SALIDA - despliega la respuesta del modelo.

Variables descriptivas

REGISTRO·ENTRADA

DOCUMENTO - con rango {1, 2, ..., Ndoc}; DOCUMENTO = x indica que el documento número x está siendo actualmente presentado al modelo.

CONCEPTO·i (i = 1,2,...,Ncon)

COMPRENSION - con rango [0,1]; COMPRENSION = q indica que el CONCEPTO·i ha sido aprendido una fracción q de la comprensión total del mismo.

REGISTRO·SALIDA

Y - con rango {ENTENDIDO, NO·ENTENDIDO}

PARAMETROS

Caracterización del estudiante

Δu - con rango en los reales positivos; Incremento en el refuerzo cuando DOCUMENTO·ENTENDIDO.

Δn - con rango en los reales positivos; Incremento en el refuerzo cuando DOCUMENTO·NO·ENTENDIDO.

Δf - con rango en los reales positivos; Incremento en el olvido cuando el concepto no fue usado.

Caracterización del curso

Ndoc - con rango en los enteros positivos; El número de DOCUMENTOS.

Ncon - con rango en los enteros positivos; El número de CONCEPTOS.

Para cada DOCUMENTO x

x-CONCEPTOS·RELEVANTES - con rango en el subconjunto {1,2,...,Ncon}; Los conceptos que son relevantes para el entendimiento del DOCUMENTO x.

DIFICULTAD(x) - con rango en los reales positivos; Nivel de dificultad del DOCUMENTO x.

Interacción entre componentes

Cuando un DOCUMENTO x es ingresado al modelo el siguiente procedimiento es llevado a cabo:

1. Los x-CONCEPTOS·RELEVANTES son leidos, y basados en la comparación del total de COMPRENSION y el nivel de dificultad DIFICULTAD(x) del DOCUMENTO, una decisión de ENTENDIDO o NO·ENTENDIDO es presentada.

2. Para cada x-CONCEPTO·RELEVANTE·i, si DOCUMENTO x fue ENTENDIDO, la COMPRENSION del CONCEPTO·i se incrementa en un monto de Δu (que representa el refuerzo). Si DOCUMENTO x fue NO·ENTENDIDO, su COMPRENSION es incrementada en Δn (que representa algún aprendizaje del uso, aun si su uso no fue totalmente satisfactorio). Usualmente Δu > Δn.

3. Para cada NO-x-CONCEPTO·RELEVANTE·j, la COMPRENSION de CONCEPTO·j es decrementada en un monto de Δf (que representa la tendencia a olvidar los conceptos que no son usados).

[anterior] [home] [siguiente]

1.2.5. Ejemplo 5 - Transporte de pasajeros

Este ejemplo modela el transporte en bus de pasajeros entre dos estaciones. Los pasajeros pueden abordar el bus en cualquier estación y permanecer en el bus tantas paradas como deseen, ya que en el modelo real, actualmente no se tiene control sobre el tiquete de los pasajeros una vez que ellos entran en el bus. La compañía de buses está interesada en invertir en personal o equipo para resolver este problema, y debido a esto ha inicido la construcción del modelo.

Descripción

Componentes

PUERTA·ENTRADA·1, PUERTA·ENTRADA·2, ESTACION·1, ESTACION·2, BUS.

Variables descriptivas

PUERTA·ENTRADA·i (i = 1, 2)

#·LLEGANDO·i - con rango en los enteros positivos; #·LLEGANDO·i = Xi indica que Xi personas están entrando a la estación en este momento.

ESTACION·i (i = 1, 2)

#·ESPERANDO·i - con rango en los enteros positivos; #·ESPERANDO·i = Qi indica que Qi personas están actualmente esperando en ESTACION·i por el BUS.

BUS

#·EN·BUS - con rango en los enteros positivos; #·EN·BUS = Qb indica que hay Qb pasajeros actualmente en el BUS.

TIEMPO·DE·VIAJE - variable aleatoria con rango en los reales positivos.
TIEMPO·DE·VIAJE = s significa que el BUS toma s unidades de tiempo para ir de la estación actual a la siguiente.

PASAJEROS·QUE·BAJAN - variable aleatoria con rango en los enteros positivos;
PASAJEROS·QUE·BAJAN = n significa que n pasajeros dejarán el BUS en la estación.

PARAMETROS

CAPACIDAD - con rango en los enteros positivos; especifica el máximo número de pasajeros que el BUS puede transportar.

Pi (i = 1, 2) - probabilidad de que un pasajero abandone el BUS en la ESTACION·i.

MEDIA (SIGMA) - con rango en los reales positivos; promedio y desviación estándar de TIEMPO·DE·VIAJE entre estaciones.
Kon (Koff) - con rango en los reales positivos; El tiempo que le toma a cada pasajero entrar (bajar) del BUS.

Interacción entre componentes

1. El BUS viaja de ESTACION·i a ESTACION·j. El tiempo de llegada a la ESTACION·j es determinado muestreando el TIEMPO·DE·VIAJE (normalmente distribuido, con parámetros MEDIA y SIGMA).

2. Al llegar a la ESTACION·j el BUS:

(a) Deja a los pasajeros que desean bajarse (este número es una muestra de PASAJEROS·QUE·BAJAN)

(b) Recoge pasajeros en la ESTACION·j hasta que la ESTACION·j esté vacía (#·ESPERANDO·j = 0) o el BUS esté lleno (#·EN·BUS = CAPACIDAD).

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.2.6. Categorías de modelos

Los cinco ejemplos anteriores ilustran algunas características fundamentales de los modelos. Una categorización básica tiene que ver con el *tiempo* en el cual los eventos de los modelos ocurren. Un modelo puede ser de *tiempo continuo* si el tiempo es especificado como un flujo continuo. En un modelo de *tiempo discreto* el tiempo transcurre a saltos. Los ejemplos 1, 2 y 4 son de tiempo discreto. Los ejemplos 3 y 5 son de tiempo continuo.

Una segunda categoría se relaciona con el rango de las variables descriptivas del modelo. El modelo puede ser de *estado discreto* si las variables sólo pueden contener un conjunto discreto de valores. Un modelo es de *estado continuo* si el conjunto de valores puede ser representado por un número real o intervalos de ellos. Si el modelo contiene variables de rango continuo y discreto, se dice que el modelo es de *estado mixto*. El ejemplo 3 es de estado continuo. El ejemplo 2 es de estado discreto, y los ejemplos 1, 4 y 5 son de estado mixto.

Los modelos de tiempo continuo pueden ser divididos a su vez en modelos de *eventos discretos* y modelos de *ecuaciones diferenciales*. Un modelo especificado por ecuaciones diferenciales es un modelo de tiempo continuo y estados continuos, en el cual los cambios de estado son continuos, por lo que los cambios en el tiempo son controlados por ecuaciones diferenciales. El ejemplo 3 es un modelo de este tipo. En un modelo de eventos discretos, aunque en el sistema real el tiempo transcurra de forma continua, los cambios de estado ocurren como saltos discontinuos. Los saltos son gatillados por eventos y éstos ocurren en forma arbitraria, separados unos de otros, por lo que un número finito de eventos puede ocurrir en un lapso de tiempo finito.

Una tercera categoría incorpora las variables de tipo aleatorio en la descripción del modelo. En un modelo *determinístico* no aparecen estas variables, mientras que en un modelo *estocástico* o *probabilístico* hay al menos una variable cuyo valor se calcula de forma aleatoria. El ejemplo 5 es estocástico y los ejemplos 1, 2, 3 y 4 son determinísticos.

Una cuarta forma de categorizar los modelos es relacionándolos con la manera en que el sistema real interactúa con su entorno. Si el sistema real está aislado del entorno, entonces se dice que es *autónomo*. Si por el contrario el sistema recibe influencias del entorno, se dice que es *no autónomo* o *dependiente del medio.* En este caso el modelo tiene variables de entrada (INPUT) las cuales no son controladas por el modelo, pero tiene que responder a ellas. Los ejemplos 4 y 5 son modelos no autónomo. Los ejemplos 1, 2 y 3 son sistemas autónomos.

[anterior] [home] [siguiente]

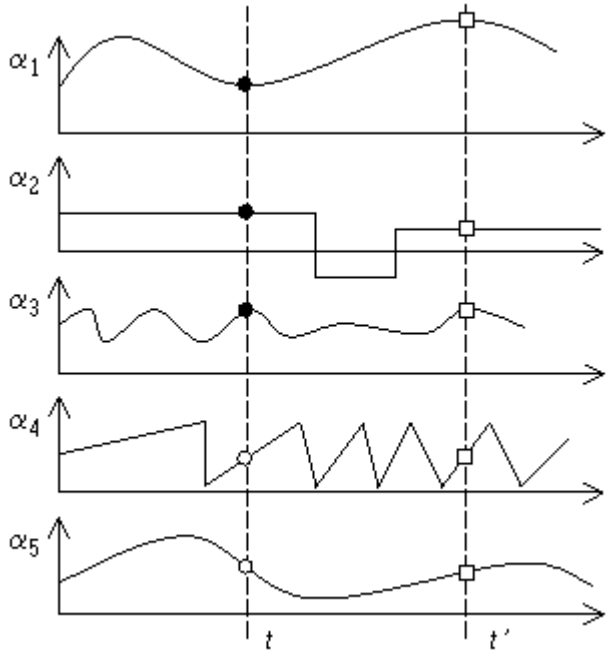
1.3. Especificación formal de modelos

El concepto de estado

Las componentes de un modelo son descritas por un conjunto de variables descriptivas. Las reglas que especifican la interacción entre componentes determinan la manera en la cual estas variables descriptivas cambian con el tiempo. Para que un computador sea capaz de simular el modelo, debe "conocer" estas reglas de interacción.

En muchos modelos es posible designar un pequeño subconjunto de todas las variables descriptivas de tal forma que es suficiente conocer el valor actual de este subconjunto de variables para calcular los valores futuros de todas las variables descriptivas. A este subconjunto de variables las llamaremos "variables de estado".

Considere un modelo con las variables descriptivas α1, α2, α3, ..., αn. Decimos que los valores de estas variables son y1, y2, y3, ..., yn en un instante de tiempo t, si α1 toma el valor y1, α2 toma el valor y2, ..., αn toma el valor yn en el instante t. Se considera un modelo "bien definido" ("bien descrito") si las reglas de interacción entre componentes determinan, para cada instante futuro t' (t' > t), un único conjunto de valores y'1, y'2, y'3, ..., y'n dados los valores y1, y2, y3, ..., yn en el instante t. De este modo, dados los valores y1, y2, ..., yn en un tiempo t, el computador puede calcular los valores y'1, y'2, ..., y'n para cualquier tiempo t' (con t' > t). Esto se muestra en la siguiente figura.

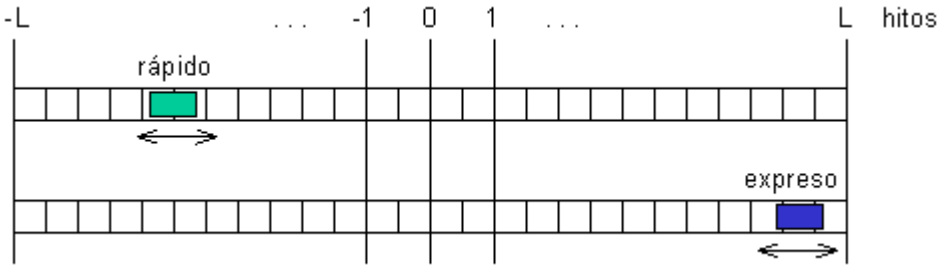


La figura anterior representa un modelo con cinco variables decriptivas α1, α2, α3, α4, α5. Si el modelo está "bien definido", los valores representados por círculos (en t) determinan los valores representados por cuadrados (en t'). Si α1, α2 y α3 conforman el conjunto de variables de estado, los valores de los círculos negros son suficientes para determinar todos los valores representados por cuadrados.

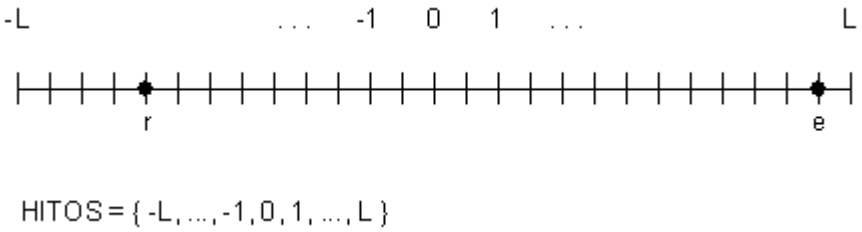
[anterior] [home] [siguiente]

1.3.1. Ejemplo 6 - Sistema de trenes

Los maquinistas de dos trenes, el "rápido" y el "expreso" están tratando de llegar a un punto común para traspasar una carga. Cada tren corre sobre sus propios rieles, paralelos a los del otro. Los trenes disponen de un sistema simple de comunicación que permite detectar si el otro tren está hacia adelante o hacia atrás. El ferrocarril corre entre -L y L, y hay hitos en los cuales se activa el sistema de comunicación (hay un total de 2L + 1 hitos).



Podemos simplificar la figura anterior de la siguiente manera:



Descripción

Componentes

TREN·RAPIDO, TREN·EXPRESO, HITOS.

Variables descriptivas

TREN·RAPIDO

RAPIDO - con rango HITOS; describe la ubicación del TREN·RAPIDO.

TREN·EXPRESO

EXPRESO - con rango HITOS; describe la ubicación de TREN·EXPRESO.

Para cada P en HITOS

R·SEÑAL·EN·P - con rango {-1,1}

E·SEÑAL·EN·P - con rango {-1,1}

Interacción entre componentes

La base temporal es discreta: t, t+1, t+2, ...

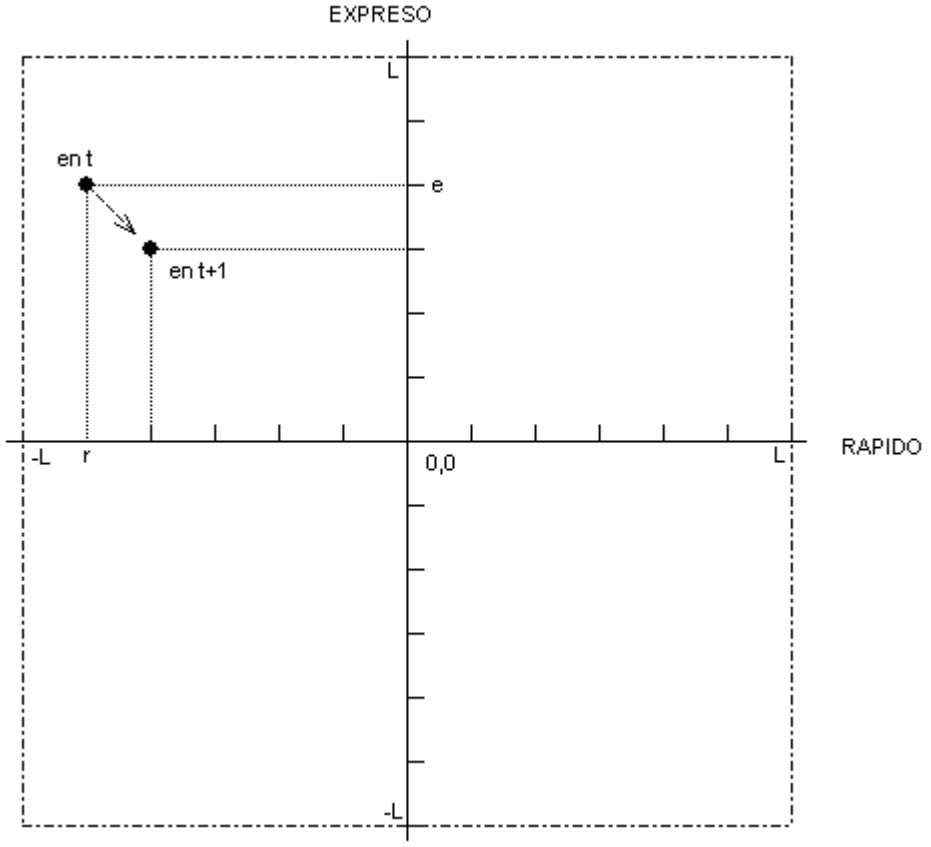
(a) El valor de R·SEÑAL·EN·P en el instante t es D((RAPIDO en t) - P) donde D(x) es 1 si x >= 0 y -1 si x < 0.

(b) El valor de E·SEÑAL·EN·P en el instante t es D((EXPRESO en t) - P).

(c) RAPIDO en t+1 = [RAPIDO en t] + [E·SEÑAL·EN·(RAPIDO en t) en t].

(d) EXPRESO en t+1 = [EXPRESO en t] + [R·SEÑAL·EN·(EXPRESO en t) en t].

El estado del modelo lo podemos representar por el par (r,e), como se muestra en la siguiente figura:



Verificamos ahora que RAPIDO y EXPRESO son variables de estado. Supongamos que conocemos los valores de RAPIDO y EXPRESO en el instante t. Llamemos a estos valores **r** y **e**, respectivamene. Veremos cómo calcular los valores de todas las variables descriptivas en t+1.

De (a), tenemos que para cada hito P: R·SEÑAL·EN·(P en t) = D(r-P)

Similarmente, E·SEÑAL·EN·(P en t) = D(e-P)

En particular, para el hito r (la ubicación del rápido):

R·SEÑAL·EN·(r en t) = D(r - r) = 1
E·SEÑAL·EN·(r en t) = D(e - r) (*)

Ejercicio: Calcular señales en el hito e en t.

De (c) y de (*) tenemos: RAPIDO en t+1 = r + E·SEÑAL·EN·(r en t) = r + D(e - r) (**)

Ejercicio: Calcule EXPRESO en t+1.

Usando (a) y (**) tenemos:

R·SEÑAL·EN·(P en t+1) = D((RAPIDO en t+1) - P) = D(r + D(e - r) - P) (***)

Ejercicio: Calcular E·SEÑAL·EN·(P en t+1).

De las ecuaciones (**) y (***), y de los ejercicios, vemos que los valores de todas las variables descriptivas en t+1 se expresan en términos de **r** y **e**, los valores del rápido y el expreso en el instante t.

[anterior] [home] [siguiente]

[anterior][home] [siguiente]

1.3.2. Propiedades de las variables de estado

Podemos ver ahora cómo el computador puede realizar los cálculos necesarios para ejecutar una simulación de manera *iterativa*. Suponga que queremos calcular los valores de las variables descriptivas $y'_1, y'_2, ..., y'_n$ en t', dados los valores de $y_1, y_2, ..., y_m$ en t. Suponga que las reglas de interacción pueden ser traducidas a un programa computacional, el cual, para un conjunto finito de tiempos $t_1, t_2, t_3, ..., t_i, t_{i+1}, ...$, puede hacer lo siguiente.

Dado cualquier conjunto de valores de estado $y^i_1, ..., y^i_m$ en t_i, el programa calcula los valores únicos $y^{i+1}_1, ..., y^{i+1}_{m+1}, ..., y^{i+1}_n$ en t_{i+1} especificados por las reglas de interacción. Cuando esto sucede, decimos que el programa puede calcular o simular la *transición del modelo* de t_i a t_{i+1}.

El conjunto $\{t_1, t_2, ...\}$ es llamado *tiempos computacionales.* Son los tiempos del modelo en los cuales el programa puede producir el conjunto de valores de descripción del modelo. Algunas veces estos tiempos son múltiplos sucesivos de algún tiempo *h*, por lo que $t_{i+1} - t_i = h$. Esto es llamado una *simulación de tiempo discreto.* Se supone que las reglas de interacción no dependen del tiempo, sólo de los valores de estado $y_1, ..., y_m$. El modelo es llamado *invariante en el tiempo*.

Imagime que la simulación de un modelo bien definido es llevada a cabo por un programa que tiene acceso a las variables descriptivas $\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n$. Las siguientes propiedades explican el concepto de estado.

1. *Inicialización del programa*. Suponga que al programa se le da la tarea de calcular los valores $y'_1, ..., y'_n$ en t', dados los valores $y_1, ..., y_n$ en t. Entonces, solamente las variables de estado necesitan ser inicializadas.

2. *Repetir una ejecución*. Suponga que queremos repetir el cálculo de los valores $y'_1, ..., y'_n$ en t' dados los valores $y_1, ..., y_n$ en t, por ejemplo porque perdimos algún resultado de la primera

ejecución de la simulación. Las dos ejecuciones pueden ser hechas en diferentes computadoras en diferentes tiempos, y aún así los resultados van a ser los mismos, dados los mismos valores iniciales y_1, ..., y_m.

3. *Interrupción del programa y re-inicio*. Suponga que después de calcular los valores y'_1, ..., y'_n en t' se interrumpe el programa. El programa debe ser capaz de re-iniciar desde el mismo punto en el cual fue detenido, y debe calcular los mismos valores finales. Para esto basta guardar los valores asociados a α_1, α_2, α_3, ..., α_m en el momento en que se interrumpió el programa.

4. *Recuperación del programa*. Suponga que ocurre una falla en el computador mientras el programa está siendo ejecutado. Cuando la falla es reparada, se quiere seguir con la simulación, pero se quiere evitar re-inicializar el programa desde el estado inicial. La re-inicialización del programa debería obtener el mismo resultado si ésta se hace desde el inicio o desde el punto en que ocurrió la falla.

Procedimiento prototipo para simulación de tiempo discreto de modelos invariantes en el tiempo

Si se trata de calcular las variables de estado y_1, ..., y_n para instantes $t = t_M, t_{M+1}, ..., t_{M+N}$ (con $t_{i+1} = t + h$) y además las reglas no dependen del tiempo, sino sólo del estado anterior, estamos frente a un caso que llamaremos *de tiempo discreto e invariante en el tiempo*. El siguiente sería un plan a seguir para implementar un programa de simulación para estos casos.

1. Inicializar las variables de estado α_1, α_2, α_3, ..., α_n con y_1, y_2, y_3, ..., y_n.

2. Inicializar el reloj en t_M.

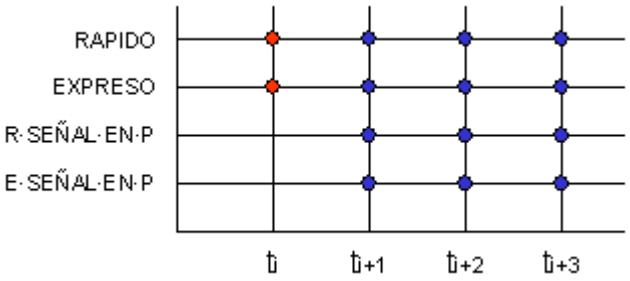
3. Aplicar las reglas de interacción al contenido de las variables de estado actuales para producir los valores correspondientes al siguiente estado. Calcular a partir de las variables de estado el resto de las variables descriptivas.

4. Avanzar el reloj en la unidad de tiempo h especificada en el modelo.

5. Ver si el reloj excedió el tiempo de simulación ($t_M + Nh$). Si excedió el tiempo, termina, si no, vuelve a 3.

Ejemplo:

La siguiente figura muestra el proceso de aplicar el paso 3. Los puntos rojos indican las variables de estado, y los puntos azules las variables descriptivas. A partir de las variables de estado es posible obtener las variables descriptivas para todos los tiempos siguientes.



Especificación formal de modelos

El paso 3 se puede ver como un subprograma que acepta una lista de valores de estado y entrega una lista de variables descriptivas actualizada. Este subprograma puede verse como una función (en sentido matemático) f cuyo dominio es el conjunto de todos los valores posibles de las variables de estado y como rango el conjunto de los valores posibles de todas las variables descriptivas, es decir:

$$f(y_1, ..., y_m) = (y'_1, ..., y'_m, y'_{m+1}, ..., y'_n)$$

donde $y_1, ..., y_m$ son los valores de las variables de estado y $y'_1, ..., y'_m, y'_{m+1}, ..., y'_n$ son los valores de las variables descriptivas una unidad de tiempo después.

Podemos decir que f está constituída de dos funciones:

$$y_1, ..., y_m \rightarrow \delta \rightarrow y'_1, ..., y'_m \rightarrow \lambda \rightarrow y'_1, ..., y'_m, y'_{m+1}, ..., y'_n$$

donde:

$$\delta(y_1, ..., y_m) = (y'_1, ..., y'_m)$$

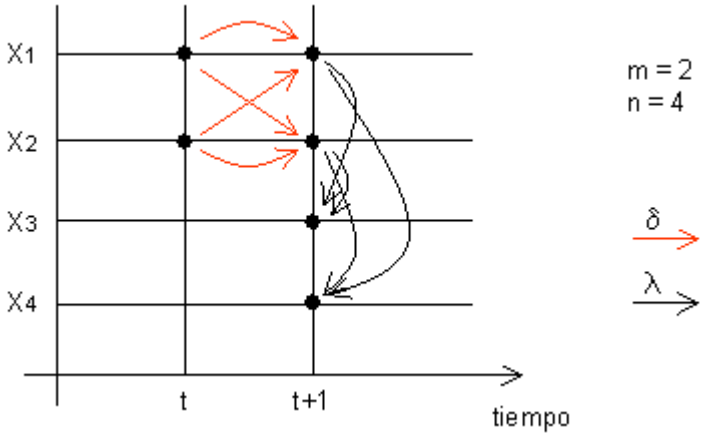
$$\lambda(y'_1, ..., y'_m) = (y'_1, ..., y'_n)$$

$$f(y_1, ..., y_m) = \lambda(\delta(y_1, ..., y_m))$$

Nótese que la función δ no es instantánea, es decir, los valores $y'_1, ..., y'_m$ son en el instante t+1 (si $y_1, ..., y_m$ en t) y también $y'_1, ..., y'_n$ (la salida de la función λ) son en ese mismo instante t+1.

δ toma el estado en que se encuentra el modelo en el instante actual y produce el estado en el que estará el modelo en el próximo instante de cálculo. δ se llama la *función de transición de estados*, o más corto, la *función de transición.*

λ toma el estado del modelo en un instante dado y produce la descripción total del modelo en ese estado. λ se llama *función de salida*.



En la práctica, podemos estar interesados en observar sólo algunas de las variables descriptivas (no todas), las que se denominan *variables de salida.* Se puede modificar la función λ de modo que produzca los valores de las variables de salida (que son una descripción parcial del modelo en su estado actual).

Formalicemos los estados y las salidas:

Sea VARIABLES el conjunto de variables descriptivas, VARIABLES·ESTADO el conjunto de las variables de estado, y VARIABLES·SALIDA el conjunto de variables de salida. Entonces:

$$\text{VARIABLES·ESTADO} \subseteq \text{VARIABLES}$$
$$\text{VARIABLES·SALIDA} \subseteq \text{VARIABLES}$$

Sea m el número de variables de estado. Entonces un *estado* del modelo es una tupla $(y_1, ..., y_m)$ donde cada y_i es un valor en el rango de la variable de estado i-ésima. El conjunto de todas estas tuplas se denomina ESTADOS.

Formalmente, se define como:

$$\text{ESTADOS} \subseteq X_{\beta \in \text{VARIABLES·ESTADO}} \text{RANGO·}\beta$$

Similarmente, si hay p variables de salida, una *salida* del modelo es una tupla $(y_1, ..., y_p)$, donde cada y_j es un valor en el rango de la variable de salida j-ésima. El conjunto de todas estas tuplas se denomina el conjunto de SALIDAS.

Formalmente, se define como:

$$\text{SALIDAS} \subseteq X_{\gamma \in \text{VARIABLES·SALIDA}} \text{RANGO·}\gamma$$

Con estas definiciones entonces:

δ: ESTADOS $\rightarrow$ ESTADOS y
λ: ESTADOS $\rightarrow$ SALIDAS

Podemos interpretar esto de la siguiente forma: si ESTADO es el estado del modelo en el instante t, entonces δ(ESTADO) es el estado del modelo en t+1, y λ(ESTADO) es la salida del modelo en el instante t.

Ejemplo: (caso de los trenes)

Habíamos visto que RAPIDO y EXPRESO (la ubicación de los trenes) eran un conjunto de variables de estado. El rango de cada una de estas variables es HITOS. Luego:

ESTADOS = HITOS x HITOS

y un ESTADO típico tiene la forma (r,e) donde r $\in$ HITOS y e $\in$ HITOS.

También vimos cómo calcular el estado en t+1 dado el estado en t. Escribamos la función de transición:

δ: ESTADOS $\rightarrow$ ESTADOS

tal que para cada (r,e) $\in$ ESTADOS:

δ(r,e) = (r + D(e - r), e + D(r - e))

porque r + D(e - r) es la ubicación del rápido en t+1 y e + D(r - e) es la ubicación del expreso en t+1.

Supongamos ahora que elegimos como variables de salida las lecturas de las señales que pueden hacerse en los hitos L y -L (los extremos del ferrocarril). Las variables de salida son entonces:

R·SEÑAL·EN·L
E·SEÑAL·EN·L
R·SEÑAL·EN·(-L)
E·SEÑAL·EN·(-L)

Ya que cada una de estas variables tiene rango {-1,1}, tenemos que:

SALIDAS = {-1,1} x {-1,1} x {-1,1} x {-1,1}

También habíamos calculado las lecturas de las señales en cualquier punto P en t, y que eran completamente determinadas por las ubicaciones de RAPIDO y EXPRESO en t. Así, podemos especificar δ:

δ: ESTADOS $\rightarrow$ SALIDAS

Por tanto, tomando en cuenta esos cálculos, para todo (r,e) $\in$ ESTADOS:

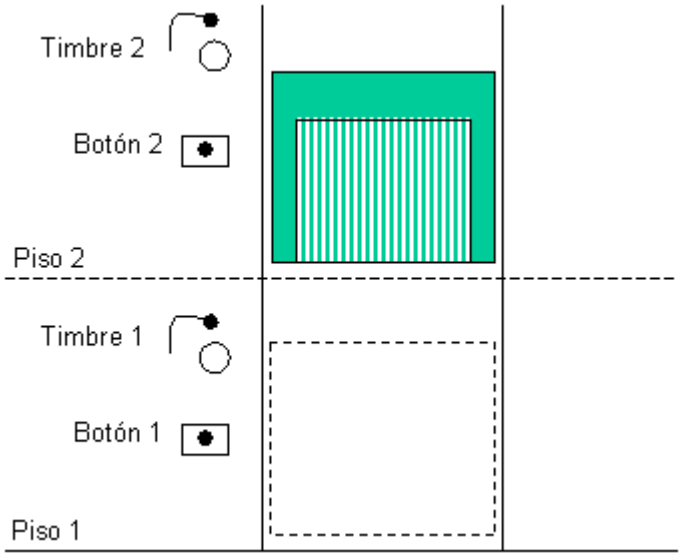
$\lambda(r,e) = (D(r - L), D(e - L), D(r + L), D(e + L))$

[anterior][home] [siguiente]

[anterior] [home] [siguiente]

1.3.3. Ejemplo 7 - Un montacargas simplificado

Se trata de un montacargas que puede viajar entre dos pisos. Tiene una puerta. En los pisos hay un botón para llamar al montacargas, y un timbre que suena cuando llega y abre su puerta. No hay botón interior, por lo que oralmente se debe pedir a alguien en el otro piso que presione el botón para hacer moverse el montacargas.



Se está modelando para ver cuán complejo es agregarle otros componentes.

Descripción informal

Componentes

MONTACARGAS, PISO·1, PISO·2.

Variables descriptivas

MONTACARGAS

PUERTA - con rango {ABIERTA, CERRADA}

LUGAR - con rango {UNO, DOS, SUBIENDO,BAJANDO}; LUGAR = x indica si el montacargas está en ese instante en un piso, subiendo o bajando.

RETARDO - con rango {0, 1, 2, 3}; RETARDO = y significa que hace y unidades de tiempo que se le pidió algo específico al montacargas.

Para PISO·i (i = 1, 2)

BOTON·i - con rango {APRETADO, NO·APRETADO}; BOTON·i = z indica si el botón en el piso i ha sido apretado o no en este instante.

TIMBRE·i - con rango {SONIDO, SILENCIO}.

Ejercicio: ¿Cuáles serán las variables de entrada?

Interacción entre componentes

1. El MONTACARGAS permanece indefinidamente en el piso al que llegó con su puerta ABIERTA, a menos que alguien presione un BOTON.

2. Si se presiona un BOTON y éste es del mismo PISO en el que está el MONTACARGAS, éste permanece en ese piso con la puerta ABIERTA durante las próximas 3 unidades de tiempo.

3. Si se presiona un BOTON y el MONTACARGAS no está con la puerta ABIERTA (está SUBIENDO o BAJANDO) no pasa nada.

4. Si se presionan ambos botones al mismo tiempo, tiene prioridad el botón del mismo piso en el que está el MONTACARGAS.

5. Si se presiona un BOTON y el MONTACARGAS está en el otro piso y con la puerta ABIERTA, en la próxima unidad de tiempo ésta se cierra, y en la siguiente unidad de tiempo viaja (sube o baja con la puerta CERRADA), demorando 2 unidades de tiempo adicional. Después abre la puerta.

6. Al llegar a un piso, se hace sonar el TIMBRE del piso correspondiente durante una unidad de tiempo.

Especificación de la máquina secuencial (autómata):

(ENTRADAS, ESTADOS, SALIDAS, δ, λ)

Veamos cada uno de estos elementos de la quíntupla:

(a) ENTRADAS:
Las variables de entrada son BOTON·1 y BOTON·2. Una entrada es entonces (b_1, b_2), con $b_1, b_2 \in$ {APRETADO, NO·APRETADO}. Por simplicidad, resumiremos APRETADO como A y NO·APRETADO como N.

Por ejemplo, si en el instante t el BOTON·1 está apretado y el BOTON·2 no lo está, el valor de las variables de entrada es (A,N). Así:

ENTRADAS = {(A,A), (A,N), (N,A), (N,N)}

Para especificar la función δ, algunas veces queremos decir que dá lo mismo el valor de un cierto botón. En este caso usamos i (indiferente).

(b) ESTADOS:
Las variables de estado son PUERTA, LUGAR, RETARDO (un conjunto minimal). Pronto mostraremos que son variables de estado. Luego, un estado será un trío (p,l,r), donde

p ∈ {ABIERTA, CERRADA}
l ∈ {UNO, DOS, SUBIENDO, BAJANDO}
r ∈ {0, 1, 2, 3}

Por tanto,

ESTADOS ⊆ {ABIERTA, CERRADA} x {UNO, DOS, SUBIENDO, BAJANDO} x {0, 1, 2, 3}

En realidad es un subconjunto propio porque hay estados imposibles, tales como (ABIERTA, SUBIENDO, i), ya que el montacargas sólo sube con la puerta cerrada.

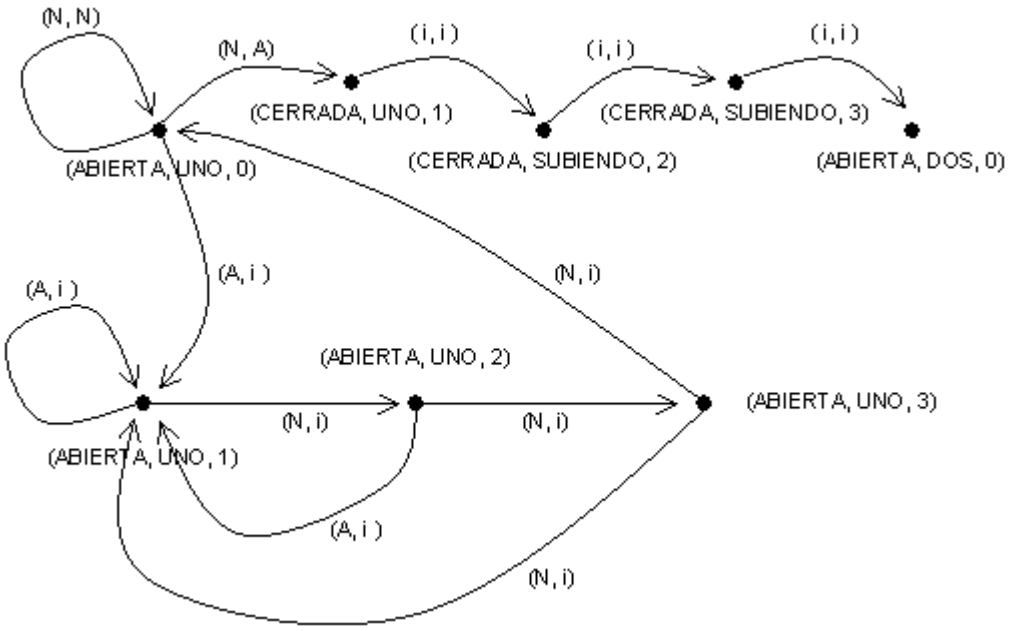
(c) SALIDAS:
Arbitrariamente elegimos las variables de salida. Digamos que son PUERTA, LUGAR, TIMBRE·1, TIMBRE·2. Por tanto,

SALIDAS ⊆ {ABIERTA, CERRADA} x {UNO, DOS, SUBIENDO, BAJANDO} x {SONIDO, SILENCIO} x {SONIDO, SILENCIO}

(d)δ:

Consideremos primero un estado inicial factible y vayamos construyendo las transiciones a partir de ahí. Un estado inicial factible es, por ejemplo, (ABIERTA, UNO, 0).

Hagamos un diagrama (grafo) en que los nodos son los estados, y los arcos son las entradas:



Ejercicio: Completar el diagrama de transiciones.

Con este diagrama es muy simple construir la tabla para

δ:ESTADOS x ENTRADAS → ESTADOS

estado	entrada	estado
PUERTA, LUGAR, RETARDO	BOTON·1, BOTON·2	PUERTA, LUGAR, RETARDO

ABIERTA, UNO, 0	N, N	ABIERTA, UNO, 0
ABIERTA, UNO, 0	A, i	ABIERTA, UNO, 1
ABIERTA, UNO, 0	N, A	CERRADA, UNO, 1
ABIERTA, UNO, 1	A, i	ABIERTA, UNO, 1
ABIERTA, UNO, 1	N, i	ABIERTA, UNO, 2
ABIERTA, UNO, 2	A, i	ABIERTA, UNO, 1
ABIERTA, UNO, 2	N, i	ABIERTA, UNO, 3
ABIERTA, UNO, 3	A, i	ABIERTA, UNO, 1
ABIERTA, UNO, 3	N, i	ABIERTA, UNO, 0
CERRADA, UNO, 1	i, i	CERRADA, SUBIENDO, 2
CERRADA, SUBIENDO, 2	i, i	CERRADA, SUBIENDO, 3
CERRADA, SUBIENDO, 3	i, i	ABIERTA, DOS, 0
...	...	...

Ejercicio: Completar la tabla de δ

(e)λ :

$\lambda(p, l, r, b_1, b_2) =$

$$\begin{cases} (p, l, r, \text{SILENCIO}, \text{SONIDO}) & \text{si } p = \text{CERRADA}, l = \text{SUBIENDO y } r = 3 \\ (p, l, r, \text{SONIDO}, \text{SILENCIO}) & \text{si } p = \text{CERRADA}, l = \text{BAJANDO y } r = 3 \\ (p, l, r, \text{SILENCIO}, \text{SILENCIO}) & \text{en otros casos} \end{cases}$$

Ejercicio: Comprobar que hemos demostrado que {PUERTA, LUGAR, RETARDO} es un conjunto de variables de estado.

Ejercicio: Incorporar en el prototipo de simulación, la lectura de variables de entrada, para dejarlo apropiado para modelos de tiempo discreto no-autónomos.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.4. Modelos de eventos discretos

Supongamos que los valores de las variables de entrada del ejemplo 7 (montacargas) se nos dan en una tabla:

t	BOTON·1	BOTON·2
0	N	N
1	N	N
2	N	N

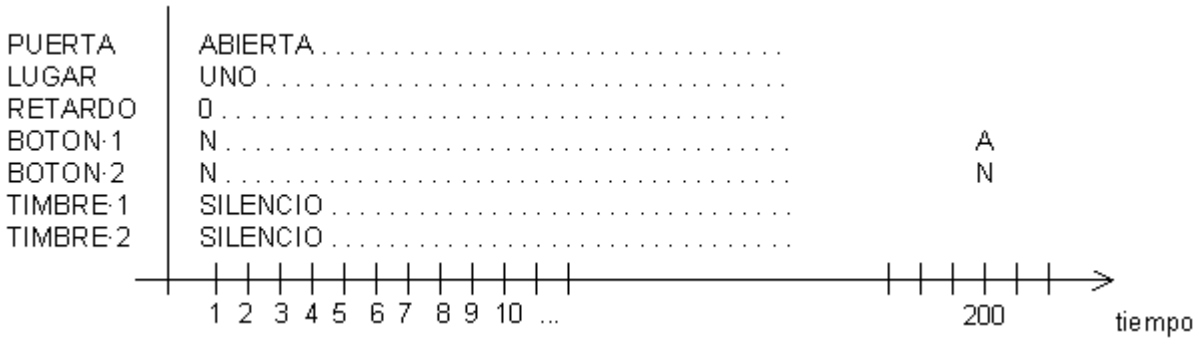
3	N	N
...	...	...
200	A	N
201	N	N
202	N	N
203	N	N
...	...	...
300	N	A
...	...	...
355	A	A

Con esto podemos hacer la simulación del montacargas!

¿Qué podemos decir de la eficiencia del programa?

Así como para almacenar una matriz poco densa hay maneras más eficientes de hacerlo, también para simular hay formas más eficientes de realizarlo que calculando, para cada incremento de t, el nuevo estado y el valor de las variables de salida.

Si hacemos un diagrama del valor de las variables:



Resulta claro en este caso que es un desperdicio hacer el cálculo de la transición de estados durante 200 iteraciones, cuando en realidad *nada significativo* ha ocurrido.

Sabemos que en el instante t = 200, por el contrario, hay algo interesante. Esto lo llamamos un *evento*.

Note que podríamos, en el instante t = 0, decidir avanzar directamente al instante t = 200, ya que nada ha cambiado en el modelo (salvo el tiempo...). Esto nos ahorarría tiempo de computador, sin duda.

En la mayor parte de los sistemas que en la práctica se simulan, ocurre algo parecido a lo que hemos descrito para el caso del montacargas. Por tanto, vamos a estudiar lo que se denomina *simulación de eventos discretos*.

Comencemos observando que tenemos una tabla de las entradas, que en el caso del montacargas es el punto de partida de los eventos. En general, para poder realizar simulación de eventos discretos, es necesario que los enventos sean *predecibles*, en el sentido de que el *instante* en que ellos van a ocurrir puede ser conocido de alguna manera. En sistemas verdaderos, en realidad el *instante de ocurrencia* de un evento no se conoce por una tabla, sino de otra forma que veremos más adelante.

Una segunda condición para realizar simulación de eventos discretos es que todo cambio de estado es una consecuencia de un evento pre-programado, o la secuela de otro cambio de estado que sí fue un evento pre-programado.

El modelo entonces avanza *a saltos*. Nada significativo ocurre *entre* los saltos.

Veamos ahora cómo se especifican los instantes de ocurrencia. La forma usual es mediante una *variable aleatoria.* Como sabemos, una variable aleatoria Y tiene asociada una función de distribución acumulada, y al *muestrearse* entrega un valor de su rango.

$Y : S \rightarrow R$ (en los números reales), donde S es el espacio muestral
$Y : S \rightarrow G$

En un programa de computador, *muestrear* una variable aleatoria consiste en llamar un subprograma que entrega como resultado un valor del rango de la variable aleatoria.

En el ejemplo del montacargas, podríamos tener una variable aleatoria APR·1, tal que si la muestreamos nos da el número de instantes, a partir del actual, en que el BOTON·1 va a tener valor "A".

Ejercicio: ¿Cuál será el rango de APR·1? ¿Es APR·1 discreta o continua?

Similarmente con APR·2.

Antes de que veamos un ejemplo, necesitamos algoo de notación:

R^+ : números reales positivos
R^+_0 : números reales positivos y cero
$R^+_{0,\infty}$: números reales positivos, cero e infinito ($\infty + r = \infty$ para todo r en R)

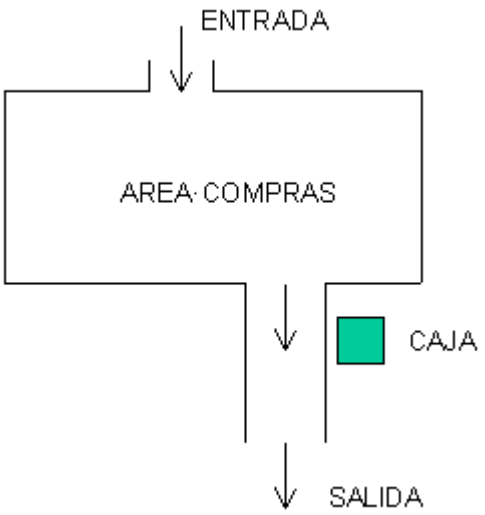
Si A es un conjunto, denotamos como A* al conjunto de todas las secuencias finitas de A. Una secuencia típica de A* es $a_1, a_2, ..., a_n$, donde cada $a_i \in A$. Esta secuencia se dice que tiene largo n. Cuando n = 0, tenemos la secuencia vacía.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.4.1. Ejemplo 8 - Un supermercado simplificado

Los clientes que entran a un supermercado tienen nombres sacados del conjunto {a, b, ..., z}. Un cliente que entra, pasa al AREA·COMPRAS, donde selecciona sus adquisiciones. Cuando termina se va a la caja, donde espera en COLA. Después de pagar al cajero, el cliente se va por la SALIDA.



Observemos que una cola en la caja puede representarse como $x_1, x_2, ..., x_n$, con $x_i \in \{a, b, c, ...\}$. Es decir, una cola específica es un elemento de {a, b, c, ...}*.

Descripción

Componentes

ENTRADA, AREA·COMPRAS, CAJA, SALIDA.

Variables descriptivas

ENTRADA

HOLA - con rango {∅, a, b, c, ...}; HOLA = ∅ indica que no hay clientes en la entrada. HOLA = x significa que el cliente x está en la ENTRADA.

AREA·COMPRAS

T·COMPRAS - con rango R^+; una variable aleatoria que da el iempo que demora un cliente en hacer sus adquisiciones en el AREA·COMPRAS.

LISTA·CLIENTES - con rango $(\{a, b, ...\} \times R^+)^*$. $(x_1, \tau_1), (x_2, \tau_2), ..., (x_n, \tau_n)$ significa que el cliente x_i saldrá del AREA·COMPRAS en τ_i unidades de tiempo a partir de ahora.

CAJA

COLA - con rango {a, b, c, ...}*. COLA = $x_1, x_2, ..., x_n$ indica que x_1 está primero en la cola, x_2 es segundo, etc.

T·SERVICIO - con rango R^+. Variable aleatoria que da el tiempo en el que será procesado el cliente que está primero en la cola.

SERVICIO·RESTANTE - con rango R^+. SERVICIO·RESTANTE = σ significa que el cliente que está siendo atendido, dejará la CAJA en σ unidades de tiempo a partir de ahora.

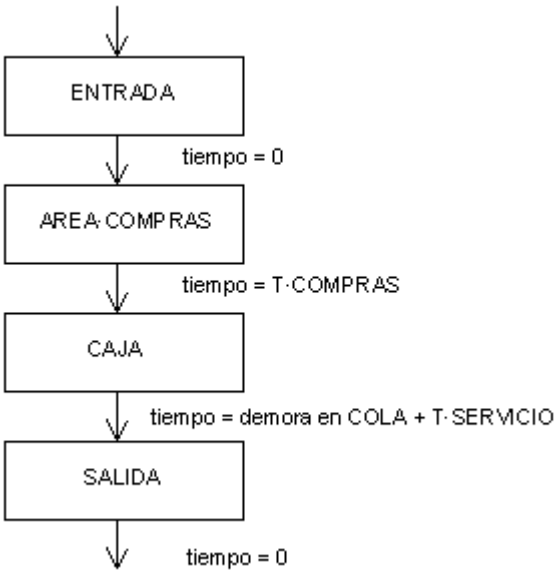
OCUPADO - con rango {SI, NO}; indica si la CAJA está atendiendo o no un cliente.

SALIDA

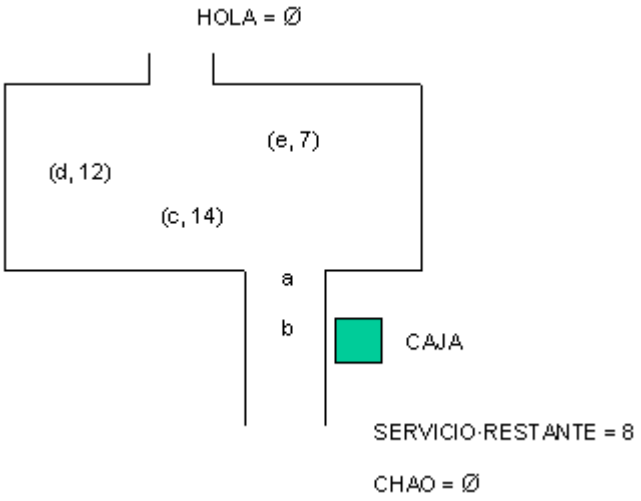
CHAO - con rango {∅, a, b, c, ...}; CHAO = ∅ indica que no hay clientes en la salida. CHAO = x significa que el cliente x está yéndose.

Interacción entre componentes

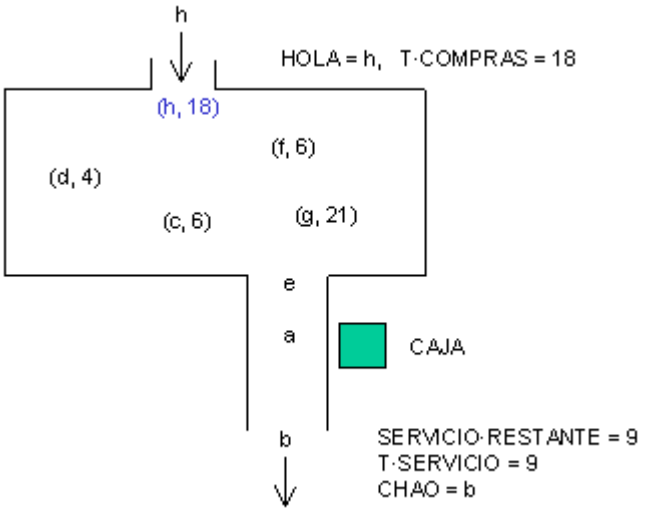
Cuando un cliente x llega a la ENTRADA en t, su presencia es indicada por HOLA = x. Entra al área de compras (HOLA se pone en ∅) y después de muestrear T·COMPRAS, obtiene un tiempo de adquisiciones τ. Por tanto, (x, τ) es puesto en la lista LISTA·CLIENTES. A medida que el reloj avanza, (x, τ) es decrementado hasta que (x, 0) esté en la lista. En ese instante, el cliente x deja el AREA·COMPRAS, y se pone al final de la COLA enla caja. A medida que los clientes son procesados, avanza hacia el frente de la COLA. Cuando es primero, muestrea T·SERVICIO para obtener un tiempo σ y se asigna σ a SERVICIO·RESTANTE. El cliente x espera en la cabeza de la COLA hasta que SERVICIO·RESTANTE se hace 0. En ese momento se va de la CAJA (y del supermercado). Su paso por la SALIDA (que toma tiempo 0) se señala por CHAO = x.



En un instante dado t, por ejemplo, la situación puede ser la siguiente:



En el instante t + 8 la situación podría ser la siguiente (por ejemplo):



[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

1.4.2. Instantes de ocurrencia

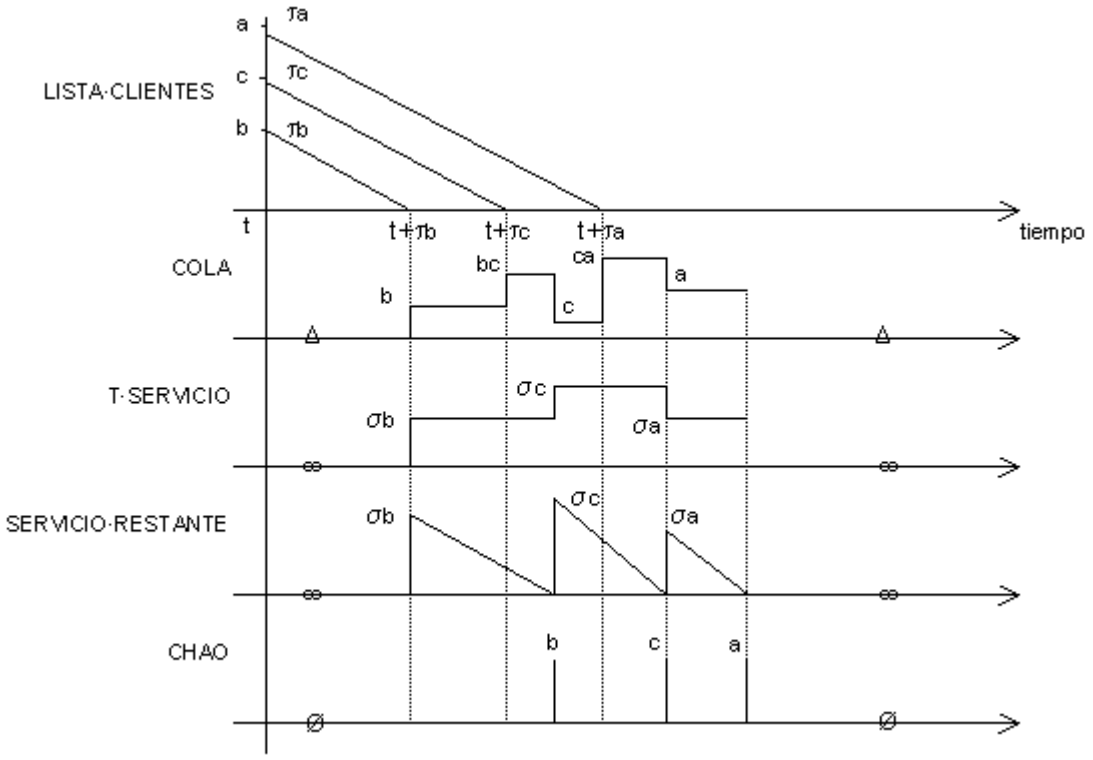
Con el ejemplo anterior (ejemplo 8) podemos clarificar las ideas de simulación de eventos discretos. En lugar de avanzar el tiempo del modelo como t, t+1, t+2, ..., si en un momento determinado el tiempo del modelo (también decimos el *reloj* del modelo) es t, podemos avanzar el tiempo al próximo *instante de ocurrencia.*

Pensemos, para empezar, que no hay llegada de clientes (es decir, un modelo autónomo). Esto puede ocurrir en la noche, cuando el supermercado ha cerrado la ENTRADA. Entonces, sea t el instante actual, t' el próximo instante de ocurrencia.

$$t' = t + \min \{\tau_1, \tau_2, ..., \tau_n, \sigma\}$$

Es decir, t' será el instante actual más el mínimo de los tiempos que quedan para que se produzca un

evento. Los τ_1, τ_2, ..., τ_n y σ son relojes decrecientes(timers): cuando su valor se hace 0, *ocurre* un evento. ¿Cómo será la evolución de las variables durante el tiempo? Veamos:



Ejercicio: Hacer el diagrama para la variable OCUPADO.

Ejercicio: Poner, en el eje del tiempo, para cada variable, los instantes de ocurrencia.

Ejercicio: Para el ejemplo, liste los eventos (donde ocurren) y los instantes de ocurrencia correspondientes, ordenadamente en el tiempo.

Examinemos lo que ocurre *entre* dos instantes de ocurrencia. Por ejemplo, entre $(t + \tau_b)$ y $(t + \tau_c)$:

- La variable COLA permanece constante (con valor b).

- La variable T·SERVICIO no es muestreada: permanece constante con el último valor de σ_b.

- La variable SERVICIO·RESTANTE decae linealmente.

- La variable CHAO permanece constante (valor $\varnothing$).

En general, entre dos instantes de ocurrencia cualesquiera sucede lo mismo. Por tanto, el *tiempo* se puede hacer saltar entre instantes de ocurrencia, y esto es la base de la simulación de eventos discretos.

Supongamos que el subprograma que entrega una muestra de la variable aleatoria T·SERVICIO *necesite* el valor anterior que ella misma generó para dar la muestra. Con este supuesto y recordando que la ENTRADA está cerrada, las variables de estado son:

LISTA·CLIENTES, con valor típico (x, τ_1), ..., (x, τ_2).

COLA, con valor típico y_1, ..., y_n.

T·SERVICIO, con valor típico r.

SERVICIO·RESTANTE, con valor típico σ.

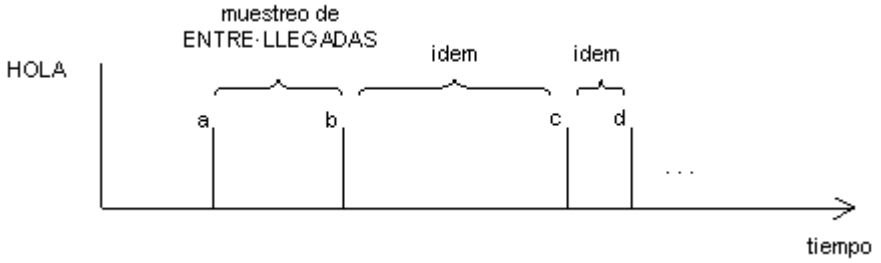
Luego, los *estados* son:

$S = ((x_1, \tau_1), y_1, r, \sigma)$
$S = ((x_2, \tau_2), y_1, r, \sigma)$
...

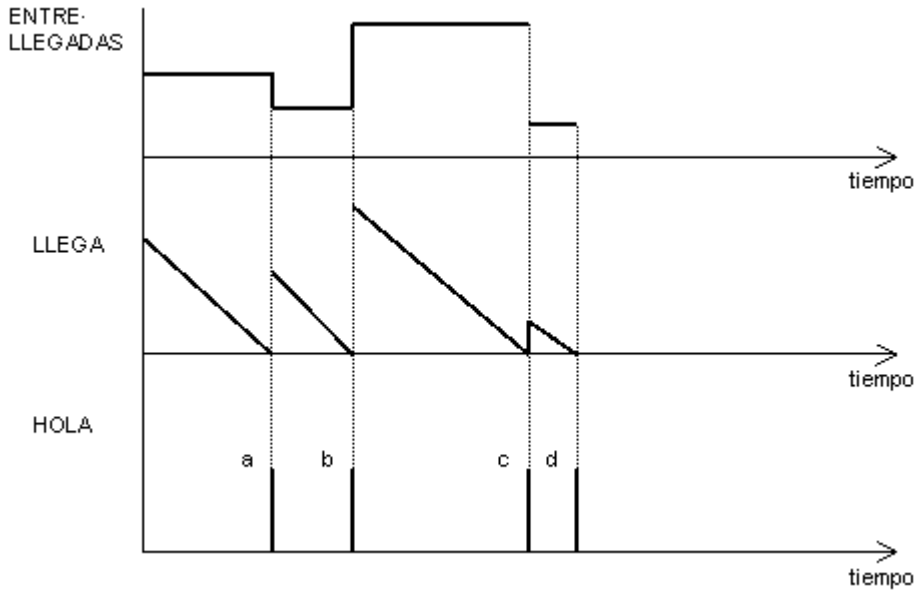
Veamos ahora el caso no-autónomo.

La llegada de clientes es un fenómeno *externo* al modelo, pero también puede modelarse. En un sentido purista, alguien podría afirmar que el modelo, en ese caso, sigue siendo autónomo...

En muchos casos, la llegada de clientes, o mejor dicho, el periodo de tiempo entre llegadas de clientes, puede modelarse por una variable aleatoria, con una distribución acumulada conocida. Sea ENTRE·LLEGADAS esta variable (es decir, si muestreamos ENTRE·LLEGADAS obtenemos la cantidad de tiempo que hay que esperar para que llegue un nuevo cliente a partir del último).



Podemos crear una nueva variable LLEGA, que se inicializa con muestreo de ENTRE·LLEGADAS, y que va decreciendo con el tiempo, de modo que cuandose hace 0, aparece el nuevo cliente, se muestrea ENTRE·LLEGADAS, se asigna a LLEGA, etc. Por tanto, las variables quedan:



Finalmente, a cada cliente que llega se le asigna un tiempo de compras, muestreando la variable aleatoria T·COMPRAS. Por tanto, todos los eventos son predecibles, que era una de las condiciones de este tipo de simulación.

¿Cuál es ahora el próximo instante de ocurrencia? Si el valor en t de LLEGA es γ, entonces:

$$t' = t + \min \{\tau_1, \tau_2, ..., \tau_n, \sigma, \gamma\}$$

Ejercicio: Describa todo lo que ocurre con un cliente que llega (cuando γ se hace 0).

Ahora estamos en condiciones de ver el procedimiento de eventos discretos.

Necesitamos una estructura de datos para la simulación de eventos discretos: la lista PROXIMOS·EVENTOS. Se trata de una lista ordenada de menor a mayor por el instante de ocurrencia:

PROXIMOS·EVENTOS

COMPONENTE	INSTANTE
EVENTO·DE·α	t_α
EVENTO·DE·β	t_β
EVENTO·DE·δ	t_δ
...	...

Donde $t_\alpha <= t_\beta <= t_\delta$

El procedimiento es el siguiente:

* *inicialización*

1. Poner variable RELOJ con el tiempo de modelo inicial t_0.

2. Poner variables S_1, S_2, ..., S_p con los valores iniciales de las variables de estado (s_1, s_2, ..., s_p).

3. Para las componentes α que tengan un evento, poner el par (EVENTO·DE·α, t_α) en la lista PROXIMOS·EVENTOS, en que t_α es el instante de ocurrencia calculado. (Ordenar lista).

* *avance del tiempo*

4. Poner el RELOJ con el t_α del primer par en la lista PROXIMOS·EVENTOS.

* *transición*

5. Realice la transición de estados, extrayendo de la lista PROXIMOS·EVENTOS todos los pares que tengan instante igual a RELOJ, re-programando componentes, cambiando variables, etc. En esta etapa se re-colocan pares (EVENTO·DE·α, t_α) en la lista PROXIMOS·EVENTOS, cuidando de ponerlos en el orden apropiado.

* *¿todavía hay una transición con este valor de RELOJ?*

6. Si el valor de RELOJ iguala el instante del primer par, vaya a 5.

* *chequeo de término*

7. Si el valor de RELOJ iguala o excede el instante de término t_1, PARE. En caso contrario, vaya a 4.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.1. Introducción a Bases de Datos

El almacenamiento, manipulación y recuperación de información en forma eficiente, es vital y estratégico para cualquier organización. Las bases de datos juegan un rol crítico en casi todas las áreas donde las computadoras son usadas, incluyendo negocios, ingeniería, medicina, leyes, educación, etc.

Una *Base de Datos* (BD) es una colección de datos relacionados que representa un cierto modelo o abstracción del mundo real (algunas veces llamado el mini-mundo). Los cambios en este mini-mundo son reflejados en la base de datos. Una base de datos es diseñada, construida y llenada con datos para un propósito específico. Tiene un grupo de usuarios particular, y algunas aplicaciones pre-establecidas en las cuales estos usuarios están interesados.

En otras palabras, una base de datos tiene alguna "fuente" de la cual los datos son derivados, algún grado de interacción con eventos en el mundo real, y una audiencia que está activamente interesada en el contenido de la base de datos.

Un *Sistema Administrador de Base de Datos* (SABD) es una colección de programas que permite a los usuarios crear y mantener una base de datos.

La importancia de almacenar, manipular y recuperar la información en forma eficiente ha llevado al desarrollo de una teoría esencial para las bases de datos. Esta teoría ayuda al diseño de bases de datos y procesamiento eficiente de consultas por parte de los usuarios.

EL uso de las BD es contrario al enfoque tradicional, en que cada sistema maneja sus propios datos y archivos. Al usar BD, todos los datos se almacenan en forma integrada, y están sujetos a un control centralizado. Las diversas aplicaciones operan sobre este conjunto de datos.

Consideremos el siguiente ejemplo: Se desea almacenar la información de los alumnos de esta escuela, con los ramos que inscriben cada semestre y sus respectivas notas. Estos datos se pueden almacenar en una matriz, fijando un máximo de cursos que pueden inscribir cada semestre. La siguiente tabla muestra el enfoque tradicional.

Matricula	Nombre	Semestre	Curso 1	Nota 1	Curso 2	Nota 2	Curso 3	Nota3	...
654564	Juan Pérez	97/1	FI21A	4.5	MA22A	4.2	QI21A	4.3	
353090	Luis González	97/1	FI33A	R	MA34A	4,6	CC30A	5.5	
672680	José Tapia	97/1	FI10A	*	CC10A	*	MA11A	*	
...	...								
654564	Juan Pérez	97/2	MA33A	R	SD20A	5.7	MA26A	E	
353090	Luis González	97/2	CC31A	6,2	MA37A	4,5	CC31B	R	
672680	José Tapia	97/2	FI10A	R	CC10A	R	MA11A	4.2	
...									

Se pueden observar las siguientes desventajas:

· Se repiten varios datos, como el nombre del alumno, su número de matrícula, etc.
· Si se desea modificar el nombre de una persona, entonces se debe buscar en toda la matriz.
· Para obtener el BIA de una persona, o saber cuanta gente inscribe un curso cada semestre, debo leer secuencialmente todo el archivo.
· Etc.

Cuando se tienen pocos datos no es mucha la perdida de tiempo y espacio, pero cuando hablamos de cientos de miles de datos, o peor aún, millones de datos, nos enfrentamos a un serio problema. Esta redundancia al definir y almacenar los datos implica espacio de almacenamiento desperdiciado y esfuerzos redundantes para mantener actualizados los datos.

En el enfoque de bases de datos se mantiene un único almacén de datos que se define una sola vez y al cual tienen acceso muchos usuarios. Las principales ventajas sobre el enfoque tradicional son:

· Evita los datos repetidos (redundancia).
· Evita que distintas copias de un dato tengan valores distintos (inconsistencia).
· Evita que usuarios no autorizados accedan a los datos (seguridad).
· Protege los datos contra valores no permitidos (integridad o restricciones de consistencia).
· Permite que uno o más usuarios puedan accesar simultáneamente a los datos (concurrencia).

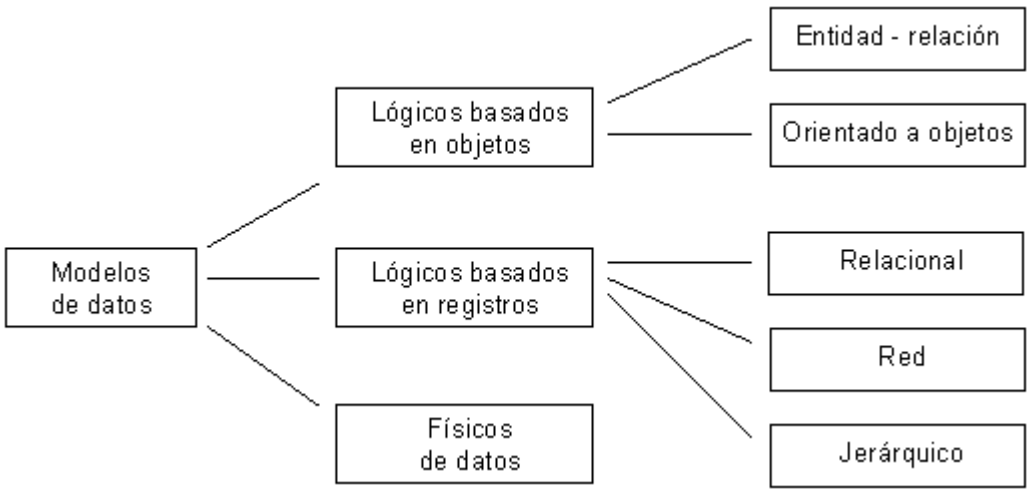
Modelos de datos

Una característica fundamental del enfoque de BD es que proporciona cierto nivel de abstracción de los datos, al ocultar detalles de almacenamiento que la mayoría de los usuarios no necesitan conocer. Los modelos de datos son el principal instrumento para ofrecer dicha abstracción.

Un *modelo de datos* es un conjunto de conceptos que pueden ser usados para describir la estructura de una BD. Con el concepto de *estructura de una BD* nos referimos a los tipos de datos, las relaciones y las restricciones que deben cumplirse para esos datos. Por lo general, los modelos de datos contienen además un conjunto de operaciones básicas para especificar lecturas y actualizaciones de la base de datos.

Los principales objetivos del proceso de modelamiento es saber identificar cuál es el problema y encontrar la forma de representarlo en un sistema. Esto significa saber de los datos, saber quienes van a usarlos y como van a ser usados.

La siguiente figura muestra los tipos de modelos de datos.



[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.2. Modelo Entidad-Relación

El modelo Entidad-Relación (ER) es uno de los modelos de datos más populares. Se basa en una representación del mundo real en que los datos se describen como entidades, relaciones y atibutos. Este modelo se desarrollo para facilitar el diseño de las bases de datos, y fue presentado por Chen en 1976.

El principal concepto del modelo ER es la *entidad*, que es una "cosa" en el mundo real con existencia independiente. Una entidad puede ser un objeto físico (una persona, un auto, una casa o un empleado) o un objeto conceptual (una compañía, un puesto de trabajo o un curso universitario). En nuestro ejemplo de la sección anterior podemos definir dos entidades: alumnos y cursos.

Cada entidad tiene propiedades específicas, llamadas *atributos*, que la describen. Por ejemplo, una sala de clases tiene un nombre (19S, F20), una ubicación, un cupo máximo, etc. En nuestro ejemplo, la entidad "alumno" posee los atributos nombre y matrícula. Una entidad particular tiene un valor para cada uno de sus atributos.

Cada uno de los atributos de una entidad posee un dominio, el que corresponde al tipo del atributo. Por ejemplo, "matrícula" tiene como dominio al conjunto de los enteros positivos y "nombre" tiene

como dominio al conjunto de caracteres.

Para todo conjunto de valores de una entidad, debe existir un atributo o combinación de atributos, que identifique a cada entidad en forma única. Este atributo o combinación de atributos se denomina llave (pimaria). Por ejemplo, el número de matricula es una buena llave para la entidad alumno, no así el nombre, porque pueden existir dos personas con el mismo nombre.

Una *relación* se puede definir como una asociación entre entidades. Por ejemplo, la entidad "libro" puede estar relacionada con la entidad "persona" por medio de la relación "está pedido". La entidad "alumno" puede estar relacionada con la entidad "curso" por la relación "está inscrito". Una relación también puede tener atributos. Por ejemplo, la relación "está inscrito" puede tener los atributos "semestre" y "nota de aprobación".

Ejemplo:
Suponga que estamos modelando los datos de una COMPAÑIA. La base de datos COMPAÑIA debe mantener información sobre los empleados de la compañía, los departamentos y los proyectos. La descripción del mini-mundo (la parte de la compañía a ser representada en la base de datos) es la siguiente:

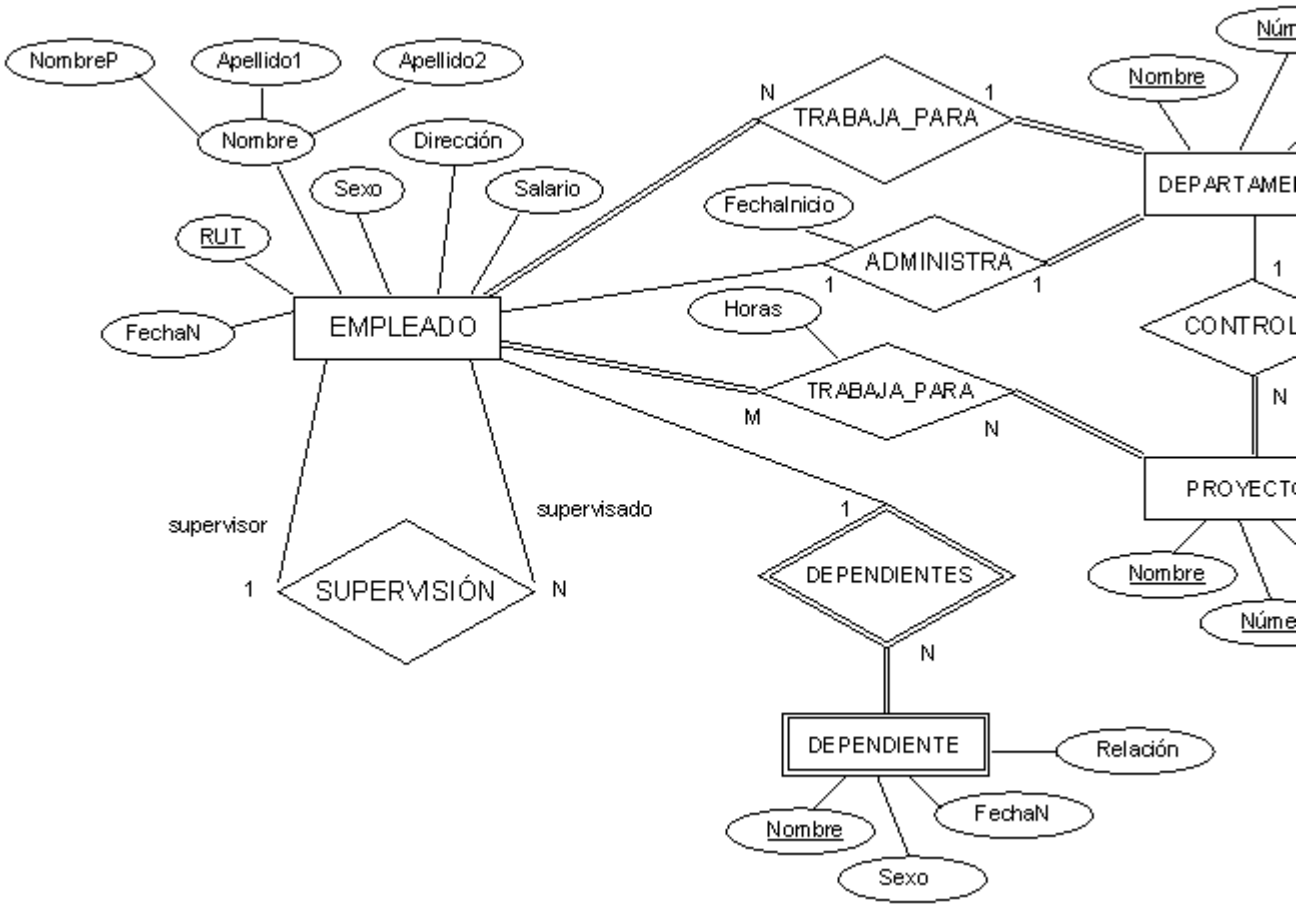
1. La compañia está organizada en departamentos. Cada departamento tiene un nombre único. un número único, y un empleado particular quien lo administra. Se quiere saber la fecha en que el empleado administrador empezó a hacerse cargo del departamento. Un departamento puede tener varios locales.

2. Cada departamento controla un cierto número de proyectos. Cada proyecto tiene un nombre y número únicos, y un local.

3. Para cada empleado se desea tener su nombre, rut, dirección, salario, sexo y año de nacimiento. Un empleado es asignado a un departamento, pero puede trabajar en varios proyectos, los que no son necesariamente controlados por el mismo departamento. Se quiere saber el número de horas semanales que un empleado trabaja en cada proyecto. Se quiere además saber cuál es el supervisor directo de cada empleado.

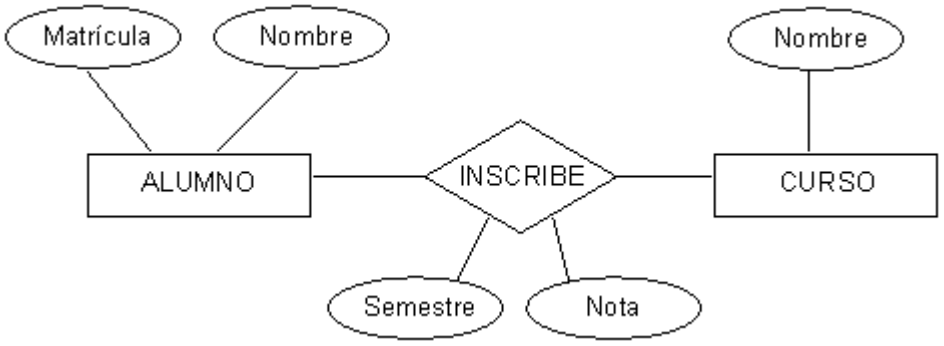
4. Se desea conocer las personas dependientes de cada empleado para propósitos de seguros. De cada dependiente se desea conocer el nombre, sexo, fecha de nacimiento y relación con el empleado.

La siguiente figura muestra el esquema de esta base de datos, a través de una notación gráfica llamada *diagrama ER.*



En este diagrama los rectángulos representan conjuntos de entidades, los elipses representan atributos y los rombos representan conjuntos de relaciones.

Usando esta notación, podemos ahora hacer el diagrama E-R del ejemplo anterior de los alumnos y los cursos matriculados.



[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.2.1. Tipos de relaciones

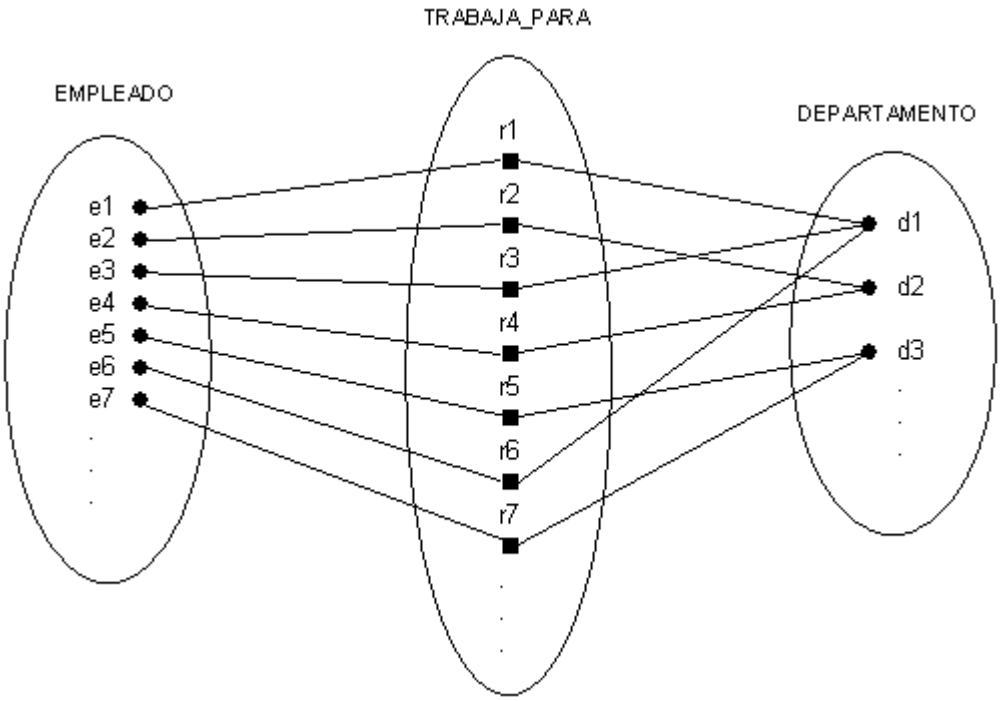
Un tipo de relación R entre n tipos de entidades E_1, ..., E_n define un conjunto de asociaciones entre estos tipos.

Puede ser visto como un conjunto de *instancias de la relación* r_i, donde cada r_i asocia n entidades

(e_1, ..., e_n), y cada entidad e_j en r_i es un miembro del tipo de entidad E_j ($1 <= j <= n$).

Un tipo de relación es un subconjunto del producto cartesiano E_1 x E_2 x ... x E_n.

Ejemplo. Algunas instancias de la relación TRABAJA_PARA del ejemplo anterior, podrían ser las siguientes.



Un tipo de relación podría también interpretarse como un conjunto de pares ordenados, en este caso: (e1, d1), (e2, d2), (e3, d1), (e4, d2), (e5, d3), (e6, d1), (e7, d3).

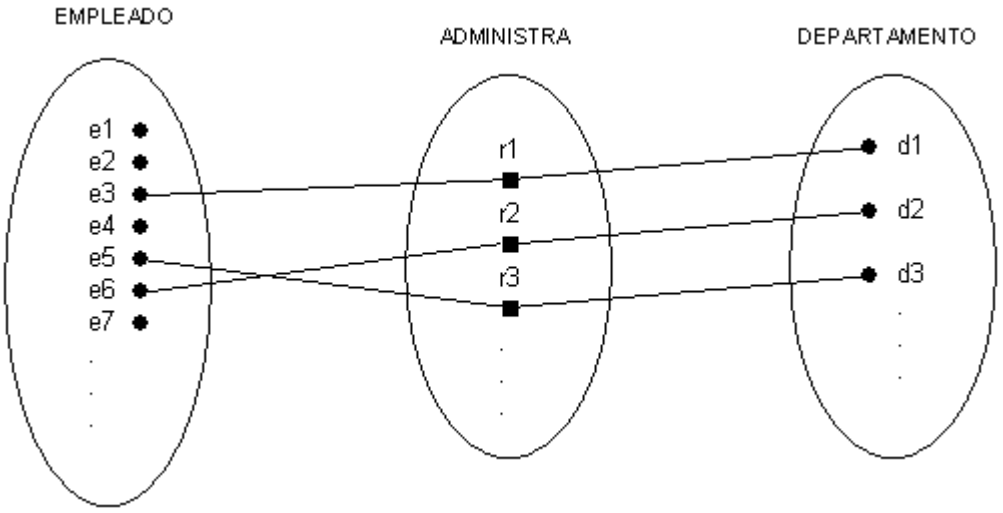
Según el número de entidades relacionadas (o *razón de cardinalidad*), se pueden definir tres tipos de relaciones:

1. Relaciones Uno a Uno (1:1). Una entidad A está asociada a lo más con una entidad B, y una entidad B a lo más con una entidad A. Ejemplo: "Ser jefe de" es una relación 1:1 entre las entidades empleado y departamento.

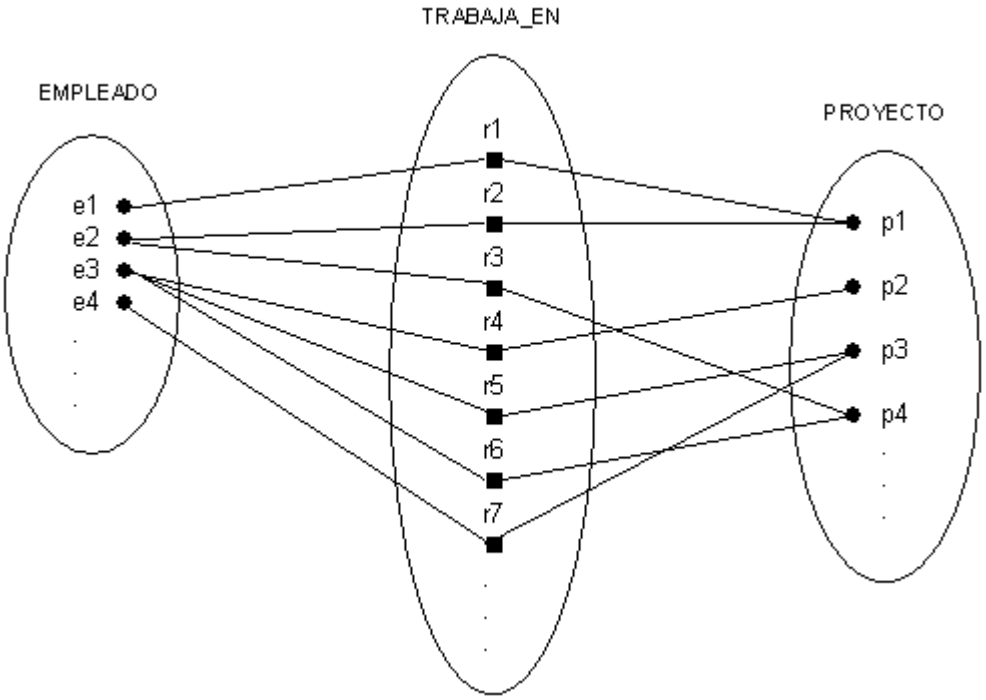
2. Relaciones Uno a Muchos (1:n). Una entidad A está asociada con una o varias entidades B. Una entidad B, sin embargo, puede estar a lo más asociada con una entidad A. Ejemplo: "Ser profesor" es una relación 1:n entre profesor y curso, suponiendo que un curso sólo lo dicta un profesor.

3. Relaciones Muchos a Muchos (n:m). Una entidad A está asociada con una o varias entidades B, y una entidad B está asociada con una o varias entidades A. Ejemplo: "Estar inscrito" es una relación n:m entre las entidades alumno y curso.

El siguiente es un ejemplo de la relación ADMINISTRA, con participación parcial de EMPLEADOS, y participación total de DEPARTAMENTOS.



La siguiente figura muestra un ejemplo de la relación M:N TRABAJA_PARA.

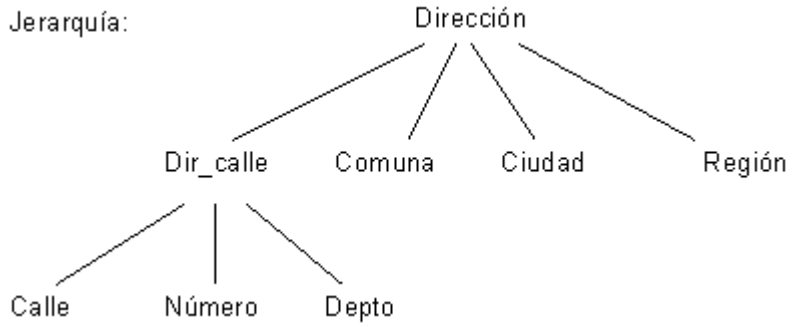


[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.2.2. Tipos de atributos

Los *atributos compuestos* se pueden dividir en sub-partes más pequeñas, que representan atributos más básicos con significados propios. Por ejemplo, una "dirección" puede sub-dividirse en: dir-calle, comuna, ciudad, región. Ejemplo:



Los atributos no sub-dividibles se llaman *atómicos* o *simples*. Si no hay necesidad de referirse a los elementos individuales de una dirección, entonces la dirección completa puede considerarse un atributo simple.

Atributos de valor simple son los que tienen un sólo valor para una entidad particular. Por ejemplo: edad.

Atributos multivalorados pueden tener un conjunto de valores para una misma entidad. Por ejemplo: "títulos profesionales" (una persona puede no tener ninguno, uno, dos o más).

En algunos casos una entidad particular puede no tener valores aplicables para un atributo. Ejemplo: "depto". Para estas situaciones tenemos un valor especial llamado *nulo*. También, si no se conoce el valor.

Un *tipo de entidad* define un conjunto de entidades con los mismos atributos. Ejemplo:

Nombre del tipo de entidad: EMPLEADO
Atributos: Nombre, Edad, Sueldo
Conjunto de entidades: (Juan Pérez, 55, 800.000), (Federico Pardo, 40, 550.000), (Rodrigo Pozo, 25, 400.000).

En los diagramas E-R, un tipo de entidad se representa como una caja rectangular, los nombres de los atributos como elipses y las relaciones como rombos. Los atributos multivalorados se representan con elipses dobles.

Un tipo de atributo usualmente tiene un atributo cuyos valores son distintos para cada entidad individual (atributo *clave* o *llave*) y sus valores se usan para identificar cada entidad unívocamente. Para una entidad tipo PERSONA, un atributo clave típico es el RUT. Algunas veces, varios atributos juntos forman una clave (la combinación debe ser distinta). Estos atributos clave aparecen subrayados en los diagramas.

Cada atributo simple tiene un *conjunto de valores* o *dominio* asociado, que especifica el conjunto de valores que puede asignarse a cada entidad individual. Por ejemplo, si las edades de los empleados pueden variar entre 16 y 70, entonces el dominio de Edad es $\{x \in N / 16 <= x <= 70\}$. Los dominios no se muestran en los diagramas.

Un atributo A del tipo de entidad E cuyo dominio es V, puede definirse como una función de E al conjunto potencia V (conjunto de todos los subconjuntos de V):

$$A: E \rightarrow P(V)$$

El valor del atributo A para la entidad e es A(e). Un valor nulo se representa por el conjunto vacío.

Para un atributo compuesto A, el dominio V es el producto cartesiano de $P(V_1)$, ..., $P(V_n)$ donde V_1, ..., V_n son los dominios de los atributos simples que forman A:

$$V = P(V_1) \times P(V_2) \times ... \times P(V_n).$$

Notemos que atributos compuestos y multivalorados pueden ser anidados de cualquier manera. Podemos representar anidamiento agrupando componentes de un atributo compuesto entre paréntesis ( ), separando componentes con comas, y mostrando atributos multivalorados entre llaves { }.

Ejemplo: Si una persona puede tener más de una dirección, y en cada una de ellas hay múltiples teléfonos, podemos especificar un atributo DirTel para una PERSONA así:

> { DirTel ({ Teléfono (CodigoArea, NumTel) }, Dirección (DirCalle (Calle, Número, NumDepto), Comuna, Ciudad, Región)) }

La persona Juan Pérez puede tener una instancia de este atributo así:

> { DirTel ({ Teléfono (2, 442-2855) }, Dirección (DirCalle (Blanco, 2120, nulo), Santiago, Santiago, RM)), DirTel ({ Teléfono (2, 241-3416) }, Dirección (DirCalle (Manuel Montt, 74, 201), Providencia, Santiago, RM)) }

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.3. El modelo relacional

Este modelo considera la Base de Datos (BD) como una colección de relaciones. De manera simple, una relación representa una *tabla*, en que cada fila representa una colección de valores que describen una entidad del mundo real. Cada fila se denomina *tupla*.

Dominios, tuplas, atributos, relaciones

Def. Un *dominio* D es un conjunto de valores atómicos. Atómico quiere decir que cada valor en el dominio es indivisible. Es útil dar nombres a los dominios. Ejemplo:

> Números-telefónicos-locales: el conjunto de número de teléfono de 7 dígitos.
> RUTs: números de 8 dígitos más un caracter que puede ser del 0 al 9 o K
> Nombres: el conjunto de nombres de personas Notas: valores entre 1.0 y 7.0

También se puede especificar un tipo de datos o formato para cada dominio. Un *schema de relación* R, denotado $R(A_1, A_2, ..., A_n)$ está constituido por un nombre de relación R y una lista de atributos $A_1, ..., A_n$. Cada atributo A_i es el nombre de un rol jugado por el dominio D en el schema de la relación R.

D se llama el dominio de A_i y se denota $dom(A_i)$. Un schema relacional se usa para describir una relación. R es el nombre de esta relación. El *grado de una relación* es el número n de atributos del schema de la relación.

Ejemplos:

ESTUDIANTE(Nombre, Rut, Teléfono, Dirección, Edad, Carrera, Prom-nota) tiene grado 7.
dom(Nombre) = Nombres
dom(Teléfono) = Números-telefónicos-locales
etc.

Def. Una *relación* o *instancia de relación* r del schema de relación $R(A_1, A_2, ..., A_n)$, denotado también como r(R) es un conjunto de n-tuplas $r = \{t_1, t_2, ..., t_m\}$. Cada n-tupla t es una lista ordenada de n valores $t = <v_1, ..., v_n>$, donde cada valor v_i, i <= i <= n, es un elemento de $dom(A_i)$ o es un valor nulo.

Ejemplo:

ESTUDIANTE	Nombre	Rut	Teléfono	Dirección	Edad	Carrera	Prom-nota
	Benjamín González	13.245.622-1	224-4211	Rosas 3241	19	Plan común	4.8
	Sergio Soto	12.341.228-5	nulo	Gay2142	20	Ing. Ind.	5.1
	...	...	...	...	...	...	...

Cada tupla representa una entidad de estudiante en particular. La definición de relación puede replantearse así: Una relación r(R) es un subconjunto del producto cartesiano de los dominios que definen r:

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times ... \times dom(A_n))$$

El número total de tuplas en el producto cartesiano es:

$$|dom(A_1)| * |dom(A_2)| * ... * |dom(A_n)|$$

Una instancia de relación refleja sólo las tuplas válidas que representa un estado particular del mundo real. A medida que el mundo real cambia, también lo hace la relación, transformándose en otro estado de relación (el schema R es relativamente estático y no cambia excepto muy pocas veces).

Notción

Un schema de relación R de grado n se denota $R(A_1, A_2, ..., A_n)$

Una n-tupla t en una relación r(R) se denota $t = <v_1, ..., v_2>$, donde v_i es el valor correspondiente del atributo A_i

$t[A_i]$ se refiere al valor v_i en t para el atributo A_i

$t[A_u, A_w, ..., A_z]$ donde $A_u, A_w, ..., A_z$ es una lista de atributos de R, se refiere a las subtuplas de valores $<v_u, v_w, ..., v_z>$ de t correspondientes a los atributos especificados en la lista

Las letras Q, R, S denotan nombres de relación

Las letras q, r, s denotan estados de relación

Las letras t, u, v denotan tuplas

Los nombres de atributos se califican con el nombre de relación a la cual pertenecen. Por ejemplo, ESTUDIANTE.Nombre o ESTUDIANTE.Edad

Para la tupla t = <Benjamín González, 13.245.622-1, 224-4211, Rosas 3241, 19, Plan común, 4.8>, tenemos t[Nombre] = <Benjamín González>, t[Rut, Prom-notas, Edad] = <13.245.622-1, 4.8, 19>

Restricciones

Las restricciones de dominios especifican que el valor de cada atributo A debe ser un valor atómico del dominio dom(A).

Una relación se define como un conjunto de tuplas. Por definición todos los elementos de un conjunto son distintos. Luego todas las tuplas de una relación deben ser distintas. Esto implica que dos tuplas no pueden tener la misma combinación de valores para todos sus atributos. Pero puede haber otros subconjuntos de atributos de un schema de relación R con la propiedad de que no haya dos tuplas en una instancia de relación r de R que tengan la misma combinación de valores para esos atributos. Supongamos que denotamos tal subconjunto de atributos por SC. Entonces para cada dos tuplas distintas t1 y t2 en una instancia de relación r de R, tenemos la restricción:

$t1[SC] \neq t2[SC]$

Cualquier conjunto de atributos SC es denominado *super llave* del schema de relación R. Cada relación tiene al menos una super llave (el conjunto de todos sus atributos). Una *llave* o *clave* K de un schema de relación R es una super llave de R con la propiedad adicional de que al sacar cualquier atributo A de K deja un conjunto de atributos K' que no es super llave de R (una clave es una super llave minimal).

El valor de un atributo clave se usa para identificar unívocamente una tupla en una relación. El hecho que un conjunto de atributos constituya una clase es una propiedad del schema de la relación, y es invariante en el tiempo.

En general, un schema de relación puede tener más de una clave, y en ese caso, cada una de las llaves es una *llave candidata.* Una de las llaves candidatas se designa como llave primaria de la relación. Usamos la convención de que los atributos que forman la llave primaria de un schema de relación se subrayan.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.3.1. Base de datos relacional

Un *schema de Base de Datos (BD) relacional* es un conjunto de schemas de relación $S = (R_1, R_2, ..., R_m)$ y un conjunto RI de restricciones de integridad.

Una *instancia de BD relacional* db de S es un conjunto de instancias de relación $db = \{r_1, ..., r_n\}$ tal que cada r_i es una instancia de R_i y tal que las relaciones r_i satisfacen las restricciones de integridad especificadas en RI.

Ejemplo:

EMPLEADO

NPILA	APPAT	APMAT	RUT	FNAC	DIRECCION	SEXO	SUELDO	RUTSUPERV	NDEPTO

DEPARTAMENTO

DNOMBRE	DNUMERO	RUTGERENTE	GERFECHAINIC

UBICACIONES_DEPTO

DNUMERO	DUBICACION

PROYECTO

PNOMBRE	PNUMERO	PUBICACION	DNUM

TRABAJA_EN

ERUT	PNO	HORAS

CARGA

ERUT	NOMBRE_CARGA	SEXO	FNAC	PARENTESCO

Los siguientes datos corresponden a una instancia de la base de datos.

EMPLEADO	NPILA	APPAT	APMAT	RUT	FNAC	DIRECCION	SEXO	SUELDO	RU
	Juan	Pérez	García	12345678	9-1-55	Toesca 965	M	120	33
	Alicia	Zelaya	Roa	99988777	19-7-58	Blanco 2120	F	105	98
	Juana	Besa	Martínez	98765432	20-6-31	Mapocho 2540	F	240	88
	Francisco	Cea	Daza	33344555	8-12-45	Condell 221	M	310	88
	Jaime	Ramos	Salas	88866555	10-11-30	Vitacura 1015	M	360	nu

DEPARTAMENTO	DNOMBRE	DNUMERO	RUTGERENTE	GERFECHAINIC
	Of. Central	1	88866555	19-6-71
	Administración	4	98765432	1-1-85
	Investigación	5	33344555	22-5-78

UBICACIONES_DEPTO	DNUMERO	DUBICACION
	1	Providencia
	4	Ñuñoa
	5	La Florida

	5	Pirque

PROYECTO	PNOMBRE	PNUMERO	PUBICACION	DNUM
	Producto X	1	La Florida	5
	Producto Y	2	Pirque	5
	Computarización	10	Ñuñoa	4
	Reorganización	20	Providencia	1

TRABAJA_EN	ERUT	PNO	HORAS
	12345678	1	32.5
	12345678	2	7.5
	33344555	2	10.0
	99988777	10	10.0
	98765432	10	10.0
	98765432	20	15.0
	88866555	20	nulo

CARGA	ERUT	NOMBRE_CARGA	SEXO	FNAC	PARENTESCO
	33344555	Alicia	F	5-4-86	Hija
	33344555	Teodoro	M	25-10-83	Hijo
	33344555	Ximena	F	3-5-54	Cónyuge
	98765432	Rodolfo	M	28-2-32	Cónyuge
	12345678	Alicia	F	5-5-57	cónyuge

Observemos que DNUMERO es el mismo para DEPARTAMENTO y PROYECTO. Pero el mismo concepto se llama DNO en EMPLEADO y DNUM en PROYECTO. No hay problema.

La *restricción de integridad de entidad* establece que ningún valor de llave primaria puede ser nulo. Esto es porque ellas identifican tuplas de la relación.

La *restricción de integridad referencial* se especifica entre dos relaciones y se usa para mantener la consistencia entre tuplas de las dos relaciones. Informalmente, una tupla en una relación que hace referencia a otra relación debe referirse a una tupla existente en esa relación. Por ejemplo, NDEPTO de EMPLEADO debe coincidir con el DNUMERO de alguna tupla de la relación DEPARTAMENTO. Para una definición formal, necesitamos el concepto de llave foránea.

Def. Un conjunto de atributos LF en el schema de relación R_1 es una *llave foránea* de R_1 si satisface las siguientes reglas:

1. Los atributos en LF tienen el mismo dominio que los atributos de la llave primaria LP de otro schema de relación R_2. Los atributos LF se dice que *referencian* la relación R_2.

2. Un valor de LF en una tupla t_1 de R_1 ya sea es nulo, o ocurre como un valor de LP para

alguna tupla t_2 de R_2.

Ejemplo: NDEPTO en una tupla t_1 de EMPLEADO debe coincidir con el valor de una llave primaria DNUMERO en alguna tupla t_2 de la relación DEPARTAMENTO, o el valor de DNO puede ser nulo si el empleado no pertenece a un departamento.

Las restricciones anteriores no consideran las *restricciones semánticas* que quizás puedan especificarse y sostenerse en una BD relacional. Por ejemplo, "el sueldo de un empleado no puede exceder el de su supervisor", "el número máximo de horas que puede trabajar un empleado en todos los proyectos es 56".

Operaciones de actualización

La operación *Insert* provee una lista de valores de atributos para una nueva tupla t que se va a insertar en una relación R. Ejemplo:

Insert <"Cecilia", "Rodríguez", "Kolonsky", "67767898", "5-4-50", "Ejército 565", "F", 100, nulo, "4"> en EMPLEADO. Esta inserción satisface todas las restricciones, así que es aceptable.

Insert <"Alicia", "Zelaya", "Roa", "99988777", "15-3-50", "Gay 1315", "F", 120, "98765432", "4"> en EMPLEADO. Viola la restricción de clave porque otra tupla con el mismo Rut ya existe en la relación. Inaceptable.

Insert <"Cecilia", "Rodríguez", "Kolonsky", nulo, "5-4-50", "Ejército 565", "F", 100, nulo, 4> en EMPLEADO. Viola la restricción de integridad (nulo para clave primaria Rut). Inaceptable.

Insert <"Cecilia", "Rodríguez", "Kolonsky", "67767898", "5-4-50", "Ejército 565", "F", 100, "98765432", 7> en EMPLEADO. Viola la restricción de integridad referencial porque no existe una tupla en DEPARTAMENTO con DNUMERO = 7.

El SABD puede hacer dos opciones cuando hay violación de restricciones. Una es rechazar la inserción. La otra es intentar corregir la razón de rechazo de la inserción.

Operaciones de borrado

La operación *Delete* borra tuplas de una relación. Es posible que se viole la *integridad referencial* cuando una tupla que se quiere borrar es referenciada por las llaves foráneas de otras tuplas de la BD. Las tuplas a borrar se especifican a través de condiciones sobre los atributos de la relación. Ejemplos:

Delete tupla con ERUT = "99988777" AND PNO = 10 en la relación TRABAJA_EN. Esta operación es aceptable.

Delete tupla con RUT = "99988777" en la relación EMPLEADO. Inaceptable porque dos tuplas en TRABAJA_EN se refieren a esta tupla. Si se borra, hay violaciones a la integridad referencial.

Hay tres opciones con respecto a una operación de borrado que causa una violación. La primera es *rechazar* la operación. La segunda es intentar *propagar* el borrado. Una tercera opción es *modificar* los valores de los atributos referenciantes que causan la violación (cada uno de estos valores puede ser puesto en nulo o cambiado para referenciar otra tupla válida). Hay que observar que si un atributo referenciante que causa una violación, es parte de la llave primaria, no puede ser nulo, pues se violaría la integridad de entidad.

Operaciones de modificación

La operación *Modify* se usa para cambiar valores a uno o más atributos en una tupla (o tuplas) de una relación R. Es necesario especificar una condición sobre los atributos de R para seleccionar la o las tuplas a modificar. Ejemplos:

Modify SUELDO de la tupla de EMPLEADO con RUT = "99988777" a 135. Aceptable.

Modify NDEPTO de la tupla de EMPLEADO con RUT = "99988777" a 1. Aceptable.

Modify NDEPTO de la tupla de EMPLEADO con RUT = "99988777" a 7. Inaceptable pues viola la integridad referencial.

Modify RUT de la tupla de EMPLEADO con RUT = "99988777" a "98765432". Inaceptable pues viola restricciones de clave primaria e integridad referencial.

El modificar un atributo que no es llave primaria ni llave foránea no tiene problemas. El modificar una llave primaria es similar a borrar una tupla e insertar otra en su lugar. Por tanto, es relevante la discusión anterior (Insert y Delete). Si se modifica un atributo de una llave foránea, el Sistema Administrador de Base de Datos (SABD) debe asegurarse que el nuevo valor se refiere a una tupla existente en la relación referenciada.

Ejercicios:

Discuta *todas* las restricciones de integridad que viola cada una de las siguientes operaciones (si es que las hay), y las maneras más apropiadas de resolver estas violaciones.

Insert <"Producto A", 4, "La Florida", 2> en PROYECTO.

Insert <"Producción", 4, "94377554", "1-oct-98"> en DEPARTAMENTO.

Delete tupla con PNOMBRE = "Producto X" en la relación PROYECTO.

Modify RUTGERENTE y GERFECHAINIC de la tupla DEPARTAMENTO con DNUMERO = 5 a "12345678" y "01-10-93", respectivamente.

Modify HORAS de la tupla de TRABAJA_EN con RUT = "99988777" y PNO = 10 a "50".

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.4. SQL - Lenguaje de consulta en BD relacionales

SQL ("Structured Query Language") es un lenguaje para realizar consultas en BD relacionales. Fue desarrollado por IBM, y después de algunas modificaciones fue estandarizado en 1986.

SQL usa los términos *tabla, fila* y *columna* para *relación, tupla* y *atributo*, respectivamente.

Un schema SQL es identificado por un *nombre de schema*, e incluye un identificador de autorización que indica el usuario dueño del schema, así como los *descriptores* para cada elemento en el schema. Los elementos del schema incluyen las tablas, vistas, dominios y otros constructores que describen el schema. La instrucción CREATE SCHEMA es usada para crear un nuevo schema. Por ejemplo, la

siguiente instrucción crea un schema llamado COMPAÑIA, cuyo dueño es JPérez:

CREATE SCHEMA COMPAÑIA **AUTHORIZATION** JPérez;

La instrucción CREATE TABLE es usada para especificar una nueva relación, dándole un nombre y especificando sus atributos y restricciones. A cada atributo se le da un nombre, un tipo de datos (para especificar su dominio de valores) y opcionalmente algunas restricciones. De este modo se especifican las restricciones de integridad RI definidas en la sección anterior.

El siguiente ejemplo muestra las instrucciones de creación de datos en SQL para el ejemplo mostrado en la sección anterior.

```
CREATE TABLE EMPLEADO
    ( NPILA       VARCHAR(15)   NOT NULL,
    APPAT         VARCHAR(15)   NOT NULL,
    APMAT         VARCHAR(15)   NOT NULL,
    RUT           VARCHAR(10)   NOT NULL,
    FNAC          DATE,
    DIRECCION     VARCHAR(30),
    SEXO          CHAR,
    SUELDO        DECIMAL(5,2),
    RUTSUPERV     VARCHAR(10),
    NDEPTO        INT           NOT NULL,
  PRIMARY KEY (RUT),
  FOREIGN KEY (RUTSUPERV) REFERENCES EMPLEADO(RUT),
  FOREIGN KEY (NDEPTO) REFERENCES DEPARTAMENTO(DNUMERO));

CREATE TABLE DEPARTAMENTO
    ( DNOMBRE     VARCHAR(15)   NOT NULL,
    DNUMERO       INT           NOT NULL,
    RUTGERENTE    VARCHAR(10)   NOT NULL,
    GERFECHAINIC  DATE,
  PRIMARY KEY (DNUMERO),
  UNIQUE (DNOMBRE),
  FOREIGN KEY (RUTGERENTE) REFERENCES EMPLEADO(RUT));

CREATE TABLE UBICACIONES_DEPTO
    ( DNUMERO     INT           NOT NULL,
    DUBICACION    VARCHAR(15)   NOT NULL,
  PRIMARY KEY (DNUMERO, DUBICACION),
  FOREIGN KEY (DNUMERO) REFERENCES DEPARTAMENTO(DNUMERO));

CREATE TABLE PROYECTO
    ( PNOMBRE     VARCHAR(15)   NOT NULL,
    PNUMERO       INT           NOT NULL,
    PUBICACION    VARCHAR(15),
    DNUM          INT           NOT NULL,
  PRIMARY KEY (PNUMERO),
  UNIQUE (PNOMBRE),
  FOREIGN KEY (DNUM) REFERENCES DEPARTAMENTO(DNUMERO));
```

```
CREATE TABLE TRABAJA_EN
    ( ERUT    VARCHAR(10)    NOT NULL,
    PNO       INT            NOT NULL,
    HORAS     DECIMAL(3,1)   NOT NULL,
   PRIMARY KEY (ERUT, PNO),
   FOREIGN KEY (ERUT) REFERENCES EMPLEADO(RUT),
   FOREIGN KEY (PNO) REFERENCES PROYECTO(PNUMERO));

CREATE TABLE CARGA
    ( ERUT            VARCHAR(10)    NOT NULL,
    NOMBRE_CARGA      VARCHAR(15)    NOT NULL,
    SEXO              CHAR,
    FNAC              DATE,
    PARENTESCO        VARCHAR(8),
   PRIMARY KEY (ERUT, NOMBRE_CARGA),
   FOREIGN KEY (ERUT) REFERENCES EMPLEADO(RUT));
```

También se puede agregar explícitamente el nombre del schema a cada tabla, separado por un punto. Por ejemplo:

```
CREATE TABLE COMPAÑIA.EMPLEADO ...
```

Esto hace que la tabla EMPLEADO sea parte del schema COMPAÑIA.

Los tipos de datos disponibles para los atributos incluyen: numérico, tira de caracteres, caracter, fecha y hora. Los tipos *numéricos* pueden incluir números enteros de varios tamaños (INT y SMALLINT), números reales de varias precisiones (FLOAT, REAL, DOUBLE PRECISION). Además se pueden declarar números con formato, usando DECIMAL(i,j). Las *tiras de caracteres* pueden ser de largo fijo (CHAR(n)) o de largo variable (VARCHAR(n), donde n es el máximo número de caracteres). La *fecha* tiene 10 posiciones, típicamente AAAA-MM-DD. La *hora* tiene al menos 8 posiciones, típicamente HH:MM:SS. Solamente fechas y horas válidas son permitidas en las implementaciones de SQL.

En SQL es posible especificar directamente el tipo de dato para cada atributo, como se mostró en el ejemplo anterior. Pero también sepueden declarar dominios, y usar el nombre de éstos. Esto facilita hacer cambios enlos tipos de datos (cambiando sólo el dominio y no cada dato declarado). Por ejemplo, podemos crear el dominio TIPO_RUT con la siguiente instrucción:

```
CREATE DOMAIN TIPO_RUT AS VARCHAR(10);
```

A partir de ahora, podemos usar TIPO_RUT en lugar de VARCHAR(10), por ejemplo en los atributos RUT, RUTSUPERV, RUTGERENTE y ERUT del ejemplo anterior.

Debido a que SQL permite el "NULL" (nulo) como valor de sus atributos, es necesario especificar la restricción "NOT NULL" para los atributos que no permiten este valor (por violaciones de integridad). Esta restricción siempre debe ser especificada para los atributos que son llaves primarias en cada relación.

Es posible definir un *valor por defecto* para un atributo agregando la cláusula **DEFAULT** "valor" en la definición del atributo.

La cláusula **PRIMARY KEY** especifica uno o más atributos que forman la llave primaria de la relación. La cláusula **UNIQUE** especifica llaves alternas. La integridad de referencia es especificada a través de la cláusula **FOREIGN KEY**.

Como se discutió en secciones anteriores, las restricciones de integridad referencial pueden ser violadas cuando las tuplas son insertadas o borradas, o cuando se cambia el valor de un atributo que es llave foránea. Al crear el schema es posible especificar las acciones a ser tomadas cuando una restricción de integridad referencial es violada, ya sea por borrado de una tupla referenciada en otra tabla, o por modificación del valor de una llave primaria referenciada en otra tabla. Estas acciones son: ON DELETE (cuando la tupla se borra) y ON UPDATE (cuando la tupla se modifica), que pueden tener las opciones: SET NULL (ponga en nulo), CASCADE (actualice todas las referencias "en cascada"), y SET DEFAULT (ponga el valor por defecto). Por ejemplo:

```
CREATE TABLE EMPLEADO
    ( ...,
    NDEPTO    INT    NOT NULL    DEFAULT 1,
  CONSTRAINT EMPLP
    PRIMARY KEY RUT,
  CONSTRAINT RUTSUPLF
    FOREIGN KEY (RUTSUPERV) REFERENCES EMPLEADO(RUT)
       ON DELETE SET NULL ON UPDATE CASCADE,
  CONSTRAINT NDEPTOLF
    FOREIGN KEY (NDEPTO) REFERENCES DEPARTAMENTO(DNUMERO)
       ON DELETE SET DEFAULT ON UPDATE CASCADE );
```

En el ejemplo anterior, si la tupla de un empleado supervisor es borrada, el valor de RUTSUPERV es puesto en nulo (NULL) para todas las tuplas de empleados que referencian al empleado de la tupla borrada. Además, si el valor de RUT es modificado para un empleado supervisor (por ejemplo porque fue ingresado incorrectamente), el nuevo valor es actualizado "en cascada" en RUTSUPERV para todos los empleados que referencian la tupla modificada de este supervisor.

En el caso de NDEPTO, éste es puesto en 1 (el valor declarado por defecto) si la tupla correspondiente a ese número de departamento es borrada de la tabla de DEPARTAMENTO, y es actualizado en cascada (en toda la tabla) cuando la tupla correspondiente en DEPARTAMENTO es actualizada.

A las restricciones se les puede dar nombre usando la palabra CONSTRAINT.

Para borrar un schema completo se usa la instrucción DROP SCHEMA, con dos opciones: CASCADE o RESTRICT. Por ejemplo, para borrar el schema de base de datos COMPAÑIA y todas sus tablas, dominios y otros elementos, se usa la opción CASCADE:

```
DROP SCHEMA COMPAÑIA CASCADE;
```

Si en la instrucción anterior se reemplaza la opción CASCADE por RESTRICT, el schema es borrado solamente si no tiene elementos. En caso de que el schema tenga algún elemento, el borrado no es ejecutado.

Una relación o tabla puede ser borrada del schema de BD usando la instrucción DROP TABLE. Por ejemplo, si la relación CARGA con información de los dependientes de los empleados no va a ser utilizada más en la BD COMPAÑIA, se puede borrar de la siguiente manera:

```
DROP TABLE CARGA CASCADE;
```

Si la opción RESTRICT es usada en lugar de CASCADE, la tabla es borrada solamente si ésta no es referenciada en ninguna restricción (por ejemplo como llave foránea en otra tabla). Con la opción CASCADE todas las restricciones que referencian esta tabla, son borradas automáticamente del schema, junto con la tabla.

La definición de una tabla puede ser modificada usando la instrucción ALTER TABLE. Con esta instrucción es posible agregar o borrar atributos (columnas), cambiar la definición de una columna, y agregar o borrar restricciones. Por ejemplo, para agregar un atributo con el puesto de los empleados de la tabla EMPLEADO, se usa:

ALTER TABLE COMPAÑIA.EMPLEADO **ADD** PUESTO VARCHAR(12);

Para borrar una columna se puede usar CASCADE o RESTRICT. Con CASCADE todas las restricciones son borradas automáticamente del schema, junto con la columna. Por ejemplo:

ALTER TABLE COMPAÑIA.EMPLEADO **DROP** DIRECCION **CASCADE**,

Si en la instrucción anterior se usa la opción RESTRICT en lugar de CASCADE, el atributo DIRECCION es borrado solamente si ninguna restricción lo referencia.

También es posible borrar una cláusula por defecto así como definir una nueva. Por ejemplo:

ALTER TABLE COMPAÑIA.EMPLEADO **ALTER** NDEPTO **DROP DEFAULT**;
ALTER TABLE COMPAÑIA.EMPLEADO **ALTER** NDEPTO **SET DEFAULT** "5";

Finalmente, se pueden borrar o agregar restricciones en una tabla. Para borrar una restricción ésta debe tener un nombre (dado con CONSTRAINT). Por ejemplo, para borrar la restricción NDEPTOLF de la tabla EMPLEADO:

ALTER TABLE COMPAÑIA.EMPLEADO **DROP CONSTRAINT** NDEPTOLF **CASCADE**;

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.4.1. Consultas en SQL

SQL tiene una instrucción principal para recuperar información de una base de datos: el comando SELECT. Esta instrucción tiene muchas opciones. La forma básica de la instrucción SELECT es la siguiente:

SELECT <lista de atributos>
FROM <lista de tablas>
WHERE <condición>

donde:

<lista de atributos> es una lista de nombres de atributos cuyos valores van a ser recuperados por la consulta.

<lista de tablas> es una lista de nombres de relaciones requeridos para procesar la consulta.

<condición> es una expresión de búsqueda condicional (lógica) que identifica las tuplas que van a ser recuperadas por la consulta.

En los siguientes ejemplos usaremos las tablas (junto con la información de éstas) definidas en las secciones anteriores.

Consulta 0

Recuperar la fecha de nacimiento y la dirección del empleado cuyo nombre es "Juan Pérez".

```
Q0:  SELECT FNAC, DIRECCION
     FROM   EMPLEADO
     WHERE  NPILA = "Juan" AND APPAT = "Pérez"
```

Esta consulta involucra solamente la relación EMPLEADO, señalada en la cláusula FROM. La consulta salecciona la tupla de EMPLEADO que satisface la condición de la cláusula WHERE, y selecciona los valores de esta tupla correspondientes a los atributos FNAC y DIRECCION.

Consulta 1

Recuperar el nombre y la dirección de todos los empleados que trabajan en el departamento "Investigación".

```
Q1:  SELECT NPILA, APPAT, DIRECCION
     FROM   EMPLEADO, DEPARTAMENTO
     WHERE  DNOMBRE = "Investigación" AND DNUMERO = NDEPTO
```

En la cláusula WHERE, la condición DNOMBRE = "Investigación" es una *condición de selección*. La condición DNUMERO = NDEPTO es una *condición de asociación*, y asocia la llave foránea NDEPTO de la relación EMPLEADO, con el correspondiente número de departamento (DNUMERO) de la relación DEPARTAMENTO. La siguiente consulta tiene dos condiciones de asociación

Consulta 2

Para todos los proyectos localizados en "La Florida", liste el número de proyecto, el número de departamento que lo controla, y el nombre, dirección y fecha de nacimiento del gerente de ese departamento.

```
Q2:  SELECT PNUMERO, DNUM, NPILA, APPAT, DIRECCION, FNAC
     FROM   PROYECTO, DEPARTAMENTO, EMPLEADO
     WHERE  DNUM = DNUMERO AND RUTGERENTE = RUT AND PUBICACION = "La
            Florida"
```

La condición DNUM = DNUMERO relaciona un proyecto con su correspondiente departamento, mientras que la condición RUTGERENTE = RUT relaciona el departamento que controla el proyecto, con el empleado que administra ese departamento.

En SQL, un mismo nombre puede ser usado por dos (o más) atributos en diferentes relaciones. Cuando esto sucede, y una consulta se refiere a dos o más atributos con el mismo nombre, el nombre

de la relación debe ser puesto como prefijo del nombre de cada atributo, para avitar ambigüedad. Por ejemplo, supongamos que los atributos NDEPTO y NPILA de EMPLEADO se llamaran DNUMERO y NOMBRE respectivamente, y el atributo DNOMBRE de DEPARTAMENTO también se llamara NOMBRE. Entonces, para evitar ambigüedad en Q1, deberíamos usar como prefijos de los atributos, los nombres de las relaciones:

```
Q1A:  SELECT EMPLEADO.NOMBRE, APPAT, DIRECCION
      FROM   EMPLEADO, DEPARTAMENTO
      WHERE  DEPARTAMENTO.NOMBRE = "Investigación" AND
             DEPARTAMENTO.DNUMERO = EMPLEADO.DNUMERO
```

La ambigüedad también aparece en el caso de que la consulta se refiera a la misma relación dos o más veces, como en el siguiente ejemlo.

Consulta 3

Para cada empleado, recuperar el nombre y primer apellido, y el nombre y primer apellido de su supervisor inmediato.

```
Q3:   SELECT E.NPILA, E.APPAT, S.NPILA, S.APPAT
      FROM   EMPLEADO E, EMPLEADO S
      WHERE  E.RUTSUPERV = S.RUT
```

En este caso, se pueden declarar nombres alternativos para la misma relación, llamados *alias*. El nombre del alias se escribe inmediatamente después del nombre de la relación. También se puede declarar usando la palabra "AS", por ejemplo: EMPLEADO AS E. También es posible renombrar (como alias) todos los atributos de una relación en la cláusula FROM, de esta manera: EMPLEADO AS E(NP, AP1, AP2, RUT, FN, DIR, SE, SU, RUTS, ND). De este modo, se podría comparar (en la consulta anterior) E.RUTS = S.RUT.

Los alias pueden ser usados en cualquier consulta, no sólo cuando hay nombres repetidos. Los alias tienen sentido sólo en la consulta en que son definidos. Un alias no cambia "físicamente" el nombre de ninguna relación ni atributo de la BD.

En una consulta puede omitirse la cláusula WHERE, lo que indica que no hay condiciones sobre las tuplas a seleccionar (TODAS las tuplas son seleccionadas). Por ejemplo:

Consulta 4

Recuperar todos los números de RUT de los empleados.

```
Q4:   SELECT RUT
      FROM   EMPLEADO
```

Si más de una relación es especificada en la cláusula FROM, y no existe cláusula WHERE, entonces el *producto cruz* (todas las posibles combinaciones de tuplas) de estas relaciones es seleccionado. Por ejemplo:

Consulta 5

Recuperar todas las combinaciones de números de RUT de los empleados y nombre de departamentos.

```
Q5:  SELECT RUT, DNOMBRE
     FROM   EMPLEADO, DEPARTAMENTO
```

Es importante especificar cada condición de selección y cada condición de asociación en la cláusula WHERE. Si alguna de estas condiciones es omitida, relaciones incorrectas o muy grandes pueden dar como resultado.

Para recuperar todos los valores de los atributos de las tuplas seleccionadas, se puede usar un asterisco (no es necesario poner todos los nombres), el cual significa *todos los atributos*. Por ejemplo:

Consulta 6

Recuperar los valores de todos los atributos de EMPLEADO que trabajan en el departamento número "5".

```
Q6:  SELECT *
     FROM   EMPLEADO
     WHERE  NDEPTO = 5
```

Consulta 7

Recuperar los valores de todos los atributos de EMPLEADO y los atributos del DEPARTAMENTO en que el empleado trabaja, para cada empleado del departamento de "Investigación".

```
Q7:  SELECT *
     FROM   EMPLEADO, DEPARTAMENTO
     WHERE  DNOMBRE = "Investigación" AND NDEPTO = DNUMERO
```

Consulta 8

Recuperar el producto cruz de las relaciones EMPLEADO y DEPARTAMENTO.

```
Q8:  SELECT *
     FROM   EMPLEADO, DEPARTAMENTO
```

En una consulta SQL pueden aparecer tuplas duplicadas. Si no queremos que esto suceda, se puede usar la palabra DISTINCT en la cláusula SELECT, en cuyo caso sólo tuplas distintas aparecen en la relación. Por ejemplo:

Consulta 9

Recuperar el salario de cada empleado.

```
Q9:  SELECT SUELDO
     FROM   EMPLEADO
```

En la consulta anterior obtenemos una tabla con la lista de salarios de todos los empleados. Sin embargo, si dos o más empleados ganan lo mismo, el mismo valor aparece varias veces en la tabla.

Si queremos que no se repitan los salarios, usamos DISTINCT:

```
Q9A:  SELECT DISTINCT SUELDO
      FROM    EMPLEADO
```

En SQL existe una operación UNION que regresa la unión (como en conjuntos) de relaciones, es decir, regresa todas las tuplas que aparecen en alguna de las relaciones. Las tuplas duplicadas son eliminadas del resultado, a menos que se especifique la cláusula ALL después de la operación. Por ejemplo:

Consulta 10

Regresar una lista con todos los números de proyecto que involucran un empleado de apellido "Pérez", ya sea como trabajador o como gerente del departamento que controla ese proyecto.

```
Q10:  ( SELECT PNUMERO
      FROM    PROYECTO, DEPARTAMENTO, EMPLEADO
      WHERE   DNUM = DNUMERO AND RUTGERENTE = RUT AND APPAT = "Pérez" )
      UNION
      ( SELECT PNO
      FROM    TRABAJA_EN, EMPLEADO
      WHERE   ERUT = RUT AND APPAT = "Pérez" )
```

El primer SELECT recupera los proyectos que involucran a "Pérez" como gerente del departamento que controla el proyecto, y el segundo SELECT regresa los proyectos que involucran a "Pérez" como trabajador en el proyecto. Note que si varios empleados tienen como primer apellido "Pérez", se recuperan los números de proyecto en que están involucrados todos ellos. Al aplicar la operación UNION a los dos SELECT, se obtiene el resultado deseado.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

2.4.1. Consultas en SQL (continuación)

Algunas consultas requieren que ciertos valores de la base de datos sean antes recuperados y luego usados en las condiciones de comparación. Estas consultas pueden ser implementadas a través de *consultas anidadas*, donde hay consultas completas dentro de las cláusulas WHERE de otras consultas. Por ejemplo, la consulta anterior (Q10) podría ser implementada así:

```
Q10A:  SELECT DISTINCT
       PNUMERO
       FROM    PROYECTO
       WHERE   PNUMERO IN    ( SELECT PNUMERO
                             FROM PROYECTO, DEPARTAMENTO, EMPLEADO
                             WHERE DNUM = DNUMERO AND RUTGERENTE =
                             RUT AND APPAT = "Pérez" )
               OR
               PNUMERO IN    ( SELECT PNO
                             FROM TRABAJA_EN, EMPLEADO
```

WHERE ERUT = RUT **AND** APPAT = "Pérez")

La primera consulta anidada selecciona los números de proyecto de los proyectos que tienen a "Pérez" como gerente, mientras que la segunda selecciona los números de proyecto que tienen a "Pérez" como trabajador. El operador de comparación IN compara un valor v con un conjunto de valores V, y regresa TRUE si v es uno de los elementos en V.

Además del operador IN, pueden ser usados otros operadores para comparar un valor v (típicamente un nombre de atributo) con un conjunto V (típicamente una consulta anidada). El operador "= ANY" (o "= SOME") regresa TRUE si el valor v es igual a algún valor en el conjunto V. ANY y SOME tienen el mismo significado, y en este caso son equivalentes a IN. Otros operadores que pueden ser combinados con ANY (o con SOME) incluyen >, >=, <, <= y <>. Por ejemplo, la condición "v > ALL V" regresa TRUE si el valor v es mayor que *todos* los valores del conjunto V.

Consulta 11

Recuperar los nombres de los empleados cuyo salario es mayor que el salario de todos los empleados del departamento 5.

```
Q11:  SELECT NPILA, APPAT
      FROM   EMPLEADO
      WHERE  SUELDO > ALL ( SELECT SUELDO
                            FROM EMPLEADO
                            WHERE NDEPTO = 5 )
```

En las consultas anidadas pueden usarse atributos de las consultas exteriores. Por ejemplo:

Consulta 12

Recuperar el nombre de cada empleado que tenga un dependiente con el mismo nombre de pila y sexo que el empleado.

```
Q12:  SELECT E.NPILA,
             E.APPAT
      FROM   EMPLEADO
      WHERE  E.RUT IN (   SELECT ERUT
                          FROM CARGA
                          WHERE ERUT = E.RUT AND E.NPILA =
                          NOMBRE_CARGA AND CARGA.SEXO = E.SEXO )
```

Las consultas anidadas *siempre* pueden ser escritas como consultas de un sólo bloque. Por ejemplo, la consulta anterior puede ser escrita así:

```
Q12A: SELECT E.NPILA, E.APPAT
      FROM   EMPLEADO E, CARGA C
      WHERE  E.RUT = C.ERUT AND E.SEXO = C.SEXO AND E.NPILA =
             C.NOMBRE_CARGA
```

La cláusula **EXISTS** es usada para chequear cuándo el resultado de una consulta anidada está vacío (no contiene tuplas). Por ejemplo:

Consulta 13

Recuperar los nombre de los empleados que no tienen dependientes.

```
Q13:  SELECT NPILA, APPAT
      FROM   EMPLEADO
      WHERE  NO EXISTS ( SELECT *
                         FROM CARGA
                         WHERE ERUT = RUT )
```

Es posible renombrar los atributos que aparecen en el resultado de una consulta, agregando el calificador AS seguido del nuevo nombre. Por ejemplo, la consulta 3 podría ser modificada para distinguir el nombre del empleado del nombre de su supervisor, de la siguiente manera:

```
Q3A:  SELECT E.NPILA AS NOMBRE_EMP, E.APPAT AS APELLIDO_EMP, S.NPILA AS
            NOMBRE_SUP, S.APPAT AS APELLIDO_SUP
      FROM  EMPLEADO AS E, EMPLEADO AS S
      WHERE E.RUT = S.RUTSUPERV
```

SQL permite usar algunas funciones como COUNT (cuenta el número de tuplas en una consulta), SUM (regresa la suma de algún atributo numérico), MIN (regresa el mínimo), MAX (regresa el máximo) y AVG (regresa el promedio). Por ejemplo:

Consulta 14

Regresar la suma de los salarios de todos los empleados, el salario máximo, el salario mínimo y el promedio de los salarios.

```
Q14:  SELECT SUM(SUELDO), MAX(SUELDO), MIN(SUELDO), AVG(SUELDO)
      FROM   EMPLEADO
```

Consulta 15

Regresar el número de empleados del departamento de "Investigación".

```
Q15:  SELECT COUNT(*)
      FROM   EMPLEADO, DEPARTAMENTO
      WHERE  DNUMERO = NDEPTO AND DNOMBRE = "Investigación"
```

Consulta 16

Cuente el número de salarios distintos en la base de datos.

```
Q16:  SELECT COUNT(DISTINCT SUELDO)
      FROM   EMPLEADO
```

Consulta 17

Regrese los nombres de todos los empleados que tienen más de dos dependientes.

```
Q17:  SELECT NPILA, APPAT
      FROM   EMPLEADO
      WHERE  ( SELECT COUNT (*)
               FROM CARGA
               WHERE ERUT = RUT ) >= 2
```

En muchos casos se quiere aplicar una función a *grupos de tuplas* en una relación. Por ejemplo, si queremos conocer el promedio de salarios de empleados por cada departamento. En estos casos necesitamos agrupar por cierto(s) atributo(s). Para hacer esto, SQL provee la cláusula GROUP BY.

Consulta 18

Para cada departamento regrese: el número de departamento, el número de empleados en ese departamento y el salario promedio.

```
Q18:  SELECT    NDEPTO, COUNT(*), AVG(SUELDO)
      FROM      EMPLEADO
      GROUP BY NDEPTO
```

Consulta 19

Para cada proyecto regrese su número, nombre, y número de empleados que trabajan en él.

```
Q19:  SELECT    PNUMERO, PNOMBRE, COUNT(*)
      FROM      PROYECTO, TRABAJA_EN
      WHERE     PNUMERO = PNO
      GROUP BY PNUMERO, PNOMBRE
```

En el ejemplo anterior, el agrupamiento y las funciones son aplicados después de la unión de las dos relaciones. Sin embargo, algunas veces se desea recuperar los valores de estas funciones solamente para grupos que satisfacen ciertas condiciones. Por ejemplo, supongamos que queremos modificar la consulta anterior para que solamente aparezcan los proyectos con más de dos empleados. Para hacer esto, SQL provee la cláusula HAVING, la cual puede aparecer junto con la cláusula GROUP BY. HAVING provee una condición sobre el grupo de tuplas asociadas con cada valor de los atributos agrupados, y en el resultado aparecen solamente los grupos que satisfacen esta condición. Por ejemplo:

Consulta 20

Para cada proyecto en que trabajan más de dos empleados, recuperar el número de proyecto, el nombre del proyecto, y el número de empleados que trabajan en el proyecto.

```
Q20:  SELECT    PNUMERO, PNOMBRE, COUNT(*)
      FROM      PROYECTO, TRABAJA_EN
      WHERE     PNUMERO = PNO
      GROUP BY PNUMERO, PNOMBRE
      HAVING    COUNT(*) > 2
```

Consulta 21

Para cada departamento que tenga más de cinco empleados, recuperar el número de departamento y el número de empleados que ganan más de $350.000.

```
Q21:  SELECT DNOMBRE, COUNT(*)
      FROM   DEPARTAMENTO, EMPLEADO
      WHERE  DNUMERO = NDEPTO AND SUELDO > 350.000 AND NDEPTO IN
             ( SELECT NDEPTO
             FROM EMPLEADO
             GROUP BY NDEPTO
             HAVING COUNT(*) > 5 )
```

Finalmente, SQL permite ordenar las tuplas que resultan de las consultas, por los valores de uno o más atributos, usando la cláusula ORDER BY. Por ejemplo:

Consulta 22

Regrese una lista de empleados y los proyectos en los que trabajan, ordenada por departamento, y dentro de cada departamento, ordenada alfabéticamente por nombre y apellido.

```
Q22:  SELECT    DNOMBRE, NPILA, APPAT, PNOMBRE
      FROM      DEPARTAMENTO, EMPLEADO, TRABAJA_EN, PROYECTO
      WHERE     DNUMERO = NDEPTO AND RUT = ERUT AND PNO = PNUMERO
      ORDER BY DNOMBRE, APPAT, NPILA
```

Por defecto, el ordenamiento es en orden ascendente (ASC). También se puede especificar un orden descendiente (DESC). Por ejemplo, si en la consulta anterior se desea tener ordenado descendentemente por nombre de departamento, y ordenado ascendentemente por nombre de empleado, se puede especificar así:

```
ORDER BY DNOMBRE DESC, APPAT ASC, NPILA ASC
```

Otras operaciones

Para insertar una tupla en una relación se usa INSERT. Por ejemplo:

```
INSERT
INTO      EMPLEADO
VALUES    ( "Ricardo", "Cruz", "Pérez", "99887766", "1-9-72", "Blanco 1999", "M", 400,
          "33344555", 5 )
```

Para borrar tuplas de una relación se usa DELETE. Por ejemplo:

```
DELETE FROM EMPLEADO
WHERE       APPAT = "Pérez"


DELETE FROM EMPLEADO
```

```
WHERE       NDEPTO IN ( SELECT DNUMERO
                        FROM DEPARTAMENTO
                        WHERE DNOMBRE = "Investigaciones" )
```

También se pueden modificar los valores de los atributos usando UPDATE. Por ejemplo:

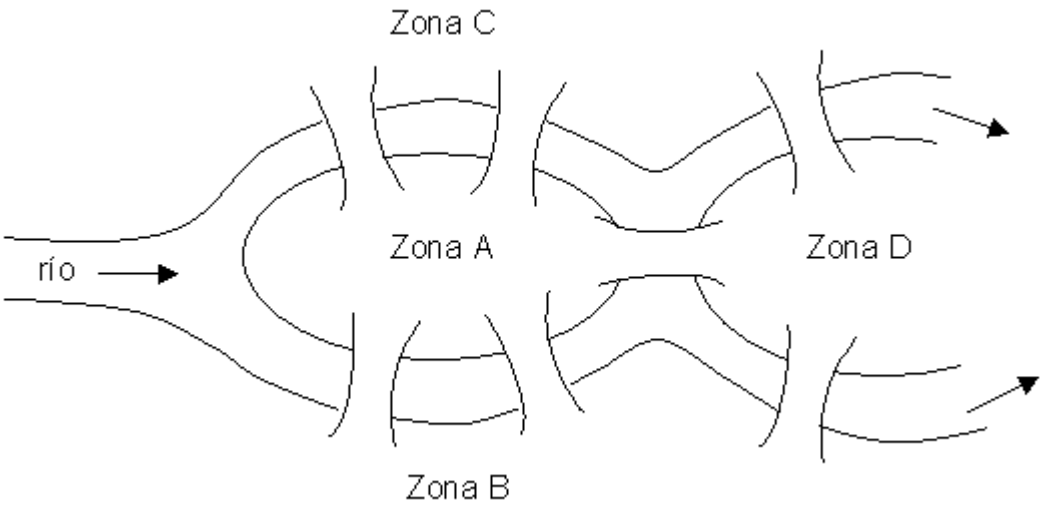
```
UPDATE PROYECTO
SET    PUBICACION = "Macul", DNUM = 3
WHERE  PNUMERO = 10
```

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

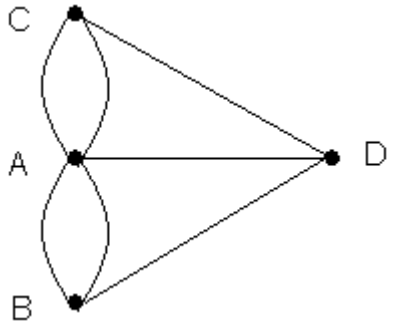
3.1. Modelamiento de redes

Las redes se modelan con *grafos*. Euler (1707-1782) es considerado el originador de la teoría de grafos, cuando en 1736 resolvió un problema famoso en su tiempo, llamado el problema de los puentes de Königsberg. El río Pregel formaba islas en esa ciudad, y había 7 puentes:



El problema era comenzar encualquiera de las cuatro zonas de tierra (A, B, C o D), atravesar sólo una vez por cada puente y volver al punto de partida.

Para resolver este problema, Euler reemplazó cada zona de tierra por un punto, y cada puente por una línea uniendo los puntos correspondientes y produciendo un *multigrafo*.



Euler demostró que el problema no tenía solución. Para ello, generalizó el problema y desarrolló un criterio para que un multigrafo dado fuera *recorrible* de la manera requerida. Este criterio pide que el grafo sea *conexo* y que cada punto tenga un número par de líneas incidentes. El multigrafo de los puentes de Königsberg es conexo, pero no todo punto tiene un número par de líneas incidentes.

Definición. Un *grafo* G consiste de un conjunto finito no vacío V = V(G) de p *puntos* (o *vértices*, o *nodos*), y un conjunto X de pares no ordenados de puntos distintos de V. Cada par x = {u, v} de puntos en X es una *línea* (o *arco*, o *lado*) de G y se dice que x *une* u y v. Escribimos x = uv y decimos que u y v son *vértices adyacentes*. Un grafo con p puntos y q líneas se llama un grafo (p, q). El grafo (1, 0) es trivial.

Ejercicio. Mostrar que un multigrafo (es decir, varias líneas uniendo dos vértices) no es un grafo.

Definición. Un *grafo dirigido* (o *digrafo*) consiste de un conjunto no vacío finito V de puntos y una colección X de pares ordenados de puntos distintos. Los elementos de X son *líneas dirigidas* o *arcos*. Un digrafo no puede tener ciclos o arcos múltiples.

Un grafo G es *rotulado* cuando los p puntos se distinguen uno de otro por nombres tales como v_1, v_2, ..., v_p.

Un subgrafo de G es un grafo que tiene todos sus puntos y líneas en G. Un *subgrafo de recubrimiento* es un subgrafo que contiene todos los puntos de G.

Definición. Un *camino* de un grafo G es una secuencia alternada de puntos y líneas v_0, x_1, v_1, ..., v_{n-1}, x_n, v_n comenzando y terminando en vértices, en que cada línea es incidente con el punto que le antecede y con el que le sigue. Un camino une v_0 y v_n y puede denotarse también v_0, v_1, ..., v_n (las líneas evidentes). Es una *trayectoria* si todos los puntos son distintos. Si los n puntos son distintos, n >= 3 y $v_0 = v_n$, entonces el camino se llama un *ciclo.*

El *grado* de un punto v_i en el grafo G, denotado grad v_i es el número de líneas incidentes a v_i. El primer teorema de teoría de grafos es (Euler):

Teorema. La suma de los grados de los puntos de un grafo G es dos veces el número de líneas:

$$\sum_{v_i \in V} \text{grad } v_i = 2q$$

Definiciones. Un *grafo completo* K_p tiene cada par de sus p puntos adyacentes. Un grafo es *conexo* si cada par de puntos están unidos por una trayectoria. Un grafo es *acíclico* si no tiene ciclos.

[anterior] [home] [siguiente]

3.2. Árboles

Definición. Un *árbol* es un grafo acíclico conexo.

Teorema. Las siguientes proposiciones son equivalentes para un grafo G:

1. G es un árbol
2. Cualesquiera dos puntos de G se unen por una trayectoria única
3. G es conexo y p = q + 1
4. G es acíclico y p = q + 1

Definición. Un grafo con pesos en los arcos es G = (V, X) con un conjunto $\{w_1, ..., w_q\}$ asociado a los arcos.

Ejemplo: En una zona aislada existe un conjunto V = $\{v_1, ..., v_p\}$ de comunidades. También existen caminos entre comunidades, de modo que el grafo es conexo. Ahora se quiere proveerlos de una red eléctrica, con la limitación de que los postes deben ir junto a los caminos (por economizar). Encontrar la red eléctrica que minimice el largo total a establecer.

Pre-solución. Basta un árbol (¿por qué?). Entonces buscamos un subgrafo (árbol) de recubrimiento de G de costo mínimo, donde el costo es la suma de los pesos, que en este caso son los largos de los caminos que serán parte del árbol.

Para encontrar un *árbol de recubrimiento mínimo* hay varios algoritmos. Mostraremos uno de ellos.

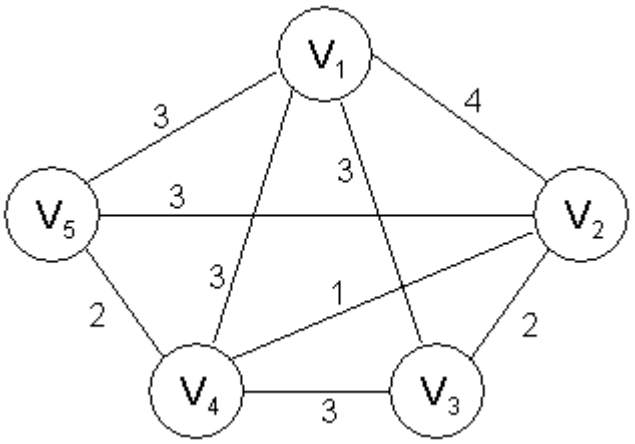
Algoritmo de Prim

Input: Un grafo conexo con pesos G = (V, X), con V = $\{v_1, ..., v_p\}$
Output: Un árbol de recubrimiento mínimo T
Método: Expandir el conjunto U Í V desde $\{v_1\}$ a V usando el arco de mínimo peso de U a V-U.

1. **Sea t ¬ v_1, T ¬ Æ, y U ¬ $\{v_1\}$**
2. **Sea w(t, u) ¬ $\min_{v \in V-U} \{w(t, v)\}$**
3. **T ¬ T È {e} donde e = tu tiene peso w(t, u)**
4. **U ¬ U È {u}**
5. **Si U = V entonces pare.**
6. **Para cada v Î V-U, sea w(t, v) ¬ min{w(t, v), w(u, v)}**

7. Vaya al paso 2

Ejemplo. Con el grafo



1. t ¬ v_1, T ¬ Æ, y U ¬ {v_1}

2-3. w(t, u) = 3, T ¬ T È {v_1 v_3} (podría haberse elegido cualquiera de v_3, v_4 o v_5)

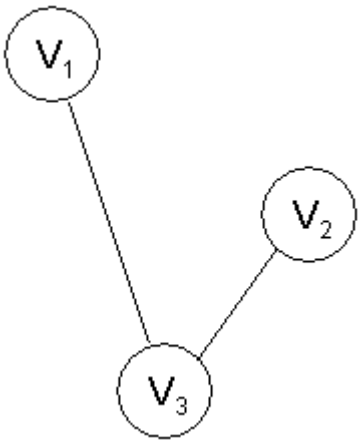
4-5. U ¬ U È {v_1}, continuamos

6. w(t, v_2) = 2, w(t, v_4) = 3, w(t, v_5) = 3 y continuamos

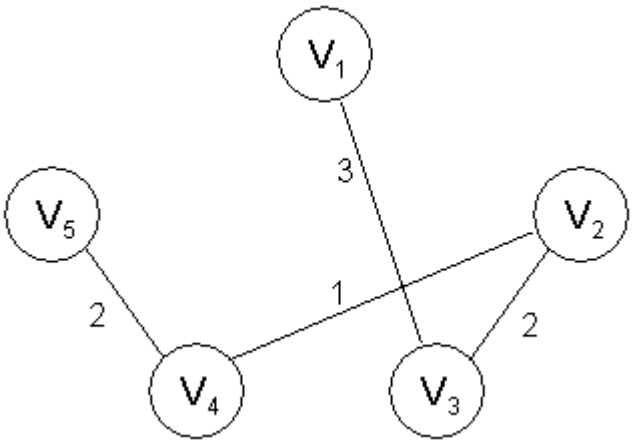
2-3. w(t, u) = 2 y T ¬ T È {v_2 v_3} = {v_1 v_3, v_2 v_3}

4-5. U ¬ U È {v_2} = {v_1, v_3, v_2} y continuamos

6. w(t, v_4) = 1, y w(t, v_5) = 3 y continuamos



Si continuamos, obtenemos:



U = {v_1, v_3, v_2, v_4, v_5}

T = {v_1v_3, v_3v_2, v_2v_4, v_4v_5}

Costo total: 3 + 2 + 1 + 2 = 8

El algoritmo de Prim tiene complejidad tiempo O($|V|^2$).

El problema de encontrar el árbol de recubrimiento mínimo con pesos para digrafos es mucho más difícil. De hecho, no se conoce un algoritmo polinomial para resolver este problema.

[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

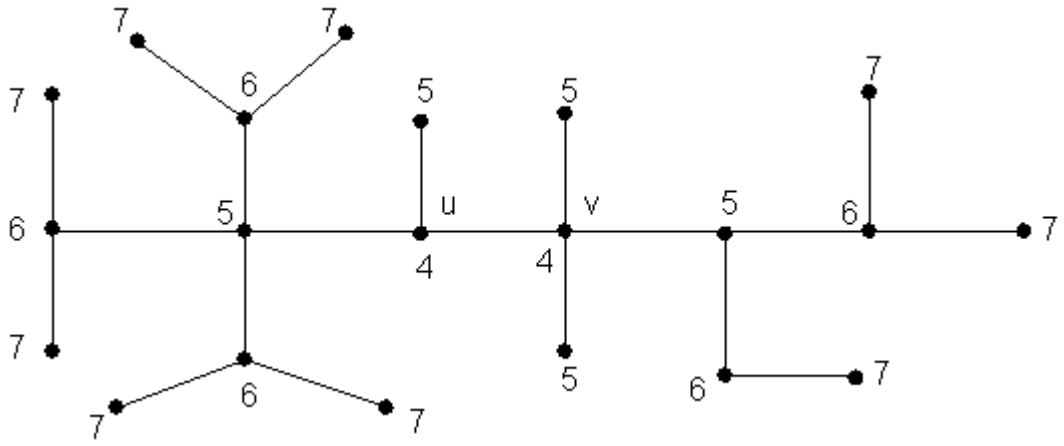
3.3. Centralidad en árboles

En un grafo, en general G(V, X):

Definición. La *distancia* d(u, v) entre dos puntos u, v Î V es el largo de la trayectoria más corta que los une, si es que la hay; en caso contrario, d(u, v) = ¥.

Definición. La *excentricidad* e(v) de un punto v en un grafo conexo G es max d(u, v), para todo u en G. El *radio* r(G) es la excentricidad mínima de los puntos. Un punto v es un *punto central* si e(v) = r(G), y el *centro* de G es el conjunto de todos los puntos centrales.

Ejemplo. Excentricidades de cada punto.



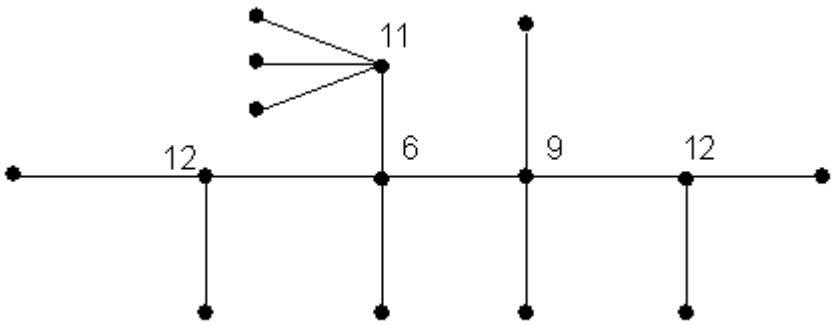
En este ejemplo, el radio es 4 y el centro consiste de dos puntos, u y v, cada uno con excentricidad mínima 4.

Teorema. Cada árbol tiene un centro consistente de uno o dos puntos adyacentes.

Definición. Una *rama* en un punto u de un árbol T es un subárbol maximal que contiene u como un vértice final ("hoja"). Así, el número de ramas en u es grad u.

Definición. El *peso* en un punto u de T (T árbol) es el número máximo de líneas en cualquier rama de u.

Los pesos en los puntos no-hojas se muestran en este ejemplo:



¿Cuánto es el peso en las hojas?

Definición. Un punto v es un *punto centroide* de un árbol T si v tiene mínimo peso, y el *centroide* de T consiste de todos estos puntos.

Teorema. Todo árbol tiene un centroide que consiste de uno o dos puntos adyacentes.

La definición de centro y centroide nos permite modelar la solución de varios problemas de centralidad en árboles. Por ejemplo:

Problema 1. Suponga que varias ciudades están conectadas por un árbol de caminos. El problema es dónde ubicar un local de pizzas, de modo de servir a todas estas ciudades, y minimizar las distancias recorridas de modo que las pizzas siempre lleguen calientes, independientemente de a dónde haya que despacharlas.

En el caso general, cada arco tiene una *distancia* asociada, y el problema es encontrar el *centro*.

La excentricidad de cada vértice v es definida como: $e(v) = \max_{x \in V} \{d(v, x)\}$

Problema 2. Nuevamente tenemos varias ciudades, y ahora se trata de ubicar una planta termoeléctrica en una de ellas, de modo de surtir a todas. Existe un árbol de distribución eléctrica. Cada vértice tiene una demanda c(v). Los arcos pierden fluido eléctrico, de manera que para proveer b unidades de electricidad al extremo v del arco (u, v), debe entregarse $g_{(u, v)}$ (b) unidades en el extremo u. En este caso, la excentricidad de cada vértice v es el número de unidades de electricidad requeridos en v para suplir toda la demanda desde ese vértice.

Problema 3. Suponga que un banco tiene oficinas en todo el país. Dispone de un sistema de distribución de mensajes electrónicos que es un árbol. Para simplificar, consideremos que todo el tráfico va desde un punto de distribución central a los vértices. Se quiere minimizar el máximo tráfico de mensajes transportados por un arco.

Si v es el punto central, el máximo tráfico estará en uno de los arcos (v, x) que salen de v.

Definimos $e(v) = \max \{ t_{(v, x)} / x \in Ady(v) \}$

Ady(v) = conjunto de vértices adyacentes a V (formalmente: $Ady(v) = \{ u / u \in V$ y $(u, v) \in X\}$)

Cuando cada vértice recibe una unidad de tráfico, el ejemplo se reduce a encontrar el centroide del árbol.

Existen otros problemas de centralidad en árboles (ver referencias). En muchos de ellos, además de encontrar el punto central, estamos interesados en calcular todas las excentricidades. Por ejemplo, en el problema 2, si se encuentra la ciudad con mínima excentricidad, quizás el alcalde no quiera la planta termoeléctrica en su ciudad. Si se tienen las excentricidades se puede saber cuán preferible es esa ciudad comparada con una segunda, cuyo alcalde está interesado en la ubicación de la planta debido a las plazas de trabajo que creará.

Estudiaremos un algoritmo lineal que permite resolver estos problemas de centralidad en árboles. Veamos alguna terminología adicional.

Definición. Un árbol con raiz $T(V, X, r_0)$ es un árbol tal que:

i) $r_0 \in V$ es la raiz.

ii) Para cualquier $(u, v) \in X$, también hay un $(v, u) \in X$ (árbol dirigido, pero todos los arcos tienen un simétrico).

iii) Existe una trayectoria única p(u, v) para cada $u, v \in V$.

Definición. Sea un árbol con raiz $T(V, X, r_0)$, y sea v, w, ..., r_0 una trayectoria, denotada por p (v, r_0), entonces:

w se llama el *padre* de v (r_0 no tiene padre).

Hijos(v) = Ady(v) - {padre(v)}

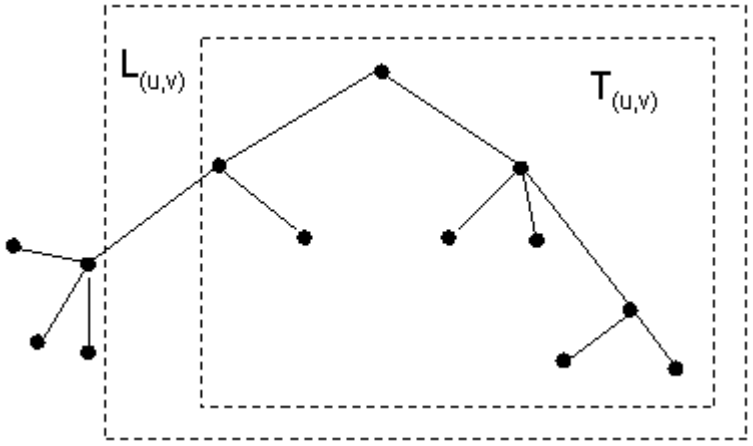
$l_{(u, v)}$ = largo del arco (u, v). Además, $l_{(u, v)} = l_{(v, u)}$

$d_{u, z} = \sum_{xi, xj \in p(u, z)} l_{(xi, xj)}$

$\{c_j : V \to R / j = 1, ..., p\}$ funciones de costo de vértices.

$\{g_{(u, v)} : R \to R / (u, v) \in X\}$ funciones de costo de lados.

Con la siguiente figura como ejemplo, definimos los conceptos de sub-árbol y rama.



Definición. El sub-árbol $T_{(u, v)}(V_{(u, v)}, X_{(u, v)}, v)$ es un triple donde:

$(u, v) \in X$

$V_{(u, v)} = \{ w \in V / u \in p(v, w) \} \cup \{ v \}$

$X_{(u, v)} = \{ (x, y) / (x, y) \in X; x, y \in V_{(u, v)} \}$

Definición. La rama $L_{(u, v)}(V_{(u, v)}, X_{(u, v)}, (u, v))$ es un triple donde:

$(u, v) \in X$

$V_{(u, v)} = \{ w \in V / u \in p(v, w) \} \cup \{ v \}$

$X_{(u, v)} = \{ (x, y) / (x, y) \in X; x, y \in V_{(u, v)} \} \cup \{ (u, v), (v, u) \}$

En lo que sigue, simplificaremos la notación así:

$T(V, X, r_0) \to T$

$T_{(u, v)}(V_{(u, v)}, X_{(u, v)}, v) \to T_{(u, v)}$

$L_{(u, v)}(V_{(u, v)}, X_{(u, v)}, (u, v)) \to L_{(u, v)}$

A cada rama $L_{(u, v)}$ le asociamos un "peso" que llamamos wt(u, v), a partir del cual calcularemos excentricidades.

El cálculo de los pesos se hace a partir de los datos del problema de centralidad específico y con los pesos que ya se han calculado. Los datos del problema pueden ser largos de arcos, funciones de costo en los vértices o funciones de costo en los arcos.

El peso wt(u, v) de una rama $L_{(u, v)}$ es un *resumen* de la información acerca de la rama.

Definición.

$$WTS(v) = \{ wt(v, x) / x \in Ady(v) \}$$

$$OTHER\_WTS(u, v) = WTS(u) - \{ wt(u, v) \}$$

Antes de que veamos un algoritmo general que nos permitirá resolver los problemas de centralidad en árboles, necesitamos dos pasos previos: estudiar recorridos de árboles y definir algunas funciones.

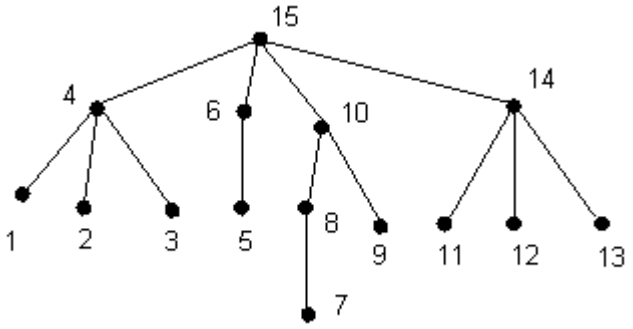
[anterior] [home] [siguiente]

[anterior] [home] [siguiente]

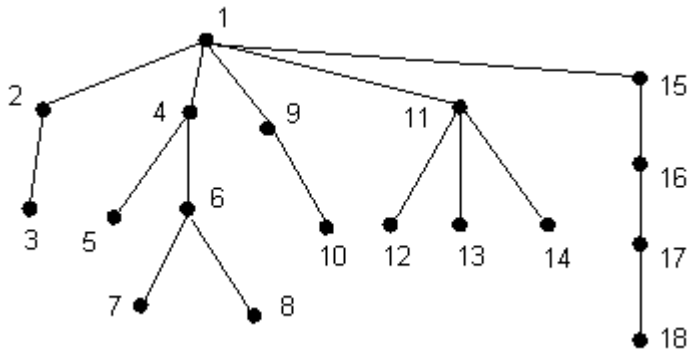
3.4. Recorrido de árboles

Suponga un árbol T con raiz r_0. Queremos visitar (recorrer) cada nodo una vez. Hay varias maneras de hacerlo.

Recorrido desde abajo ("bottom-up"). Comenzando en la raiz, visite cada nodo sólo después de haber visitado a todos sus hijos (de izquierda a derecha). Ejemplo:



Recorrido desde arriba ("top-down"). Comenzando con la raiz, visite cada nodo antes de visitar a cualquier hijo. Continúe con los hijos antes de visitar un hermano. Ejemplo:



Funciones relacionadas con pesos

Para cada problema de centralidad que se desee resolver deben existir dos funciones llamadas COMBINE y EXTRACT, tales que:

e(v) = EXTRACT(v, WTS(v))

wt(u, v) = COMBINE(u, v, OTHER_WTS(v, u))

Observemos que cuando v es una hoja, OTHER_WTS(v, u) está vacío. Así, es posible ir calculando los pesos wt(u, v) "subiendo" en el árbol, es decir, con un recorrido desde abajo.

El algoritmo que estudiaremos hace dos recorridos del árbol. En el primero, desde abajo, calcula wt(padre(v), v). En el segundo, desde arriba, calcula wt(v, padre(v)) y las excentricidades.

Para calcular los pesos wt(padre(v), v), necesitamos una cantidad que *resuma* los aportes de todas las ramas incidentes a v. Esta cantidad la llamaremos SMR(v). El peso wt(padre(v), v) será entonces SMR(v), agregando información sobre el vértice v y *extrayendo* el peso de la rama(v, padre(v)).

Por ejemplo, en el problema 3:

wt(w, v) = $\sum_{x \in V(w,v)}$ **c(x)** **Entonces:**

SMR(v) = $\sum_{x \in Ady(v)}$ **wt(v, x)**

wt(w, v) = SMR(v) - wt(v, w) + c(v)

Finalmente, la opción COMBINE vamos a descomponerla en dos funciones: MAKESUM y SUBTRACT (en el caso general), para el segundo recorrido:

SMR(v) = MAKESUM(v, WTS(v))

wt(w, v) = SUBTRACT(v, w, SMR(v), wt(v, w))

Con esto, podemos presentar el algoritmo Rosenthal-Pino.

Algoritmo de Rosenthal-Pino

Input: Un árbol con raiz.
Output: Excentricidades para todos los vértices.
Método: Dos recorridos sobre el árbol. Utiliza funiones COMBINE, EXTRACT, MAKESUM, SUBTRACT (dependientes del problema).

1. v ¬ primer vértice de recorrido desde abajo

2. WHILE v ^{1} r_o
BEGIN

3. wt(padre(v), v) ¬ COMBINE(padre(v), v, OTHER_WTS(v, padre(v))); /* note que lo que está en OTHER_WST( ) se calculó cuando se procesó Hijos(v) */

4. v ¬ sucesor(v) en recorrido desde abajo;
END; 5. v ¬ r_o; /* segundo recorrido */

6. e(r_o) ¬ EXTRACT(r_o, WTS(r_o)); /* lo en WTS(r_o) ya calculado */
REPEAT

7. SMR(v) ¬ MAKESUM(v, WTS(v));

8. FOR cada hijo w de v :
BEGIN

9. wt(w, v) ¬ SUBTRACT(v, w, SMR(v), wt(v, w));

10. e(w) ¬ EXTRACT(w, WTS(w));
END

11. v ¬ sucesor(v) en recorrido desde arriba; /* exceptuando hojas */
UNTIL se visitan todos los vértices no hojas
END;

Veamos ahora cómo podemos aplicar este algoritmo a los tres problemas que planteamos.

Problema 1. (centro)
Como entrada están los largos de cada arco $l_{(u, v)}$. El peso de una rama $L_{(padre(v), v)}$ es la distancia máxima de padre(v) a cualquier vértice z en la rama. Pero la trayectoria de padre(v) a z debe tener a v como el primer vértice después de padre(v). Así:

3. wt(padre(v), v) ¬ max(OTHER_WTS(v, padre(v))) + $l_{(padre(v), v)}$

El resumen SMR deun vértice v es un vector de dos componentes. SMR_1 es la distancia máxima desde v a cualquier punto en el árbol (¡así que también es la excentricidad de v!). Si u es el primer vértice en una trayectoria de v a un vértice en el cual se logra SMR_1(v), entonces SMR_2(v) es la distancia más larga de v a cualquier punto donde la trayectoria no pasa por u.

SMR_1(v) ¬ max(WTS(v))

Elija u tal que wt(v, w) = SMR_1(v). (cualquier u que cumpla esto es aceptable, si hay más de uno). Entonces:

SMR_2(f(v)) ¬ max(OTHER_WTS(v, w))

y así:

9. wt(w, v) ¬ { SMR_1(v) + $l_{(w, v)}$ si wt(v, w) ¹ SMR_1(v), o SMR_2(v) + $l_{(w, v)}$ en caso contrario }

10. e(w) ¬ SMR_1(w)

Problema 2 (red con pérdidas)
Las entradas son las demandas c: V ® R y las funciones de los arcos $g_{(u, v)}$: R ® R.

El peso de una rama $L_{(padre(v), v)}$ es el número de unidades de energía que se necesitan en padre(v) para proveer los vértices de la rama.

3. wt(padre(v), v) ¬ $g_{(padre(v), v)}$(total(OTHER_WTS(padre(v), v)) + c(v))

El resumen SMR de un vértice es la energía total que necesita para proveer los vértices que van desde v (no incluyéndolo):

SMR(v) ¬ total(WTS(v))

9. wt(w, v) ¬ $g_{(w, v)}$(SMR(v) - wt(v, w) + c(v))

10. e(w) ¬ total(WTS(w))

Problema 3 (centroide)
Las entradas son las demandas en los vértices c: V ® R. El peso de una rama es la demanda total de los vértices de la rama:

3. wt(padre(v), v) ¬ total(OTHER_WTS(v, padre(v))) + c(v)

El resumen SMR de un vértice v esla demanda total del árbol desde vértices distintos de v.

SMR(v) ¬ total(WTS(v))

9. wt(w, v) ¬ SMR(v) - wt(v, w) + c(v)

10. e(w) ¬ max(WTS(w))

[anterior] [home] [siguiente]