MA3705. Algoritmos Combinatoriales 2020.
Profesor: José Soto
Escriba(s): Carlos Antil y Javiera Gutierrez.
Fecha: 31 de Agosto de 2020.

Cátedra 1

1. Introducción

En este ramo se hará una introducción a tres áreas de la Matemática.

- **Matemáticas Discretas:** En esta área se estudian estructuras finitas (Grafos, Redes, Matroides). Para esta parte del curso es necesario tener una noción de los contenidos básicos de Álgebra Lineal.
- **Optimización Matemática:** Área que busca objetos óptimos en un conjunto finito. Como prerrequisito, para iniciarse en esta área, es necesario tener conocimiento previo de Optimización, sobretodo la parte Lineal, que es la que más se utilizará.
- **Algoritmos y Teoría de Computación:** En simples palabras los Algoritmos son métodos para resolver ciertos tipos de problemas, para esta etapa del curso necesitamos previas nociones básicas de programación como el control de flujo de programación.

Para el último punto, cuando se analicen algoritmos, se buscará estudiar la *eficiencia del algoritmo*.

2. Optimización Combinatorial

2.1. Estudio de problemas de Optimización (en conjuntos finitos)

Un problema es un conjunto de **instancias** con codificación común (que se escriben de cierta manera similar)

Definición 1. (Instancia) Es una tripleta de la forma $I = (\text{Dominio}, F, \text{Objetivo})$ donde:

1. Dominio: conjunto de objetos llamados factibles de dimensión finita.
2. F: es la función a optimizar.
3. Objetivo: Indican el sentido a optimizar (minimizar o maximizar).

Es importante notar que normalmente estos tres elementos no se escriben de manera explícita sino que se puede recuperar implícitamente de la codificación elegida.

Ejemplo 1 (Problema de asignación).

Consideremos los siguientes datos:

- Un conjunto A de n tareas y un conjunto B de n máquinas.
- Costos $c_{ij} \in \mathbb{R}_+$, para cada $i \in A$, $j \in B$ que representan el costo de asignar tarea i a máquina j.

Lo que queremos lograr:

- Asignar una tarea a cada máquina (de forma biyectiva).
- Minimizar la suma de los costos involucrados.

Con lo expuesto anteriormente ¿como recuperamos la instancia?

a) Dominio: Son todas las funciones φ biyectivas que parten de A y llegan a B.

b) Función: La función que se quiere minimizar es de la forma $F(\phi) = \sum_{i \in A} c_{i\phi(i)}$.

c) Objetivo: Minimizar

Ejemplo 2 (Problema del vendedor viajero)**.**

Consideremos los siguientes datos:

- Un conjunto A de n ciudades (en el plano)

Lo que desea

- Encontrar un Paseo Hamiltoniano que visite a todas las ciudades (es decir, cada ciudad es visitada exactamente una vez y se debe terminar en la ciudad inicial).
- Minimizar el largo total

Con lo expuesto anteriormente: ¿Como codificamos la instancia?

a) Dominio: Son todos los paseos Hamiltonianos posibles.

b) Función: la función a minimizar es la distancia total recorrida en un paseo Hamiltoniano.

c) Objetivo: Minimizar

Obs: Si el dominio de una instancia es finito y no vacío siempre existe un óptimo (solución). No siempre es evidente que el dominio no es vacío.

Luego de haber enunciado los problemas anteriores, nos hacemos la siguiente pregunta:

¿Qué es resolver un problema de optimización combinatorial $\mathcal{P}$?

La respuesta intuitiva sería desarrollar un método capaz de resolver el problema planteado, encontrando una solución óptima en cualquier instancia $\mathcal{P}$ o garantizar la no existencia de una solución (declarando el dominio vacío). A este método le llamaremos **Algoritmo**.

2.2. Algoritmos de Optimización Combinatorial

Definición 2. (Algoritmo) Un algoritmo, para un problema $\mathcal{P}$, es un método que cumple las siguientes propiedades:

1. Recibe una instancia (codificada) como entrada.
2. Ejecuta una secuencia de instrucciones básicas.
3. Devuelve una salida.

Definición 3. (Correctitud) Un algoritmo ALG, para un problema $\mathcal{P}$, es correcto si al aplicar ALG sobre cualquier instancia de $\mathcal{P}$, el algoritmo:

1. Termina.
2. Entrega la solución correcta al terminar.

Donde algo no correcto seria que nos retorne siempre una constante independiente de lo que le entreguemos al algoritmo o que quede en un *loop* y no sea capaz de terminar.

Ejemplo 3. Veamos un ejemplo de un (mal) algoritmo.

Tomando el Ejemplo 1 podemos crear un algoritmo que genere las $n!$ funciones biyectivas, calcule el costo asociada a cada una de ellas y que nos devuelva la mejor opción.

Este algoritmo es correcto pero es muy pesado computacionalmente por lo que no es eficiente. Para visualizar un poco a que nos referimos tomaremos $n = 25$, si pensamos en que poseemos un computador capaz de generar cada *nanosegundo* una nueva permutación (o una nueva asignación) utilizando el algoritmo anterior tendríamos 25! *nanosegundos* lo cual equivale a **491531084** años, lo cual escapa de nuestras posibilidades.

Con el ejemplo anterior se puede tener una idea intuitiva de las características principales que debe tener un buen algoritmo: **Correcto** y **Eficiente**.

Además, es necesario tener en consideración lo siguiente:

- El tiempo de resolución depende del tamaño de la entrada.
- Debe ser una función de crecimiento pequeño.

Durante el curso formalizaremos el concepto de algoritmo eficiente. Para llegar a eso necesitamos un lenguaje apropiado para estudiar los problemas de optimización combinatorial.

3. Grafos

3.1. Definiciones Básicas

Definición 4. (Grafo Simple) Sea $G = (V, E)$, $E \subseteq \binom{V}{2} = \{\{u, v\} : u, v \in V; u \neq v\}$. Llamamos vértices a todos los elementos $v \in V$ y aristas a todos los elementos $e \in E$.

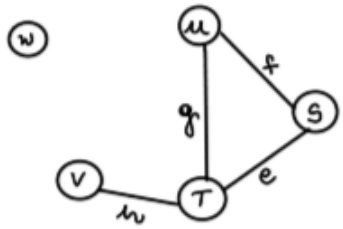

Figura 1: Grafo Simple

Definición 5. (Multigrafo) Sea $G = (V, E)$, donde V y E son los conjuntos de vértices y aristas respectivamente. Cada arista $e \in E$ posee uno o dos extremos.

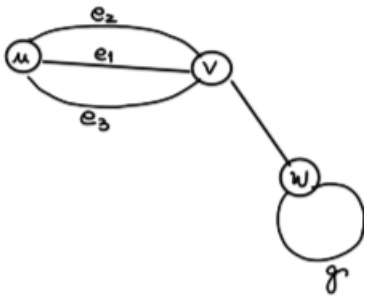

Figura 2: Multigrafo

Donde $g \in E$ posee solo un extremo y e_1, e_2, $e_3 \in E$ son aristas *paralelas* que tienen los mismos extremos u y v.

Definición 6. (Dígrafo) Sea $\vec{G} = (V, E)$, donde V y E son los conjuntos de nodos y arcos, respectivamente. Cada arco $e \in E$ tiene una cola $t(e) \in V$ y una cabeza $h(e) \in V$.

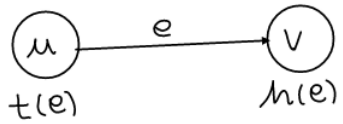

Figura 3: Dígrafo

3.2. Ejemplos y convenciones adicionales

A lo largo del curso usaremos las siguientes convenciones, para simplificar la notación

- Grafo: Para hacer mención que trabajaremos con un Grafo Simple. Todos nuestros grafos serán finitos.
- $V(G)$: Conjunto de vértices del grafo G.
  $E(G)$: Conjunto de aristas del grafo G.

Ejemplo 4. Veamos un ejemplo de un grafo, llamado **Grafo completo** (K_n):

$$K_n = \left([n], \binom{n}{2} = \{\{a, b\} : a, b \in [n], a \neq b\} \right) \tag{1}$$

donde $[n] = \{1, 2, ..., n\}$.

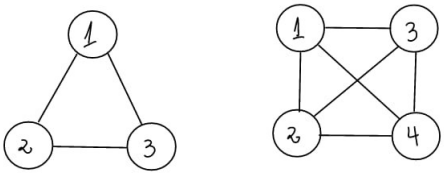

Figura 4: K_3 y K_4

3.3. Definiciones básicas de grafos:

Sea G un grafo, $e \in E(G)$ y $w, u \in V(G)$.

Definición 7. (Extremos): v es extremo de e si $v \in e$, es decir, $e = vw$ con $w \in V(G)$.

Definición 8. (Incidente): e es incidente a v si $v \in e$.

Definición 9. (Adyacentes): 2 aristas son adyacentes si comparten un mismo vértice.

Definición 10. (Vecinos): 2 vértices serán vecinos si existe una arista e que contenga a ambos (incidente a ambos).

Definición 11. Otras definiciones en grafos y multigrafos Sea $G = (V, E)$ grafo o multigrafo. Sean $F \subseteq E$ y $U, W \subseteq V$ con $U \cap W = \varnothing$ y $v \in V$.

1. $F[U, W] \subseteq E$: Aristas de F con un extremo en U y otro en W.
2. $F[U] \subseteq E$: Aristas de F con ambos extremos en U.
3. $\delta_F(W) \subseteq E$: Corte de W con aristas de F. Son aquellas aristas de F con un extremo en W y otro fuera de W.
4. $N_{\mathrm{F}}(W) \subseteq V$: Vecinos de W con aristas de F: son aquellos vértices fuera de W que tienen algún vecino en W
5. $\deg_F(v) \in \mathbb{N}$: Grado de v con aristas de F. Es el numero de aristas incidentes a v.

Solo en grafos simples tenemos la siguiente igualdad:

$$|\delta_F(W)| = |N_F(W)|$$

Ya que en los multigrafos los ciclos aumentan en 2 el grado del vértice inicial del ciclo.