

## Tarea 3

Nombre: Juan Sebastian Vargas  
 Fecha: 4/05/16

### Punto 1

#### Floyd-Warshall

Para Solucionar este problema se utilizaria el mismo algoritmo descrito en 1, donde  $D^*$  es la matriz que contiene el costo minimo de los viajes.

```

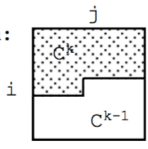
proc FW (var m:array[1..n,1..n] of R*)
{Pre:    m = D}
{Pos:    m = D*}
┌  k:= 0;
  {Inv P: 0≤k≤n ∧ m = Ck }
  do k≠n → i,j,k:= 1,1,k+1;
    {Inv P1: 1≤i≤n+1 ∧ 1≤j≤n ∧ m:
      
    }
  do i≠n+1
    → m[i,j]:= min {m[i,j], m[i,k]+m[k,j]} ;
    if j≠n → j:= j+1
    [] j=n → i,j:= i+1,1
    fi
  od
└
    
```

Figura 1: Floyd-Warshall ejemplo libro

si se necesitara encontrar y reconstruir ese camino, se haria el mismo algoritmo, solo que actualizando una matriz tal que:

Esta idea se plasma al mantener una matriz  $\text{intmd}$ , de manera que, para  $1 \leq i, j \leq n$ :

$\text{intmd}_{ij} = v$  , si existe una ruta que pasa por  $v \in V$ , con costo mínimo  
 $= 0$  , si la ruta más corta de  $i$  a  $j$  es el arco  $(i, j) \in E$   
 $= -1$  , si no hay camino de  $i$  a  $j$ .

Figura 2: Matriz para reconstruir el camino de costo minimo

Para la complejidad temporal podemos encontrarla facilmente, como vemos en el algoritmo variamos  $i, j$  y  $k$  hasta  $n$ , donde  $n$  es el numero total de Nodos, y  $T_{FW} = O(n^3)$ , para la complejidad Espacial tenemos guardamos una matriz de  $n \times n$  y es por esto que  $S_{FW} = O(n^2)$

### Dijkstra

el algoritmo que me encuentra la ruta optima de un nodo  $f$  hasta el resto de nodos es:

```

proc Dijkstra (G(V,E,c): grafo, f: V; var df:array[1..n] of R* )
{Pos:      df = Df. }
[ M:= {f};
  for v∈V      →   df[v]:= c[f,v] rof;
  {Inv P:  ⟨∀v: v∈M : df[v]=Df*⟩ ∧ ⟨∀v: v∈V\M : df[v]=CfM⟩ }
  do M≠V →   w:= Escoja_min(df,V\M);
             {Q1: P ∧ df[w]≤df[V\M] ∧ w∈V\M }
             M:= M∪{w};
             for v ∈ V\M
                 →   df[v]:= min{df[v],df[w]+c[w,v]}
             rof
  od
]
```

Figura 3: Algoritmo de Dijkstra

donde la complejidad temporal depende de dos partes, la inicializacion y las iteraciones, la inicializacion es el primer for que llena el df, y las iteraciones son los ciclos que siguen.

$$T_D = T(Ini) + T(ciclos) = O(n) + O(n)[T(Escoja_{min}) + T(eliminar)] + O(e) \quad (1)$$

Donde  $T(Escoja_{min}) = O(n)$ , ya que existen  $n$  diferentes nodos, teniendo todo en cuenta se obtiene que

$$T_D = O(n)[O(n)] + O(e) = O(n^2 + e) \quad (2)$$

pero esta seria la complejidad temporal para encontrar el camino minimo de un nodo dado  $f$  a el resto de los nodos del grafo, para obtener la totalidad de caminos minimos de todos los nodos hacia todos los nodos, debemos hacer este proceso para todos los nodos, en este casi la complejidad seria entonces:

$$T_{TD} = O(n(n^2 + e)) = O(n^3) \quad (3)$$

donde se asumio que  $e \ll n^3$ , y para la espacial se guardara  $n$  vectores de la manera  $di : i \in 1..f..n$ , y de esta manera  $S_D = O(n^2)$

## Punto 2

### 2.a

[para evitar errores se tiene que las vasijas seran  $va_i$ ] Para modelar este problema de esta manera tenemos un grafo  $G(V, E)$ , con  $V := \{ \langle v_1, \dots, v_n \rangle \mid 0 \leq v_1 \leq c_1, \dots, 0 \leq v_n \leq c_n \}$  y debemos definir cuando se llega a un estado de solucion, y esto pasa cuando:

$$v_{final} \in V \mid v_k = P, k \in 1..n \quad (4)$$

y los arcos se definen como los cambios que se hacen en los estados de las vasijas, asi:

$$\langle v_1, \dots, v_n \rangle \rightarrow_{accion} \langle v'_1, \dots, v'_n \rangle \quad (5)$$

y accion sera el cambio que se hagan en las vasijas, y como las acciones que se pueden realizar son llenado y trasvasado tenemos que:

$$\langle v_1, \dots, v_n \rangle \rightarrow_{Lk} \langle v_1, \dots, c_k, \dots, v_n \rangle \quad (6)$$

$$\langle v_1, \dots, v_n \rangle \rightarrow_{Tki} \begin{cases} \langle v_1, \dots, 0, \dots, v_i + v_k, \dots, v_n \rangle & si \quad v_i + v_k \leq c_i \\ \langle v_1, \dots, 0, \dots, c_i, \dots, v_n \rangle & si \quad v_i + v_k \geq c_i \end{cases} \quad (7)$$

### 2.b

Haremos una busqueda en el grafo, y se buscara  $v_{final} \in V$  una solucion que cumpla con la condicion anterior, apartir de un nodo  $v_{inicial} \in V$ .

## Floyd-Marshall

## Punto 3

Para este punto se deben notar un par de cosas interesantes, si  $A = \{a_1, \dots, a_n\}$  es el conjunto de actividades, si escogemos un  $a_i$  arbitrario, conocemos  $B_i$ , como  $a_i \notin B_i$ , ya que  $B_i$  son las actividades que deben hacerse antes de  $a_i$ . lo anterior nos lleva a decir que sea  $a_k$  la ultima actividad a realizar,  $|B_k| < n$ , y una afirmacion mas fuerte aun pero igual de valida,  $|B_k| = n - 1$ ,  $B_k + \{a_k\} = A$ . Para

saber si el proyecto se puede hacer, se debe poder ordenar  $A = \{a_k\} + \cup_{i=1}^n B_i$ , con  $|B_{i+1}| - |B_i| = 1$ . y es por esto que el problema se reduce a ordenar A y despues validar si estan las n actividades. en el programa se usara la funcion  $C(a_i) = B_i$ , que me retorna el conjunto de actividades previas a  $a_i$ . se ordenara una lista con la funcion  $Orden(A)$  que me entrega  $A'$  ordenado mediante seleccion, comparando el valor de  $|C(a_i)|$ , este ordenamiento tiene la complejidad normal para este algoritmo de ordenamiento  $T = O(n^2)$  veamos como nos queda  $A'$

$$Orden(A) = A' = [a'_1..a'_n], \text{ donde se cumple que: } |C(a'_1)| < .. < |C(a'_n)| \quad (8)$$

El algoritmo es:

```

proc actividad (A=[a1..an])
A':=Orden(A);
po,i:=true,1;
do i < n → x,y := |C(A'[i]),|C(A'[i+1])|;
                IF y - x ≠ 1 → po = false FI
od

```

{Pos: po=true  $\vee$  po=false } Para la complejidad del algoritmo es la siguiente:

$$T_3 = T(Orden) + T(ciclo) = O(n^2) + O(n) = O(n^2) \quad (9)$$