DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Algoritmos y Estructuras de Datos III
Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Bouzón, María Belén	128/13	belenbouzon@hotmail.com
Jiménez, Paula	655/10	puly05@gmail.com
Montepagano, Pablo	205/12	pablo@montepagano.com.ar
Rey, Maximiliano	037/13	rey.maximiliano@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Problema 1: Saliendo del Freezer

1.1. Descripción de la problemática

Este problema modela un edificio de N pisos (y planta baja) en el cual en vez de tener escaleras o ascensores, hay portales que lo transportan a uno desde un piso a otro. Los portales solo permiten ascender de un piso más bajo a uno más alto. En cada piso puede haber cualquier cantidad de portales, teniendo en cuenta que no hay dos portales que salgan del mismo piso y lleguen a un mismo piso. Todas las instancias del problema deben tener al menos un camino de portales entre la planta baja y el piso N.

Se pide diseñar un algoritmo de complejidad O(n^2) que encuentre un camino que llegue desde planta baja hasta el piso N y utilice la mayor cantidad de portales posible.

El formato de entrada es un archivo de texto. Cada problema está representado en dos líneas de texto. La primera es el valor de N. La segunda es una lista de portales que sigue el siguiente formato:

```
pd ph[; pd ph]    // pd (piso desde), ph (piso hasta) enteros, pares de p separados por "; "
```

Para la salida, lo único que interesa es la cantidad de portales que hay que utilizar para recorrer el camino que usa la cantidad máxima posible de portales. La salida es una línea de texto que tiene ese valor.

Por ejemplo, un edificio de cinco pisos que tiene, desde cada piso, un portal para llegar a todos los pisos superiores se escribiría de este modo en el formato de entrada:

```
5
0 1; 0 2; 0 3; 0 4; 0 5; 1 2; 1 3; 1 4; 1 5; 2 3; 2 4; 2 5; 3 4; 3 5; 4 5;
```

Y el resultado a este ejemplo es, por supuesto, 5, ya que el camino más largo es el que pasa por todos los pisos.

1.2. Algoritmo desarrollado

El algoritmo diseñado utiliza la técnica de programación dinámica con un enfoque *bottom-up*. Podemos subdividir el problema original en sub-problemas. Los subproblemas implican resolver lo mismo pero, en vez de partiendo desde la planta baja, partiendo desde un piso más alto. Cuanto más cercano al piso superior (N) esté el piso desde el que partimos, más chico es el subproblema.

En primer lugar *parseamos* los datos de entrada, generando una tabla de tamaño N. La tabla contiene una lista de enteros en cada posición. La lista que se encuentra en la posición i-ésima de la tabla representa a los portales que salen desde el piso i. Los valores de la lista son los pisos a los que se llega desde ese piso usando un portal.

Luego, inicializamos en -1 una tabla de resultados parciales. Nos referimos a resultados parciales porque en la posición i-ésima de la tabla resultados encontraremos la respuesta al subproblema de "máxima cantidad de portales que se pueden utilizar para llegar al piso N partiendo desde el piso i". Si no es posible llegar al piso N desde el piso i, guardaremos el valor -1 en esa posición.

Nótese que ninguna de las dos tablas tiene un lugar para el piso N, ya que no hay portales que partan desde el piso N ni tiene sentido el problema de llegar al piso N desde el mismo piso N.

Finalmente, el algoritmo itera por cada piso del edificio desde el piso $N - 1$ hasta el 0, iterando en cada piso por todos los portales del piso. Para cada portal calculamos la cantidad máxima de portales que se pueden usar para llegar a N usando ese portal. Esto se hace en tiempo constante, ya que hay que verificar solo tres cosas:

1. leer el valor de N del archivo de entrada
2. inicializar la tabla `portales` con listas (de enteros) vacías
3. inicializar la tabla `resultados` con -1
4. llenar la tabla `portales` con los portales de cada piso
5. Para cada piso `p` desde $N - 1$ hasta 0:

5.1. Inicializar variable `MAX` en -1

5.2. Para cada portal `l` en `portales[p]`:

5.2.1. inicializamos una variable `costo` en -1 para representar la máxima cantidad de portales que se pueden utilizar para llegar a N usando el portal `l`

5.2.2. Tenemos tres opciones:

5.2.2.1. Si el portal va directo a N, el costo es 1

5.2.2.2. Si el portal va a un piso que tiene en la tabla `resultados` valor -1, no es posible llegar a N usando ese portal, y dejamos el costo en -1

5.2.2.3. Finalmente, si el portal va a un piso que tiene en la tabla `resultados` un valor distinto a -1, entonces el costo será ese valor + 1 (por el portal `l`)[1]

5.2.3. Ahora que ya tenemos el costo, si el costo obtenido para este portal es mayor a la variable `MAX`, actualizamos el valor de ésta con el costo obtenido en esta iteración

5.3. Guardar en `resultados[p]` en valor de `MAX`

6. El resultado del problema completo se encontrará en `resultados[0]`

1.3. Justificación de correctitud

ACÁ PONER QUE LOS TESTS QUE ESTABAN EN JAVA LOS REPLIQUÉ EN EL ARCHIVO test.sh

1.4. Análisis de la complejidad

En en código fuente hemos comentado las distintas partes indicando los costos de cada una. Si llamamos N a la cantidad de pisos del edificio y P a la cantidad total de portales, la conclusión a la que llegamos es que el algoritmo tiene un costo de $\mathcal{O}(N + P)$.

Ahora bien, la cantidad de portales pertenece a $\mathcal{O}(N^2)$. En un edificio con la máxima cantidad posible de portales tendremos N portales en el piso 0, N-1 portales en el piso 1, etc. hasta el piso N-1, donde tendremos un solo portal. Esa sumatoria se escribe de este modo:

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} = \frac{N^2 + N}{2}$$

Por lo tanto, podemos reescribir el costo del algoritmo en función de N como perteneciente a $\mathcal{O}(N + N^2)$, lo cual pertenece a $\mathcal{O}(N^2)$, que es lo que se pide en el enunciado.

1.5. Código fuente

```
int resolver(){
     int i = n-1;
     while (i>=0){
        int max = -1;
        for (std::list<unsigned int>::iterator it=portales[i].begin(); it != portales[i].end(); ++it){
            int costo = -1;
            if (*it == n){
                // el portal va directo a N
                costo = 1;
            } else if (resultados[*it] == -1){
                // no se puede llegar a N por este portal
                costo = -1;
            } else{
                costo = resultados[*it] + 1;
            }
            if (costo > max){
                max = costo;
            }
        }
        resultados[i] = max;
        i--;
    }
    int res = resultados[0];
    return res;
```

[1] Nótese que, como los portales siempre van a pisos más alto, las posiciones de `resultados` de los pisos más altos ya van a haber sido completadas en iteraciones anteriores del algoritmo.

```
25 }
```

1.6. Experimentación

PONER ACÁ CÓMO COMPILAR Y EJECUTAR EL PROGRAMA

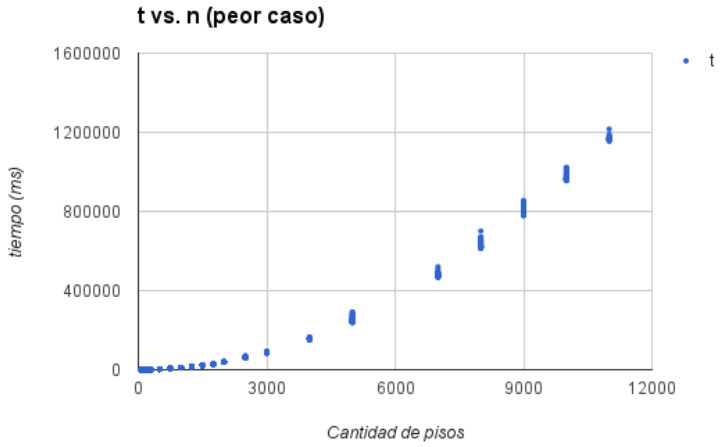

Figura 1: Relación entre tiempo de ejecución y cantidad de pisos (peor caso).

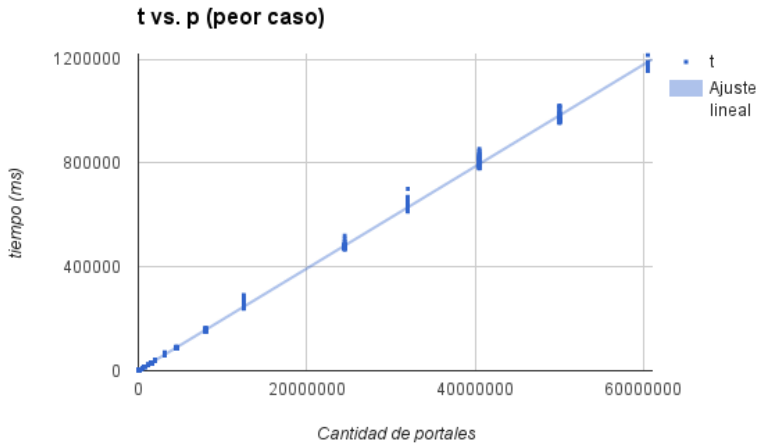

Figura 2: Relación entre tiempo de ejecución y cantidad de portales (peor caso).

1.6.1. Constrastación Empírica de la complejidad

2. Problema 2: Algo Rush

2.1. Descripción de la problemática

En este caso, se tiene un edificio con N pisos y con pasillos de longitud L y existen portales bidireccionales que comunican dos puntos determinados del mismo. Utilizar un portal consume dos segundos y caminar un metro consume un segundo. Se sabe que no hay más de un portal que comunique las mismas posiciones del mismo par de pisos. Se requiere un algoritmo que determine la cantidad minima de segundos en ir desde el inicio del piso 0 hasta el final del último piso.

2.2. Resolución propuesta y justificación

Lo primero que hace el algoritmo es transformar la entrada en un grafo. Según nuestra representación, cada portal es considerado como un nodo. Las aristas que parten de un nodo llegan a todos aquellos otros nodos a los que se puede alcanzar desde el actual, ya sea porque son un punto de destino de teletransporte del mismo, o porque se encuentran en el mismo piso y se puede alcanzar caminando.

Como estructura auxiliar para ocnstruir el grafo se utiliza un diccionario de diccionarios de nodos, cuyas claves para acceder a un nodo son el piso y la distaancia respectivamente. Un nodo contiene: un identificador (un int propio de cada nodo), la posicion en el pasillo donde se encuentra, un conjunto de nodos a los cuales se puede llegar caminando y un conjunto de nodos que son puntos de destino de teletransporte.

Posterirmente un segundo algoritmo, utilizando el grafo generado por el primero, se encarga de determinar el camino entre ambos extremos del grafo (desde el principio del piso 0 hasta el final del último piso). Para esto existe una función que calcula la distancia desde cualquier nodo hasta el extremo superior, la cual considera el camino minimo entre los nodos adyacentes y el extremo del último piso y, cuando el nodo buscado no estaba calculado anteriormente, se llama recursivamente a la función.

2.3. Análisis de la complejidad

Inicialmente, se debe construir un diccionario de N elementos y luego N diccionarios de L elementos, lo cual tiene un costo de O(nL). Luego se procesa la entrada que determina las posiciones de los portales y se van creando los nodos y clasificando en los diccionarios. Como la consulta y escritura en el diccionario es O(1), la complejidad total es de O(P). Finalmente, la complejidad de formar el grafo es de O(P+NL). Para la busqueda del camino mínimo se crea inicialmente un diccionario de N*L elementos, lo cual tiene una complejidad de O(NL). Luego comienza a ejecutarse el algoritmo recursivo. Como el algoritmo nunca recorre dos veces el mismo nodo (ya que cada vez que resuelve el valor del camino mínimo para un nodo lo almacena en el diccionario) la complejidad es O(P). Finalmente, el costo de construir el grafo y hallar el camino mínimo entre sus extremos es O(P+ NL).

2.4. Código fuente

2.5. Experimentación

2.5.1. Constrastación Empírica de la complejidad

3. Problema 3: Perdidos en los Pasillos

3.1. Descripción de la problemática

En este ejercicio se nos presenta un pabellón con M pasillos de distintas longitudes (potencialmente) y un conjunto vértices que pueden ser tanto intersecciones en las cuales dos o más de ellos convergen, o extremos incididos por un sólo corredor. A partir de este contexto se nos pide desarrollar un algoritmo que elimine cualquier ciclo posible del grafo dado, logrando crear partiendo del mismo un árbol generador cuyo peso (dado por la sumatoria de los pesos de cada arista) sea mayor o igual al de cualquier otro árbol generador posible (conocido como árbol generador máximo).

3.2. Resolución propuesta y justificación

Para resolver el problema, desarrollamos un método que hace uso de una adaptación del algoritmo de Kruskal, mediante el cuál es posible encontrar un árbol generador mínimo con una complejidad de O(m log m) (siendo m la cantidad de aristas).
El algoritmo propuesto ordena inicialmente cada pasillo de acuerdo a sus longitudes y los toma uno a uno de mayor a menor verificando si conectan vértices entre los cuales ya existe un camino o no. De ser cierto, la arista en cuestión es descartada. De ser falso, la misma pasa a ser parte del conjunto solución y se repite el procedimiento con la siguiente hasta que se hayan analizado todas.
Para que las complejidades respetaran los requerimientos, fue necesario desarrollar... (find & union) COMPLETAR
El pseudocódigo de nuestro algoritmo es el siguiente:

```
Ordenar pasillos de acuerdo a su longitud.
for pasillo p in pasillos do
  if El pasillo conecta dos intersecciones que no tenían demarcado un camino previamente then
    Agregar el pasillo al conjunto solución.
    Sumar la longitud del pasillo a la solución.
  end if
end for
```

3.3. Análisis de la complejidad

3.4. Código fuente

3.5. Experimentación

Pensar casos borde

3.5.1. Constrastación Empírica de la complejidad