DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Segundo parcial

23 de diciembre de 2015 — Algoritmos y Estructuras de Datos Avanzadas

Alumno	LU	Correo electrónico
Vileriño, Silvio	106/12	svilerino@gmail.com

Docente	Nota

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

ALGORITMOS Y ESTRUCTURAS DE DATOS AVANZADAS
Take Home / 18-NOV-2015

Fecha de entrega: 4-DIC-2015.

1. Dado un grafo $G = (V = \{v_1, \ldots, v_n\}, E)$, la matriz de distancias de G es una matriz D de $n \times n$ donde d_{ij} es la longitud (cantidad de aristas) de camino más corto entre v_i y v_j y la matriz de sucesores de G es una matriz S también de $n \times n$ donde $v_{s_{ij}}$ es el primer vertice después de v_i de algun camino mínimo entre v_i y v_j recorriendo desde v_i.

   (a) Justificar por qué conviene tener la matriz S en lugar de calcular explícitamente un camino mínimo para cada par de vertices de G dados G y D.

   (b) Dar algoritmos eficientes para computar S dados G y D.

   (c) En el caso que el grado máximo de los vertices de G está acotado por un valor constante k. ¿Qué mejora se podría hacer?

   (d) En el caso que el diámetro de G (la distancia máxima entre pares de vertices de G) está acotado por un valor constante k. ¿Qué mejora se podría hacer?

2. Dar un algoritmo exacto para listar todos los bicliques maximales (cliques bipartitos maximales) de un grafo bipartito. Un subconjunto de vertices B es un biclique de un grafo bipartito $G = (V_1 \cup V_2, E)$ si $B \cap V_1 \neq \emptyset$, $B \cap V_2 \neq \emptyset$ y cada vértice de $B \cap V_1$ es adyacente a todos los vértices de $B \cap V_2$. Mostrar la correctitud y determinar la complejidad del algoritmo propuesto. El algoritmo debe tener time delay polinomial (el tiempo que demora en encontrar la primera solución y el tiempo máximo entre dos soluciones consecutivas) y el espacio requerido también sea polinomial.

3. Da un algoritmo exacto lo más eficiente posible para determinar un clique transversal de menor cardinalidad en un grafo sin diamantes. Mostrar la correctitud y determinar la complejidad del algoritmo propuesto.

1. Ejercicio 1

1.1. Item a)

Desde el punto de vista de la complejidad espacial, si almacenamos todos los caminos mínimos, esto tiene un costo espacial de $\mathcal{O}(n^3)$ y el costo de devolver un camino es de tiempo constante $\mathcal{O}(1)$. Por otro lado, utilizando la matriz de sucesores S, podemos devolver un camino mínimo entre cualquier par de nodos con complejidad temporal $\mathcal{O}(n)$ y espacial $\mathcal{O}(n^2)$. El hecho de tener la matriz S constituye una mejora sustancial en el espacio asintótico consumido, por lo tanto resulta mas conveniente. A continuación presentamos el pseudocódigo de un algoritmo para calcular camino mínimo entre un par de nodos i, j dada la matriz S:

Algoritmo 1 Camino mínimo entre 2 nodos utilizando la matriz de sucesores S

Entrada: Matriz S, nodos i, j
Salida: Camino mínimo entre los nodos i, j

```
1: path = ∅
2: mientras i ≠ j hacer
3:     path.append(i)
4:     i = S[i][j]
5: fin mientras
6: path.append(i) //Ultimo nodo del camino
7: devolver path
```

1.2. Item b)

Para el cálculo de la matriz de sucesores S en base a G y D, proponemos dos enfoques:
El primer enfoque consiste en utilizar el algoritmo visto en clase, que utiliza un algoritmo randomizado para calcular las matrices de testigos. La matriz S se calcula a partir de las matrices de testigos de tres productos matriciales. El costo de calcular S de esta manera, es, en promedio, $\mathcal{O}(M(n) * log(n)^2)$, donde $M(n)$ corresponde al costo temporal de la multiplicacion matricial, actualmente es $\mathcal{O}(n^{2,376})$.

El segundo enfoque, se basa en la idea de iterar sobre cada par de vertices (i,j) y computar $S_{i,j}$. El cómputo de $S_{i,j}$ es como sigue: miramos cada vecino k del nodo i, si vale $D_{i,j} = D_{i,j} - 1$ entonces actualizamos al nodo k como el sucesor en el camino mínimo entre los nodos i, j. Con esta idea en mente, se presenta el siguiente pseudocódigo:

Algoritmo 2 Construcción de la matriz de sucesores S

Entrada: Grafo G, matriz de distancias D
Salida: Matriz de sucesores de caminos mínimos S

```
1: para i = 1 hasta n hacer
2:     Vec_i = Vecinos del nodo i;
3: fin para
4: para i = 1 hasta n hacer
5:     para j = 1 hasta n hacer
6:         para todo k ∈ Vec_i  hacer
7:             si D_{k,j} = D_{i,j} − 1 entonces
8:                 S_{i,j} = k;
9:                 break;
10:            fin si
11:        fin para
12:    fin para
13: fin para
```

Calculemos la complejidad del algoritmo 2 propuesto:

- $\mathcal{O}(n^2)$ de la inicialización de la matriz S
- $\mathcal{O}(n^2)$ de la inicialización de las vecindades de los nodos.
- El costo de los ciclos anidados es $\sum_{i=1}^{n}\sum_{j=1}^{n} Vec_i = n * \sum_{i=1}^{n} Vec_i = n * m$

Constituyendo un costo asintotico temporal total de $\mathcal{O}(n^2 + n * m)$.

1.3. Item c)

Dada esta precondición, resulta que al ser el grado constante, entonces m = O(n). Si utilizamos el algoritmo 2, ahora su costo temporal asintótico pasa a ser $\mathcal{O}(n^2)$.

1.4. Item d)

En este caso, tenemos que el diámetro es una constante k. Esto implica que todos los caminos mínimos pasan a tener longuitud constante. Si volvemos al tradeoff presentado en el item a), bajo esta precondición, almacenar los caminos mínimos precalculados, pasa a costar $\mathcal{O}(n^2)$ espacialmente, con lo cual utilizar la matriz S, carece de sentido.

2. Ejercicio 2

2.1. Correctitud

Proposición 1. *Sean $G = (V = V_1 \cup V_2, E)$ un grafo bipartito, $C \subseteq V$ un subconjunto de vertices tal que $C \neq V_1 \wedge C \neq V_2$ y $G' = (V = V_1 \cup V_2, E')$ con aristas adicionales tal que V_1 y V_2 sean cliques. Entonces el subgrafo inducido por C es clique maximal en G' si y solo si el subgrafo inducido por C es biclique maximal en G.*

Demostración. $\Rightarrow$)
Sea $C \neq V_1 \wedge C \neq V_2$ clique maximal de G'. Supongamos que C no es biclique de G. [1] Entonces existe al menos un nodo $v \in C \cap V_1(G)$ [2] que no es adyacente a al menos un nodo de $C \cap V_2(G)$[3]. Pero observemos entonces, que si volvemos a agregar las aristas que clausuran V_1 y V_2 conformando G', entonces C no es una clique de G', lo cual contradice la hipótesis, y es absurdo.

Ahora supongamos que la biclique C no es maximal, luego existe un nodo $w \notin C$ tal que es adyacente a todos los nodos $v \in C$ de forma tal que además estos nodos v pertenezcan al conjunto independiente contrario al cual pertenece w. Dicho esto, tenemos que w tiene intersección en G' con todos los nodos pertenecientes a su partición[4] pues agregamos esos ejes al clausurar G. Por otro lado, interseca con todos los nodos pertenecientes a C tal que estan en la particion contraria. Esto implica que entonces C no es maximal en G', lo cual es absurdo por hipótesis. □

Demostración. $\Leftarrow$)
Sea $C \neq V_1 \wedge C \neq V_2$ biclique maximal en G. Supongamos que C no es clique en G'[5]. Entonces debe faltar una arista e que tiene un extremo en cada conjunto $V_1(G')$ y $V_2(G')$ porque $V_1(G')$ y $V_2(G')$ son cliques por construcción. Con lo cual tampoco C es biclique en G, contradiciendo la hipótesis, absurdo.

[1] Recordemos que G es el grafo G' sin las aristas adicionales para forzar que V_1 y V_2 sean cliques en G'.
[2] Análogamente $v \in C \cap V_2(G)$
[3] Análogamente $C \cap V_1(G)$
[4] Conjunto independiente V1 o V2
[5] Recordemos que G' es el grafo G con las aristas adicionales para forzar que V_1 y V_2 sean cliques.

Ahora supongamos que C no es clique maximal en G', luego existe un nodo $j \notin C$ que es adyacente a todo otro vértice de C en G'. Podemos entonces decir que $j \notin C$ en G y es adyacente a todos los vertices del conjunto independiente contrario en G, luego C no es biclique maximal en G, lo cual contradice la hipotesis. □

Con estos dos resultados, planteamos la correctitud de la equivalencia entre buscar cliques maximales sobre G' y buscar bicliques maximales sobre G. Mostrando que nuestro algoritmo es correcto, mas aun, al devolver el resultado, se verifica(en tiempo polinomial) que no se estan devolviendo las cliques ficticias agregadas a G'.

2.2. Algoritmo propuesto

Proposición 2 (Visto en clase - Tsukiyama)**.** *Sea $G = (V, E)$ un grafo, existe un algoritmo para listar todos los cliques maximales con las siguientes propiedades:*

- *Complejidad espacial:* $\mathcal{O}(n + m)$
- *Time delay:* $\mathcal{O}(n * m)$

Sea $G = (V = V_1 \cup V_2, E)$ un grafo bipartito, existe un algoritmo para listar todas los bicliques maximales de G utilizando el algoritmo mencionado en la proposición anterior. La idea del mismo es como sigue:

1. Entrada: $G = (V = V_1 \cup V_2, E)$ un grafo bipartito.[6]
2. Precomputar los conjuntos independientes de nodos V_1 y V_2 de G.
3. Construir $G' = (V = V_1 \cup V_2, E')$ adicionando aristas a los conjuntos independientes V_1 y V_2 para que sean cliques.
4. Correr el algoritmo para listar cliques maximales sobre G'.
5. Revisar el primer resultado del algoritmo de Tsukiyama
   a) Si este primer output tiene interseccion vacía respecto de $V_1(G')$ o $V_2(G')$, ignorar el resultado y continuar con el algoritmo.
   b) Caso contrario, devolverlo como un resultado valido.
6. Para cada resultado T obtenido con el algoritmo luego de la primera, considerar los siguientes dos casos:
   a) Si T intersecta con $V_1(G')$ y con $V_2(G')$ de forma no vacia, considerarlo un resultado valido y continuar con el algoritmo.
   b) Caso contrario, terminar el algoritmo.

Dado el ordenamiento de nodos que proponemos para la entrada G, resulta que si V_1 o V_2 son cliques maximales, estas serán devueltas al principio y al final del algoritmo respectivamente. Esto indica que nuestros chequeos al principio de descarte del primer resultado y nuestra condicion de corte del algoritmo, son medidas licitas para garantizar correctitud. Por otro lado, dado que en nuestro algoritmo propuesto devolvemos conjuntos que cumplen las hipótesis de los resultados demostrados en la seccion anterior, y hemos establecido una equivalencia entre cliques y bicliques maximales de G y G'. El algoritmo que se propuso se puede considerar correcto.

[6] Sin perdida de generalidad consideramos este grafo tal que si numeramos los nodos, los que pertenezcan al conjunto independiente V_1 preceden a los que pertenezcan a V_2

2.3. Complejidad

La operacion de clasificacion del grafo en particiones V_1 y V_2 puede ser resuelta mediante un algoritmo de exploracion[7] de grafos y utilizando una estructura de consulta dinamica[8], esto es claramente polinomial. Por otro lado, la clausura del grafo G' puede realizarse en tiempo polinomial.

Respecto a la complejidad espacial, notemos que agregar aristas al grafo G para construir G' consume espacio polinomialmente. Por otro lado, el algoritmo propuesto, consume espacio polinomialmente, con lo cual el espacio total ocupado es polinomial.

Respecto al time delay polinomial, clausurar V1 y V2 para el preprocesamiento es temporalmente polinomial y el algoritmo presentado tiene time delay polinomial, entonces si el preprocesamiento es polinomial y correr el algoritmo sobre G' es polinomial, el tiempo de espera hasta el primer resultado es polinomial. Por otro lado, el tiempo de espera entre resultados no cambia, pues es el time delay intrínseco del algoritmo que se vió en clase y revisar que las cliques devueltas no correspondan a las cliques espurias generadas en G' sobre V_1 y V_2 tambien consume tiempo polinomial.

3. Ejercicio 3

3.1. Resultados auxiliares y algoritmo propuesto

Comenzaremos con algunos resultados teóricos y luego propondremos un algoritmo para la resolución de este problema.

Proposición 3. *Sea G diamond-free entonces toda arista pertenece a lo sumo a una clique.* ***Esta propiedad fue vista en clase.***

Proposición 4. *Sea G diamond-free, sin nodos aislados, entonces a lo sumo tiene m cliques.*

Demostración. Pensemos combinatoriamente el problema, establezcamos una analogía entre cliques como cajas y aristas como bolitas.
Como por hipótesis no existen nodos aislados, luego toda clique tiene al menos una arista, o análogamente, toda caja tiene al menos una bolita.
Por la proposición anterior vista en clase, cada arista está en una sola clique, o análogamente, cada bolita puede solo ir a parar a una sola caja.
Finalmente, concluimos que si toda caja tiene alguna bolita, entonces tiene que haber mas bolitas que cajas. Análogamente, en términos del problema, se tiene que #(cliques de G) = #(cajas) $\leq$ #(bolitas) = m. □

Proposición 5. *Sea G diamond-free, entonces tiene a lo sumo n + m cliques.*

Demostración. Consideremos al grafo G tal que tenga n nodos aislados, que son cliques. Luego sea G' = G $\setminus$ { nodos aislados }. Por la proposición anterior, G' tiene a lo sumo m cliques. Totalizando que G tiene a lo sumo n+m cliques. □

Con estos resultados se propone la siguiente idea:

- Se calculan todas las cliques de G (**Con el algoritmo visto en clase**).
- Se prueban todos los posibles subconjuntos de nodos posibles, en particular, todos los conjuntos clique transversales).
- Se filtra el subconjunto de cardinalidad mínima que tenga interseccion no vacía respecto de todas las cliques.

[7] Por ejemplo DFS o BFS
[8] Por ejemplo una tabla hash o un arreglo

A continuación se presenta el pseudocódigo que ilustra esta idea:

Algoritmo 3 Clique transversal de cardinal mínimo en grafo sin diamantes

Entrada: Grafo G
Salida: W clique transversal de cardinal mínimo

```
1: Sean {C_i}_{i∈ℕ} = { Cliques de G } // Calcular con algoritmo visto en clase
2: Sea k = #{C_i} // Cantidad de cliques de G
3: para i = 1 hasta n hacer
4:    para todo W ⊆ V(G) tal que #W = i  hacer
5:       si W ∩ C_k ≠ ∅ ∀ C_1, ..., C_k entonces
6:          devolver  W;
7:       fin si
8:    fin para
9: fin para
```

3.2. Correctitud

Este algoritmo realiza lo que se mencionó en la idea mas arriba y es correcto, pues se prueban todos los posibles conjuntos clique transversal, de menor a mayor en cardinalidad, al encontrar el primero y devolverlo finalizando el algoritmo, esto asegura que sea uno de cardinal mínimo.

3.3. Complejidad

Analicemos con un poco de cuidado, la complejidad total del algoritmo 3.

- El cálculo de las cliques del grafo G tiene un costo temporal de $\mathcal{O}(\alpha(G) * m + n^2)$ donde $\alpha(G)$ es la arboricidad del grafo G. **Algoritmo visto en clase**.

- Por otro lado, el costo temporal de los ciclos anidados es $\sum_i \binom{n}{i} * (n+m) * n$, pues para cada i, existen $\binom{n}{i}$ subconjuntos de i nodos, por otro lado, para cada subconjunto W, hay que corroborar que la interseccion no sea nula para todas las cliques $C_1, \ldots, C_k$ del grafo G. Por la proposición vista anteriormente, k = O(n + m) y como cada clique tiene O(n) elementos, ver si la interseccion es no nula contra cada una ellas cuesta O(n). Esto se repite $\beta(G)$ veces, donde $\beta(G)$ indica el cardinal del clique transversal mínimo del grafo G. Más formalmente se tiene que el costo del algoritmo es:

  - $\mathcal{O}(\sum_{i=1}^{\beta(G)} \binom{n}{i} * (n+m) * n)$
  - $= \mathcal{O}((n+m) * n * \sum_{i=1}^{\beta(G)} \binom{n}{i})$
  - $= \mathcal{O}((n+m) * n * \sum_{i=1}^{\beta(G)} \frac{n^i}{i!})$
  - $= \mathcal{O}((n+m) * n * n^{\beta(G)})$

  A este último resultado se le debe sumar el $\mathcal{O}(\alpha(G) * m + n^2)$ del calculo de las cliques de G. Entonces, El costo temporal total es:

  $\mathcal{O}(\alpha(G) * m + n^2 + (n+m) * n * n^{\beta(G)})$, pero como estamos trabajando con complejidad asintótica y vale que $\alpha(G) \leq m$, **finalmente resulta** $\mathcal{O}((n+m) * n * n^{\beta(G)})$.