

Algoritmos CC40a- 2005

Clase Auxiliar 5

Profesor: Gonzalo Navarro

Auxiliar: Mauricio Cerda

Problema 1, La Mochila, (Knapsack problem, Programación Dinámica)

Dado un entero K y un número n de objetos de diferentes tamaños, tales que el objeto i -ésimo tiene un tamaño k_i . Encuentre un grupo de objetos cuya suma sea exactamente K , o determine que dicho grupo no existe. (Buscar en Literatura, ¿qué tipo de solución existe para el problema 2D?)

Algoritmos de Ordenamiento

Partiendo con el algoritmo de ordenación por inserción, elabore una versión mejorada en donde el elemento a insertar en la nueva etapa sea el máximo, evitando de esta manera un exceso de intercambios para poder insertar, ¿cuál es ahora el hipótesis inductiva?, escriba el orden en tiempo y espacio del algoritmo (a este se le llama algoritmo por selección).

Ahora se busca mejorar el algoritmo de inserción, notando que en una pasada se puede conocer la ubicación correcta de muchos elementos, ¿Qué algoritmo se está construyendo?, escriba la recurrencia y justifique el costo de insertar dos partes simultaneamente.

El problema del algoritmo anterior es que requiere de memoria adicional, al realizar el Merge de dos grupos. Supogamos por un momento que conocemos un elemento del arreglo tal que la mitad es mayor que el, podemos realizar este orden realtivo en $n - 1$ comparaciones, ¿Cuál sería la hipótesis inductiva que garantiza la correctitud de dicho algoritmo?. Escriba la recurrencia del algoritmo suponiendo cada elemento tiene la misma probabilidad de ser elegido como pivote que otro (no la resuelva, esto se realizó en cátedra), ¿Cuál sería el problema potencial de este algoritmo?.

Ahora, se busca un algoritmo que sea capaz de ordenar sin espacio extra, y que no tenga un peor caso como el anterior, a este se le llama *HeapSort*, muestre cual sería la forma de este algoritmo. (Discuta las dos formas de construir un Heap y el costo de cada una de ellas). Ahora conociendo cada una de las singularidades de los algoritmos de ordenacion, ¿Cuál cree Ud. es el más rápido?, ¿por qué?. ¿Podría existir alguna manera se sobrepasar la cota inferior calculada en cátedra?.

Propuesto: Unificación en Prolog

Un valiente implementador, desea elaborar su propia version de PROLOG (lenguaje de programación lógica), en donde se esté seguro se está realizando una optimización en la forma de unificar.

Dicha optimización consiste en procesar el conjunto de reglas, para saber rapidamente cuales corresponden a una cierta meta. Esto se realiza con un *trie* (cada hoja del árbol representa un String, y cada nodo representa un caracter único), el problema es que para un conjunto de reglas, existen muchos *trie* posibles pues es válido permutar los caracteres en las cabezas de las reglas (ver gráfico), ya que todas las reglas se conocen de antemano. Además como restricción adicional, en PROLOG se debe mantener el orden de las reglas, pues este podría cambiar el resultado final (establece preferencias de unificación). La pregunta es como obtener el *trie* óptimo (la menor cantidad de arcos en el *trie*), recuerde el enfoque efectuado en el problema de triangulizar polígonos en la clase auxiliar anterior (programación dinámica), y utilice la restricción de que las reglas tienen un orden que se debe respetar, elabore un algoritmo para este problema y muestre de que orden sería en espacio al menos (permutaciones posibles).

Código PROLOG (probar en anakena con *pl* y luego 'ARCHIVO.pl' entre corchetes cuadrados).

```
1  verdad(_).  
2  pre(a,a,a):-verdad(a).  
3  pre(b,a,a):-verdad(a), verdad(b).  
4  pre(c,b,b):-verdad(b), verdad(c).  
5  pre(d,b,b):-verdad(d), verdad(b).
```

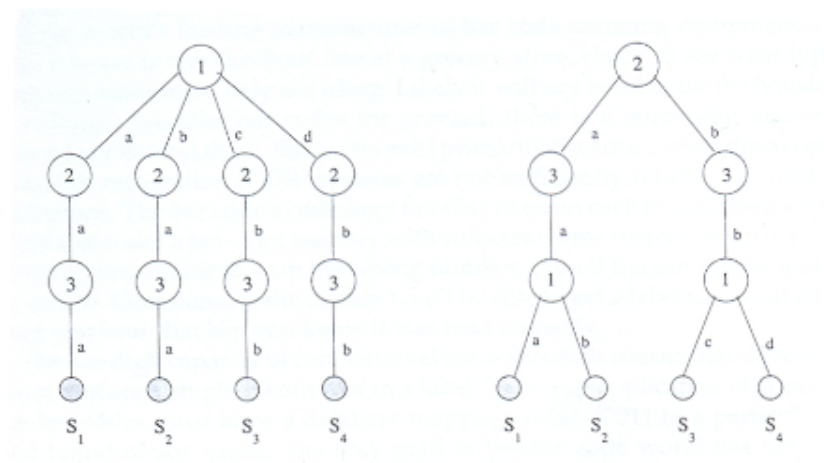


Figure 3-9. Two different tries for the given set of rule heads

Figure 1: