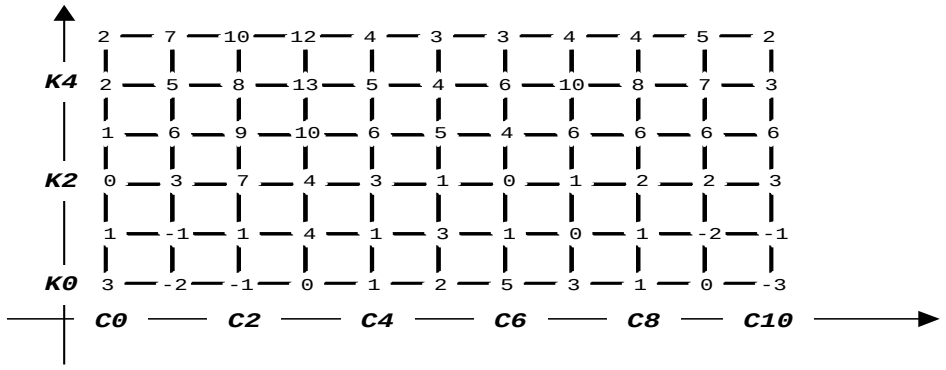
ISIS 2103 Diseño de algoritmos **Nombre:** _____
Semestre 2014-10 Parcial 3
Mayo 8, 2014
Prof. Rodrigo Cardoso

N.B: Cuando se soliciten algoritmos se esperan soluciones eficientes en tiempo y en espacio. Por tanto, soluciones menos eficientes que lo esperado pueden ser penalizadas. Si se pide el diseño de un algoritmo se pueden utilizar referencias a algoritmos conocidos, así como a sus costos computacionales.

1 [40/100] *Los caminos menos difíciles*

Una ciudad cuadriculada tiene una red de ciclovías en la que se conoce la altura de cada intersección. Por ejemplo:



La nomenclatura de la ciudad permite llamar *calles* a las ciclovías verticales y *carreras* a las horizontales. Tanto calles como carreras se numeran `0`, `1`, `2`, …, empezando en el extremo inferior izquierdo de la cuadrícula. Las esquinas se denotan con una pareja `(c,k)` donde c es el número de la calle y k es el número de la carrera que se intersectan en esa esquina. Para cada esquina se conoce, además, la *altura* a la que se encuentra con un número entero.

La *dificultad de una cuadra*, i.e., un trayecto entre dos esquinas adyacentes, es la diferencia de alturas entre la esquina final y la esquina inicial, si ésta diferencia es positiva; en otro caso, es `0`. En la figura, la dificultad de ir de `(3,3)` a `(3,4)` es 3. Pero la dificultad de ir de `(3,4)`a `(3,3)` es `0`.

La *dificultad de un camino* entre dos esquinas es la suma de las dificultades de las cuadras que constituyen el camino. Así, el camino ⟨`(0,0),(1,0),(2,0),(2,1),(3,1)`⟩ tiene dificultad 6.

Para los siguientes problemas supóngase dada una matriz `a[0..m-1,0..n-1]:int` que corresponde a la distribución de alturas de las esquinas de la ciudad. Describa algoritmos para resolver los problemas y estime sus complejidades espacial y temporal (operación básica: asignación).

1a (20/40) *Camino menos difícil de una esquina a otra*
Para un par de esquinas `(a,b)` y `(c,d)`, determinar un camino de dificultad mínima para ir desde `(a,b)` hasta `(c,d)`.

Se propone usar un algoritmo de Dijkstra sobre el grafo `G(V,E,c)` definiendo
V : esquinas de la ciudad.
E : dos arcos por cada cuadra (uno en cada sentido)
c : dificultad de la cuadra, en el sentido que se recorra.

La búsqueda empieza en a. Se define una matriz d[0..m-1,0..n-1] que guarda las dificultades de a a los demás nodos. Debe inicializarse con las esquinas vecinas de a (máximo 4) y lo demás en ∞. Es claro que se usan valores en el semianillo $(R^*, min.+, \infty, 0)$.

Las distancias (dificultades) desde un nodo escogido para marcar se calculan según la definición de la dificultad de las cuadras, al marcar el nodo, en el ciclo de relajación.

[10/10]

Para estimar los costos:
$|V| = m*n$
$|E| = 2*(m*(n-1)+n*(m-1)) = 4*m*n-4m-4n = \theta(|V|)$.

Claramente, se tiene que $|E| = O(|V| \log |V|)$. De este modo, la solución Dijkstra es buena y tiene los siguientes costos:
$T_a(m,n) = \theta(m*n \log m*n)$
$S_a(m,n) = \theta(m*n)$

[10/10]

1b (10/40) *Cota superior para dificultad mínima entre dos esquinas cualesquiera*
Determinar un entero que sea una cota superior para la dificultad de un camino de dificultad mínima entre cualquier pareja de esquinas de la ciudad.

Un grafo G(V,E) como en 1a.

Una solución evidente consiste en sumar las dificultades de todas las cuadras, porque un camino de dificultad máxima debería usar, a lo sumo, todos los arcos.

[5/5]

Este algoritmo toma |E| pasos, de modo que se puede resolver en
$T_b(m,n) = \theta(m*n)$
$S_b(m,n) = \theta(1)$.

[5/5]

1c (10/40) *Máxima dificultad mínima entre dos esquinas cualesquiera*
Determinar la máxima dificultad mínima entre cualquier pareja de esquinas de la ciudad.

La cota alcanzada en una solución para 1b puede ser demasiado grande. Por eso hay que asegurar la corrección con otro tipo de solución.

Un grafo G(V,E) como en 1a.

El algoritmo de 1a se puede repetir para cada posible pareja de esquinas. El espacio de distancias desde el nodo de origen se puede reutilizar y el máximo alcanzado se puede guardar en una variable que se actualiza a medida que el algoritmo se ejecuta.

[5/5]

Para estimar los costos, debe observarse que hay m^2*n^2 posibles parejas de esquinas. Entonces
$T_c(m,n) = \theta(m^3*n^3 \log m*n)$
$S_c(m,n) = \theta(m*n)$.
(Si se quiere pensar, sin pérdida de generalidad, que $m \geq n$, se tendrá que:
$T_c(m,n) = \theta(m^6 \log m)$, $S_c(m,n) = \theta(m^2)$)

[5/5]

Variante FW

Un grafo G(V,E) como en 1a.

Usar un algoritmo de Floyd-Warshall más una variable que calcula la distancia máxima que se lleva hasta el momento, entre los mínimos calculados (en el ciclo de relajación). El costo de actualizar esta variable es O(1).

[10/5]

Así, se tendrán los costos:
$T'_c(m,n) = \theta(m^3*n^3)$
$S'_c(m,n) = \theta(m*n)$ // se usa para la matriz inicial y para su modificación.
(Si se quiere pensar, sin pérdida de generalidad, que $m \geq n$, se tendrá que:
$T_c'(m,n) = \theta(m^6)$, $S_c'(m,n) = \theta(m^2)$)

[5/5]

2 [30/100] *Mapa 2-coloreable*

Para un grafo dirigido G(V,E) y un k:**nat**, una k-*coloración* es una función c: V → 1..k. Se dice que G es k-*coloreable* si existe una k-coloración c tal que c.u ≠ c.v para (u,v)∈E.

2a (10/30) Muestre que un árbol T(V,E), con raíz r∈V, es 2-coloreable.

Para definir una coloración c, sea u.d la distancia de un nodo u a la raíz. Se puede definir
c.u = 1, si u.d es par;
= 0, si u.d es impar.

Cada nodo tiene un camino único a la raíz; no puede haber contradicciones en los colores de elementos vecinos.

[10/10]

2b (20/30) Sea G(V,E) un grafo y sea r∈V tal que todo nodo es alcanzable desde r. Describa un algoritmo para decidir si un grafo G(V,E) es 2-coloreable (no es necesario escribir el algoritmo detalladamente). Estime las complejidades temporal y espacial de su solución.

Para decidir si G es 2-coloreable, añadir un atributo x.c:{1,2} a cada nodo c∈V.

Se puede plantear un recorrido DFS desde r. Al empezar se define r.c:= 1, pero x.c no se define para los demás nodos . Supóngase que se procesa un arco (u,x), i.e., u se descubrió y se están procesando sus sucesores. Si x es recién descubierto (B en DFS) , se define x.c del color diferente al de u. Si x ya está descubierto, se comprueba que su color sea diferente del de u (si no es así, el grafo no es 2-coloreable y el algoritmo puede parar indicando que no lo es).

Nótese que, puesto que todo nodo es alcanzable desde r, si el recorrido DFS visita todo el grafo éste debe ser 2-coloreable.

[15/15]

Variante: BFS

La misma argumentación que se usó para DFS.

[15/15]

N.B. No se ha usado en la argumentación, pero el árbol DFS generado debe ser 2-coloreable si el algoritmo termina. El chequeo de que el color de un nodo ya visitado sea diferente del de un nodo que lo redescubra es una comprobación de que el arco que se procesa es consistente con la coloración. Así, el resultado de 2a, más el hecho de que BFS/DFS produce un árbol generador (que, como árbol, es coloreable), más controlar que no hay contradicciones con nodos no de árbol, es un algoritmo correcto.

Complejidades

Son las mismas de DFS (o BFS), puesto que solo se añade un costo de $O(1)$ para almacenar un atributo $x.c$ a cada nodo x (esto es $O(|V|)$ y su mantenimiento es $O(1)$)

$T(G,r) = O(|V|+|E|)$
$S(G,r) = O(|V|)$

[5/5]

Variante: BFS

$T'(G,r) = O(|V|+|E|)$
$S'(G,r) = O(|V|)$

[5/5]

3 [30/100] *Sumas de un conjunto*

Suponga un conjunto S de n números enteros positivos, y un entero a positivo. Se quiere saber si es posible definir un conjunto $A \subseteq S$, tal que sus elementos sumen a.

3a (10/30) Exprese el problema como una búsqueda en grafos, i.e., defina SOLPOS, sat, SOL, BUSQ, s, $\rightarrow$.

Los elementos de S se pueden representar en una lista, sin repeticiones: $S = \langle s_0,\ldots,s_{n-1}\rangle$.

$SOLPOS = \{\langle x_0,x_1,\ldots,x_{n-1}\rangle | \ x_j \in \{0,1\},\ 0 \le j < n\}$

$sat(x_0,x_1,\ldots,x_{n-1}) \equiv \ (+j |\ 0 \le j < n : x_j * s_j) = a$

$SOL = \{x:SOLPOS |\ sat.x\}$

$BUSQ = \{\langle x_0,x_1,\ldots,x_{k-1}\rangle |\ 0 \le x_j \le 1,\ 0 \le j \le k < n\}$

$s = \langle\rangle$

$\langle x_0,x_1,\ldots,x_{k-1}\rangle \rightarrow (x_0,x_1,\ldots,x_{k-1},x_k) \ \equiv \ (+j |\ 0 \le j \le k : x_j * b_j) \le a$

[10/10]

Variante:

$\langle x_0,x_1,\ldots,x_{k-1}\rangle \rightarrow (x_0,x_1,\ldots,x_{k-1},x_k)$

Defecto: define arcos que no servirán en una posible solución y que se pueden omitir desde el principio.

[7/10]

3b (10/30) Justifique si

(i) hay que marcar nodos;

No: el grafo es un árbol. No hay posibilidad de armar ciclos, porque los sucesores siempre tienen un elemento más.

[4/4]

(ii) hay que verificar que la agenda se vacíe;

Sí. Puede no haber soluciones.

[3/3]

(iii) el algoritmo puede no terminar.

No. El algoritmo termina, porque BUSQ y $\rightarrow$ son finitos.

[3/3]

3c (10/30) Calcule $|BUSQ|$ y $|\rightarrow|$, como estimación de la complejidad temporal de su algoritmo.

Obsérvese que los elementos de BUSQ se pueden agrupar en niveles, de acuerdo al tamaño de las secuencias. Los elementos de nivel k se pueden representar con los números binarios de k bits y éstos son 2^k.

Para calcular $|\rightarrow|$, cada elemento en BUSQ tiene exactamente un predecesor (su padre). Es decir, hay tantos arcos como nodos, menos 1 (la raíz):

$|BUSQ| = (+k| \ 0 \leq k < n : 2^k) = 2^n - 1 = O(2^n)$
$|\rightarrow| = 2^n - 2 = O(2^n)$

Un algoritmo de agenda tiene costo temporal $O(|BUSQ| + |\rightarrow|)$, i.e. $O(2^n)$.

[10/10]