Práctico 7

BackTracking

NOTA

En la formalización se debe especificar:

- forma de la solución
- restricciones explícitas
- restricciones implícitas
- predicado/s de poda
- función/es objetivo

EJERCICIO 1 (R)

Una casa de música ofrece como servicio un sitio web en el cual permite la descarga de ringtones. La casa de música ofrece *n* ringtones diferentes, donde de cada uno de estos se conoce: su precio, su ranking y su tamaño de descarga en bytes.

Un adolescente quiere descargar un conjunto de ringtones tal que, el tamaño de descarga total sea el más bajo posible, el precio total de dicha elección no supere su presupuesto P, y a su vez, el total del valor del ranking de los ringtones seleccionados, sea como mínimo R. Téngase en cuenta que no se pueden repetir ringtones.

Asuma que se tienen definidas las siguientes funciones:

- *p(k)* indica el precio del *k*-ésimo ringtone con $1 \leq k \leq n$.
- *r(k)* indica el ranking del *k*-ésimo ringtone con $1 \leq k \leq n$.
- *d(k)* indica el tamaño de descarga en bytes del *k*-ésimo ringtone con $1 \leq k \leq n$.

Se pide:

a) Formalizar el problema en términos de Backtracking. Indicar: forma de la solución, restricciones explícitas e implícitas, función objetivo y predicados de poda.

b) Implementar la solución utilizando la estrategia BackTracking, indicando las porciones de código que se corresponden con las diferentes partes de la formalización.

EJERCICIO 2 (I)

Dado un grafo *G = (V, E)* con costos no negativos asociados a sus aristas y dos vértices $u, v \in V$, se quiere encontrar el camino de menor costo entre ellos.

Se pide formalizar el problema en términos de Backtracking. Para cada sección/ítem de la formalización debe justificar porqué coloca una expresión cualquiera en la misma.

EJERCICIO 3 (I)

Se tiene una red ferroviaria con n estaciones $E_1,\ldots, E_n$. En cada estación hay depositada una carga C_i. Se desea hacer correr un tren que partiendo de una estación E dada, recorra todas las estaciones, levantando en cada una toda la carga existente y que vuelva a la estación original E. No se podrá hacer pasar el tren por una estación que no tenga carga para levantar, salvo E.

a) Formalizar en términos de Backtracking que dada la red y la estación origen E, determine si es posible realizar tal recorrida. En caso de ser posible, debe devolver una lista que indique el orden en que fueron visitadas las estaciones.
b) Implementar la solución al problema de la parte a) utilizando la estrategia BackTracking, indicando las porciones de código que se corresponden con las diferentes partes de la formalización de la parte a).
c) Modifique el algoritmo de la parte b) para que ahora se determine todas las posibles formas de recorrer la red de la manera indicada (si existe alguna).

EJERCICIO 4 (I)

Una planta productora de aceites tiene almacenado a granel, un cierto volumen de uno de sus productos, el cual desea envasar. Para realizar dicha operación cuenta con una determinada cantidad de envases de varios tipos $e_1,..., e_n$. De cada envase e_i, se conoce su capacidad c_i y la cantidad disponible k_i.

Suponga que el conjunto de envases se encuentra ordenado en orden decreciente de capacidad.

Dado un cierto volumen V del producto a envasar, utilizando la técnica de *backtracking*:

a) Se quiere determinar todas las formas posibles de envasar V con los envases que se dispone. Todos los envases que se utilicen deberán quedar completamente llenos, salvo uno, que podrá utilizarse en forma parcial
    i. Formalizar en términos de Backtracking el problema anterior.
    ii. Implementar la solución al problema anterior utilizando la estrategia BackTracking, indicando las porciones de código que se corresponden con las diferentes partes de la formalización de la parte i).
b) Suponiendo un costo T_i para cada envase, modificar la formalización y algoritmo de la parte anterior, a efectos de recortar el espacio de soluciones y determinar la combinación que permita realizar la operación con el menor costo.

EJERCICIO 5 (R)

Un grupo musical desea editar un CD con una recopilación de sus canciones. El objetivo es que dicho CD tenga la máxima duración posible, sabiendo que un CD puede contener hasta T minutos de música.

Se dispone de las N canciones del grupo ordenadas cronológicamente y de la duración en segundos de cada una de ellas. Las canciones del CD a editar deben mantener el orden cronológico y no pueden ser recortadas.

Se pide:

a) Resuelva por Baktracking, formalizando y escribiendo un algoritmo que reciba un vector *duracion* de tamaño N representando las duraciones de las canciones del grupo ordenadas en forma cronológica. Este algoritmo retornará qué canciones deben integrar el CD, de forma de maximizar la duración.
b) Además de lo anterior, se dispone ahora de un vector e*xito* con la una ponderación para cada canción que indica su nivel de éxito. Replantee la formalización por Backtracking del problema de forma de maximizar el éxito.
c) ¿Con qué otro(s) método(s) es posible resolver el problema de la parte b)? Justifique.

EJERCICIO 6 (R)

Dos grafos $G = (V,E)$ y $H = (A,B)$ son isomorfos si existe una función $f : V \rightarrow A$ tal que $\forall (v,w) \in E \Rightarrow (f(v), f(w)) \in B$

Resuelva por Baktracking, formalizando y escribiendo un algoritmo que determine si dos grafos son isomorfos.

Para reducir el tamaño del árbol, considere que para que exista una correspondencia entre dos vértices, estos tienen que tener el mismo grado.

EJERCICIO 7 (R)

Resuelva por Baktracking, formalizando y escribiendo un algoritmo que dado un laberinto representado por una matriz de tamaño *N***M* de caracteres, encuentre el camino de salida del mismo. El algoritmo recibirá coordenadas de comienzo e imprimirá el camino hasta la salida.

Los mapas tienen 3 tipos de elementos:

'#' - Pared.
' ' - Camino.
'**S**' - Salida.

A continuación se muestra un ejemplo para *N*=7 y *M*=9:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	#	#	#	#	#	#	#	#	#
1	#								**S**
2	#		#		#	#	#	#	#
3	#		#	#	#				#
4	#		#				#		#
5	#				#	#	#		#
6	#	#	#	#	#	#	#	#	#

Partiendo del nodo (3,1) el camino de salida del laberinto es:

(3,1), (2,1), (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8).

EJERCICIO 8 (C)

Suponga que se tiene *n* hombres y *n* mujeres y se desea formar *n* parejas (entre hombres y mujeres) contemplando de la mejor manera posible las preferencias mutuas. Se tiene dos matrices *P* y *Q* de *n* x *n*, tal que *P*[*i*][*j*] es la preferencia del hombre *i* por la mujer *j* y *Q*[*i*][*j*] es la preferencia de la mujer *i* por el hombre *j*. Se quiere formar las *n* parejas de modo de maximizar la suma de los productos de las preferencias de cada integrante de una pareja por su compañero.

Resuelva por Baktracking, formalizando y escribiendo un algoritmo que resuelva el problema anterior.

EJERCICIO 9 (R)

Dado un grafo dirigido, sin lazos, con costos positivos en las aristas, que no tiene porqué ser conexo. Considerando sólo los ciclos del grafo que cumplen las siguientes propiedades:

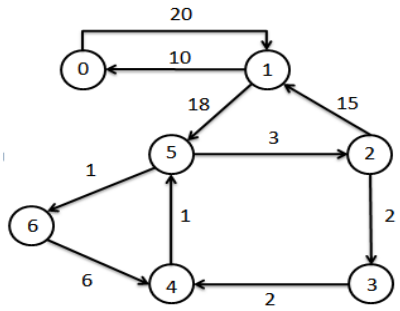
- cada arista se visita una única vez
- no se repiten vértices en el ciclo (salvo el primero y el último)

se define *ciclo mínimo* como aquel ciclo que tiene el menor costo de todos los ciclos del grafo.

El largo de un ciclo es la cantidad de aristas que lo componen.

Se **desea obtener** el ***ciclo mínimo* de largo máximo** del grafo utilizando *Backtracking*. Notar que puede existir más de un ciclo en las condiciones pedidas; en dicho caso se podrá devolver cualquiera (pero solo uno).

Los vértices se representan con enteros en el rango 0..N-1. A continuación se muestra un grafo con N = 7,



Los ciclos que cumplen las propiedades definidas anteriormente son:

- ciclo 1: [0, 1, 0,] con costo = 20 + 10 = 30
- ciclo 2: [2, 1, 5, 2] con costo = 15 + 18 + 3 = 36
- ciclo 3: [5, 2, 3, 4, 5] con costo = 3 + 2 + 2 + 1 = 8
- ciclo 4: [5, 6, 4, 5] con costo = 1 + 6 + 1 = 8

donde existen dos *ciclos mínimos* de costo 8; y el *ciclo mínimo* de largo máximo es el ciclo 3.

Se pide:

a) Formalizar el problema en términos de *Backtracking*. Indicar: forma de la solución, restricciones explícitas e implícitas, predicados de poda y función objetivo.

b) Implemente una función en C* que resuelve el problema utilizando *Backtracking* con el siguiente encabezado:

```
Lista CicloMinimo(Grafo g);
```

En caso de que no exista solución al problema, debe retornarse la lista vacía. En caso contrario, la lista debe contener los vértices del *ciclo mínimo* de largo máximo. En caso de existir más de un ciclo en las condiciones pedidas puede devolver cualquiera.

TADs auxiliares:

```
// TAD Grafo
   Grafo construirGrafo (int cantNodos);
   // Construye un grafo con 'cantNodos' nodos.
   void agregarAristaGrafo (Grafo& grafo, int nodoOrigen, int
   nodoDestino, int costo);
   // Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
   //  Se agrega una arista a 'grafo'.
   int costoAristaGrafo (Grafo & grafo, int nodoOrigen, int nodoDestino);
   // Precondicion: existe la arista [nodoOrigen, nodoDestino] en el
   grafo.
   //  Devuelve el costo de la arista [nodoOrigen, nodoDestino].
   bool existeAristaGrafo (Grafo grafo, int nodoOrigen, int nodoDestino);
   // Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
   // Retorna true sii existe una arista dirigida desde 'nodoOrigen'
   hacia
   // 'nodoDestino' en el grafo.
   int cantNodosGrafo (Grafo grafo);
   // Retorna la cantidad de nodos del grafo.
   Lista adyacentesNodoGrafo (Grafo grafo, int nodo);
   // Precondicion: 0 <= nodo < cantNodosGrafo (grafo)
   // Retorna la lista de nodos adyacentes a 'nodo' del grafo.
   void destruirGrafo (Grafo & grafo);
   // Libera la memoria asociada a la estructura.
```

```
// TAD Lista de Enteros
   Lista* ListaCrear();
   // Construye la Lista vacía.
   void ListaInsertar (Lista* l, int n);
   // Agrega el entero n a l.
   bool ListaEstaVacia (Lista* l);
   // Retorna true sii l está vacía.
   void ListaSacar (Lista* l);
   // Saca el primer elemento de l.
   // Precondicion: !ListaEstaVacia (l).
   int ListaPrimero (Lista* l);
   // Retorna el primer elemento de l.
   // Precondicion: !ListaEstaVacia (l).
   void ListaDestruir (Lista* l);
   // Libera la memoria de l.
   void ListaCopiar (Lista* l);
   // Construye una copia limpia de l.
```

Si desea utilizar otro *TAD* conocido debe declararlo como los anteriores y no implementarlo.

EJERCICIO 10 (C)

Una red de computadoras se encuentra distribuida entre varias ciudades. Todas las máquinas pueden comunicarse entre sí, ya sea directamente o bien a través de otras. El sistema ideal para prevenir fallas sería uno en el cual todos los nodos estuvieran unidos directamente con todos los demás. Como ese no es el caso real, en la práctica, uno de los índices que se utiliza para medir la 'confiabilidad' del sistema es el tamaño del mayor subconjunto de máquinas que cumplen ese requisito (el de estar unidas directamente todas con todas).

a) Plantee el problema en términos de grafos.
b) Construir un algoritmo para resolver el problema de determinar el índice de confiabilidad antes definido utilizando una matriz de adyacencias para representar el grafo.

EJERCICIO 11 (C)

Se tiene *n* personas a las cuales se les desea asignar *n* trabajos. El costo de asignar el hombre *i* al trabajo *j* es *Costo*[*i,j*].

Resuelva por Baktracking, formalizando y escribiendo un algoritmo que minimice el costo total de las asignaciones.

EJERCICIO 12 (C)

Sea un laberinto dado por una matriz *n* x *n* definida sobre (pared, libre, Minotauro), donde pared significa que no se puede avanzar en esa dirección. Nuestro héroe, Perseo debe encontrar en este laberinto al Minotauro para matarlo.

Perseo entra al laberinto por una esquina del mismo. Sólo se puede avanzar en 4 direcciones, nunca en diagonal. Se mata al Minotauro llegando hasta el lugar que este ocupa.

Resuelva por Baktracking, formalizando y escribiendo un algoritmo que encuentre todos los caminos para matar al Minotauro.

EJERCICIO 13 (R)

Se dispone de un tablero de p filas y q columnas. A su vez, se tienen p*q fichas. El juego consiste en ordenar las fichas en el tablero de manera de obtener el mayor puntaje posible.

Cada ficha es de un tipo y cada tipo de ficha tiene dos atributos: factor (número natural) y largo mínimo (número natural) en que se empieza a sumar puntaje de una línea. Puede haber tipos de fichas que tengan el mismo factor y largo mínimo siendo aún distintos tipos de fichas.

Si se forma una línea vertical, horizontal o diagonal con fichas del mismo tipo se incrementa el puntaje según el puntaje que tiene ese tipo de ficha multiplicado el largo de la fila.

Por ejemplo, si se tiene una línea de tres fichas del tipo con largo mínimo 2 esta no acumula puntaje debido a que no satisface el mínimo, en cambio si estas 3 fichas fueran de un tipo con factor 2 y largo mínimo 3, el puntaje se incrementaría en 6.

Se pide:

a. Formalizar la solución.
b. Realizar una implementación que cuente la cantidad de soluciones óptimas que existen.

```
typedef struct
{
        int factor;
        int largoMinimo;
} TipoFicha;

typedef struct
{
        int cantidad;
        TipoFicha tipoFicha;
} TipoFichaCantidad;

int cantidadOrdenamientos(int p, int q, int cantTipos, TipoFichaCantidad* tipos)
```