FACULTAD DE CIENCIAS EXACTAS INGENIERÍA Y AGRIMENSURA
Escuela de Posgrado y Educación Continua

Tópicos Avanzados de Optimización Combinatoria y Teoría de Grafos

Docentes: Graciela Nasini, Daniel Severin, Paola Tolomei, Pablo Torres

Práctica nº 3: Complejidad de Algoritmos

A los efectos de la resolución de los ejercicios de esta práctica consideramos las siguientes operaciones como elementales:

- Evaluar una expresión aritmética o booleana.
- Inicializar/actualizar una variable.
- Leer/modificar una componente de un vector, matríz o lista.
- Crear un conjunto vacío.
- Obtener un elemento de un conjunto no vacío.
- Agregar/quitar/chequear pertenencia de un elemento de un conjunto.
- Obtener el cardinal de un conjunto.

También, al referirnos a reducciones de problemas consideramos la siguiente definición. Dados los problema de optimización Π_1 y Π_2, decimos que Π_1 se *reduce* a Π_2 si existen algoritmos f y h tal que: 1) el algoritmo f lee una instancia de Π_1 y genera una instancia de Π_2, 2) el algoritmo h lee una solución del problema Π_2 y genera una solución del problema Π_1 y 3) si g es un algoritmo que resuelve Π_2 entonces $h \circ g \circ f$ es un algoritmo que resuelve Π_1. El siguiente diagrama explica el proceso de reducción:

$$\begin{array}{ccc} Inst.\ \Pi_1 & \xrightarrow{h\circ g\circ f} & Sol.\ \Pi_1 \\ \downarrow f & & \uparrow h \\ Inst.\ \Pi_2 & \xrightarrow{g} & Sol.\ \Pi_2 \end{array}$$

La complejidad (temporal) de la reducción se mide sumando la cantidad de operaciones elementales de f y h.

1. Una forma habitual de representar un grafo en una computadora es mediante los conjuntos con las vecindades de cada vértice. Es decir, representamos G con $V(G) = \{1, \ldots, n\}$ mediante una **lista** de **conjuntos** de vecindades $N(1)$, ..., $N(n)$. Determine el orden de complejidad de las siguientes operaciones en términos de n.

   a) obtener una arista de G, o contestar que G no tiene aristas,

   b) determinar el vértice de mayor grado de G,

   c) dado un conjunto $Q \subset V$, decidir si Q es una clique,

d) dado un conjunto $B \subset V$ y $k \in \mathbb{Z}_+$, decidir si B es un k-empaquetamiento de G,

e) dado un conjunto $M \subset V \times V$, decidir si M es un matching de G.

2. Determine el número de operaciones elementales que realiza el *Algoritmo del Vecino más Cercano* para el *Problema del Viajante de Comercio Euclidiano* [LCO, pág. 1] sobre una instancia de n puntos en el plano, los cuales se ingresan en dos vectores x e y, conteniendo las coordenadas de los puntos.

3. Suponga que tiene la posibilidad de utilizar 24 horas de una computadora que realiza una operación elemental en un nanosegundo (10^{-9} segundos). Determine qué tan grande puede ser la instancia del *Problema del Viajante de Comercio Euclidiano* si pretende:

   a) resolver el problema por optimalidad chequeando todos los posibles circuitos hamiltonianos (suponemos que el calcular el costo de cada ciclo hamiltoniano es una operación elemental).
   Sugerencia: Utilice *Scilab* para el cálculo de factoriales (ver tareas de laboratorio).

   b) obtener una solución factible, utilizando el *Algoritmo del Vecino más Cercano.*

   Responda las mismas preguntas de los apartados (a) y (b) para los casos en que la computadora fuera 10 y 100 veces más rápida.

4. Pruebe que la solución del *Problema del Viajante de Comercio en grafos* que genera el *Algoritmo del Vecino más Cercano* puede alejarse de la óptima tanto como se quiera.
   Sugerencia: Pruebe que, para todo $M \geq 8$, existe un grafo de 4 vértices y costos adecuados tales que la solución óptima del problema tiene costo 7 mientras que la solución por el *Algoritmo del Vecino más Cercano* tiene costo M.

5. Pruebe que la solución obtenida por el algoritmo *goloso* para el *Problema de Matching Euclídeo (de Peso Mínimo)* [LCO, pág. 4] puede acercarse tanto como se quiera al triple de la solución óptima.

6. Sea $G = (V, E)$ un grafo.

   a) Pruebe que si G es conexo, todo bosque maximal es un árbol.

   b) Pruebe que el *Problema del Árbol Generador Mínimo* se puede reducir al *Problema del Bosque de Peso Máximo.*

   c) Pruebe que, si $\mathcal{S} = E$ e $\mathcal{I} = \{J \subseteq \mathcal{S} : J \text{ es bosque de } G\}$, $(\mathcal{S}, \mathcal{I})$ es una matroide.

   d) Recordando que el algoritmo goloso resuelve correctamente el *Problema de Conjunto Independiente de Costo Máximo* sobre toda matroide y utilizando los items anterio-res, pruebe que el algoritmo de Kruskal (al final de esta práctica) resuelve correctamente el *Problema del Árbol Generador Mínimo.*

7. Las instancias de los problemas de *Matching Perfecto de Peso Mínimo* y *Matching de Peso Máximo* se definen por un grafo $G = (V, E)$ y costos $c \in \mathbb{R}^E$. En el primero, el objetivo es hallar (si existe) un matching perfecto $M \subset E$ de G tal que $\sum_{e \in M} c_e$ es mínimo. En el segundo, el objetivo es hallar un matching $M \subset E$ (no necesariamente perfecto) tal que $\sum_{e \in M} c_e$ es máximo.

   Pruebe que el Problema de Matching Perfecto de Peso Mínimo se reduce al Problema de Matching de Peso Máximo. ¿Cuántas operaciones elementales requiere esa reducción?

8. Sea G un grafo r-regular (es decir, tal que el grado de todos sus vértices es r) y sea k tal que $1 \leq k < r$. Pruebe que el Problema de k-dominación puede reducirse al Problema de $(r+1-k)$-empaquetamiento. ¿Cuántas operaciones elementales requiere la reducción?

9. Consideremos el *Problema del Camino más Corto* [LMDC, pág. 657; LCO, pág. 20]:

   Entrada: Un digrafo $D = (V, A)$, un vértice inicial $a \in V$ y un vector de costos $c \in \mathbb{R}^A$.
   Objetivo: Hallar, para cada $v \in V$, el costo del av-camino dirigido de costo mínimo.

   Pruebe que las siguientes variaciones del problema pueden reducirse al original en tiempo polinomial.

   a) *Problema del Camino más Corto no dirigido*:
   Entrada: Un grafo $G = (V, E)$, un vértice $a \in V$ y un vector de costos $c \in \mathbb{R}^E$.
   Objetivo: Hallar, para cada $v \in V$, el costo del av-camino de costo mínimo.

   b) *Problema del Camino de peso mínimo en vértices*:
   Entrada: Un grafo $G = (V, E)$, un vértice $a \in V$ y un vector de costos $c \in \mathbb{R}^V$.
   Objetivo: Hallar, para cada $v \in V$, el costo del av-camino de costo mínimo, donde el costo de un camino es la suma de los costos de cada uno de sus vértices.

10. Sean $D = (V, A)$, $c \in \mathbb{R}^A$ y $a \in V$ una instancia del Problema del Camino más Corto tal que para todo $v \in V$ existe al menos un av-camino dirigido en D. Pruebe que para todo $v \in V$ existe un av-camino dirigido de costo mínimo en D si y sólo si D no posee ningún ciclo dirigido $C \subset A$ de costo negativo (es decir, $\sum_{e \in C} c_e < 0$).

11. Sean $D = (V, A)$, $c \in \mathbb{R}^A$ y $a \in V$ una instancia del Problema del Camino más Corto. Decimos que $L \in \mathbb{R}^V$ es un *potencial factible* si $L(a) = 0$ y $L(v) + c_{vw} \geq L(w)$ para todo $vw \in A$ [LCO, pág. 20].

    a) Pruebe que para todo aw-camino dirigido P_w, $L(w)$ es cota inferior de $c(P_w)$, el costo de P_w.

    b) Pruebe que si L es un potencial factible y P_w un aw-camino dirigido tal que $L(w) = c(P_w)$, entonces P_w es un aw-camino dirigido de costo mínimo.

12. Considere el pseudocódigo del Algoritmo de Dijkstra dado al final de la práctica.

    a) Aplique el Algoritmo de Dijkstra sobre la siguiente instancia: $D = (V, A)$ con $V = \{1, 2, 3, 4\}$ y $A = \{(1, 2), (1, 3), (2, 4), (3, 2), (3, 4)\}$, $a = 1$, $c = (60, 40, 20, 10, 90)$.
    Observación: Puede verificar el resultado con la rutina `dijkstra` (ver tareas de laboratorio).

    b) Pruebe que, si todos los costos son no negativos, entonces el Algoritmo de Dijkstra resuelve el Problema del Camino más Corto.
    Sugerencia: Pruebe que, en cada iteración, L restringido al digrafo inducido por los vértices de V_2 es un potencial factible. Para ellos, pruebe que al finalizar una iteración: 1) $L(z) + c_{zv} \geq L(v)$ para todo $z \in N^-(v) \cap V_2$, 2) $L(v) + c_{vz} \geq L(z)$ para todo $z \in N^+(v) \cap V_2$ (piense lo que ocurre en la iteración en que se eligió z).

13. El *Algoritmo de Ford-Bellman* (ver pseudocódigo al final de la práctica) es correcto para resolver el Problema del Camino más corto con costos generales.

    Pruebe que:

a) El algoritmo siempre termina.

b) Si al terminar el algoritmo, "hubo_actualización" es falso, entonces efectivamente $L(v)$ contiene los tamaños de los caminos más cortos desde a hasta v.

c) **[Extra]** Si, al terminar, "hubo_actualización" es verdadero, entonces existe un ciclo dirigido con costo negativo.

14. *a*) Proponga y resuelva, utilizando la rutina `dijkstra`, una instancia sin ciclos dirigidos y con algún costo negativo donde el Algoritmo de Dijkstra falle.

    b) Proponga y resuelva, utilizando la rutina `ford_bellman`, una instancia donde exista un ciclo dirigido con costo negativo.

    (ver el uso de las rutinas en tareas de laboratorio)

15. Determine el orden de complejidad computacional de los algoritmos de Dijkstra y Ford-Bellman, en términos del tamaño de la instancia.

    Sugerencias: Determine la cantidad de operaciones elementales evaluadas en cada Paso de los algoritmos, en función del tamaño de los conjuntos involucrados en ese Paso. Analice, en términos de la cantidad de vértices y aristas, cómo varía el tamaño de los conjuntos que intervienen en cada paso (V_1, V_2, N^+) en el *peor* caso posible (el que genera mayor cantidad de operaciones elementales).

16. Utilice el algoritmo de Kruskal para encontrar un árbol generador mínimo de la instancia $G = (V, E); c$, con $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 6), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}$ y $c = (8, 18, 20, 23,\ 12, 30, 15, 30, 3, 7, 9)$. Compruebe sus iteraciones con la rutina de Scilab `kruskal.sci` (ver tareas de laboratorio al final).

17. Determine el orden de complejidad del algoritmo de Kruskal sobre una instancia $G = (V, E)$ con $|V| = n$, $|E| = m$, y $c \in \mathbb{R}^E$. Recuerde que el orden de complejidad está asociado al peor caso y tenga en cuenta las observaciones siguientes.

    Observaciones: Una implementación de Kruskal utiliza una representación de los datos que permite realizar cada operación con la siguiente complejidad:

Descripción de la operación	Cant. oper. elementales
1) Ordenar los arcos de E según su costo	$C_1 m \log m$
2) Inicializar $H = (V, F)$ con F vacío	$C_2 n$
3) Verificar si dos vértices dados pertenecen a la misma componente en H	C_3
4) Agregar una arista (u, v) a F tal que $u \in V_1$ y $v \in V_2$ donde V_1 y V_2 son diferentes componentes conexas en H	$C_4 \,\text{mín}\{\lvert V_1\rvert, \lvert V_2\rvert\}$

    donde C_1, C_2, C_3 y C_4 son constantes.

    De esta manera, el peor caso para el algoritmo de Kruskal se daría cuando, en cada iteración en la que se agrega un arco, la componente conexa de H de menor cardinal es *lo más grande posible*. Observar que Kruskal agrega un arco en $n - 1$ iteraciones de las m que realiza y que en la i-ésima iteración en que agrega un arco, el grafo H tiene $n - i + 1$ componentes conexas.

    - Pruebe que en un grafo con n vértices y k componentes conexas, la componente de menor cardinalidad tiene a lo sumo $\lfloor \frac{n}{k} \rfloor$ vértices.

- Para el orden de la complejidad del peor caso utilice la siguiente propiedad:

$$\sum_{k=1}^{n-1} \frac{1}{k} \leq 1 + ln(n-1).$$

Pseudocódigo de algoritmos:

ALGORITMO DE DIJKSTRA [LMDC, pág. 660; LCO, pág. 32]
Entrada: Un digrafo $D = (V, A)$, un vértice $a \in V$ y un vector de costos $c \in \mathbb{R}_+^A$.
Salida: Etiquetas de la forma $(L(v), ant(v))$ para todo $v \in V$ tales que $L(v)$ es el costo del av-camino dirigido P de costo mínimo y $ant(v)$ es el vértice anterior a v en P.

- **Inicialización**: Etiquetamos el vértice a con $(0, -)$ y el resto de los vértices de D con $(+\infty, -)$. También hacemos $V_1 \leftarrow \{a\}$ y $V_2 \leftarrow \varnothing$.
- **Iteración**: Mientras $V_1 \neq \varnothing$ hacer:

Paso 1. Tomamos el vértice $v \in V_1$ tal que $L(v)$ es mínimo.

Paso 2. Para todo vecino no visitado $w \in N^+(v) \backslash V_2$, si el costo de alcanzar a w pasando por v es menor al costo actual, es decir $L(v) + c_{vw} < L(w)$, entonces actualizamos la etiqueta de w con $(L(v) + c_{vw}, v)$ y agregamos w a V_1.

Paso 3. Eliminamos v de V_1 y lo agregamos en V_2.

ALGORITMO DE FORD-BELLMAN [LCO, pág. 29]
Entrada: Un digrafo $D = (V, A)$, un vértice $a \in V$ y un vector de costos $c \in \mathbb{R}^A$.
Salida: Si $i = |V|$ entonces existe un ciclo dirigido con costo negativo. Caso contrario, $L(v)$ contiene los tamaños de los caminos más cortos desde a hasta v, para todo $v \in V$.

- **Inicialización**: Etiquetamos el vértice a con $(0, -)$ y el resto de los vértices de D con $(+\infty, -)$. También hacemos $i \leftarrow 0$ y hubo_actualización $\leftarrow Verdadero$.
- **Iteración**: Mientras $i < |V|$ y hubo_actualización, hacer:

Paso 1. Hacemos $i \leftarrow i + 1$ y hubo_actualización $\leftarrow Falso$.

Paso 2. Para todo $vw \in A$, si $L(v) + c_{vw} < L(w)$ entonces actualizamos la etiqueta de w con $(L(v) + c_{vw}, v)$ y hacemos hubo_actualización $\leftarrow Verdadero$.

ALGORITMO DE KRUSKAL [LCO, pág. 19]
Entrada: Un grafo conexo $G = (V, E)$ y un vector de costos $c \in \mathbb{Z}^E$.
Salida: Un subconjunto de aristas $F \subset E$ tal que (V, F) es el árbol generador de costo mínimo de G.

- **Inicialización**: Hacemos $H = (V, F)$ con $F = \emptyset$ e indexamos los arcos de G de forma que $E = \{e_1, e_2, \ldots, e_m\}$ verifique $c_{e_i} \leq c_{e_{i+1}}$ para todo $i = 1 \ldots, m-1$ (requiere aplicar un *algoritmo de ordenamiento*).
- **Iteración**: Para cada $i = 1, \ldots, m$, hacer:

Paso 1. Si los extremos de e_i están en distintas componentes conexas de H, entonces $F \leftarrow F \cup \{e_i\}$.

Tareas de Laboratorio:

Para la instalación y uso del Scilab, comience realizando los siguientes pasos:

- Descargue el archivo denominado `Instalador-scilab-5.4.1.exe` de la pág. de la Cátedra y ejecútelo. Luego de realizada la instalación, puede borrar este archivo.
- Cree en su máquina un directorio de trabajo y descargue en éste los archivos `kruskal.sci`, `dijkstra.sci` y `ford_bellman.sci` de la pág. de la Cátedra.

- Cada vez que ejecute *Scilab*, antes de comenzar a trabajar, elija el en el *Navegador de Archivos* el directorio de trabajo creado en el ítem anterior.

Para trabajar con los ejercicios de la práctica:

- Factoriales (Ej. 3.a): Por ejemplo, para calcular 20! ejecute `factorial(20)`. La respuesta `2.433D+18` significa $2{,}433 \times 10^{18}$.
- Algoritmo de Kruskal (Ej. 16): Por ejemplo, los siguientes comandos resuelven la instancia.

```
exec('kruskal.sci',-1)
G(1,:) = [1 1 1 1 2 2 3 3 4 4 5];
G(2,:) = [2 3 4 5 3 6 4 5 5 6 6];
costos = [8 18 20 23 12 30 15 30 3 7 9];
[F, valor] = kruskal(G, costos)
```

- Algoritmos de Dijkstra y Ford-Bellman (Ej. 12 y 14): Por ejemplo, los siguientes comandos resuelven la instancia del Ej. 12.a con Dijkstra:

```
exec('dijkstra.sci',-1)
D(1,:) = [1 1 2 3 3];
D(2,:) = [2 3 4 2 4];
costos = [60 40 20 10 90];
a = 1;
[L, ant] = dijkstra(D, costos, a)
```

  El uso de la rutina de Ford-Bellman es idéntico. Ambas rutinas toman como entrada el digrafo, el cual está representado como una matríz de 2 filas por m columnas en donde cada columna $(u, v)^T$ define un arco que va desde el vértice u a v, un vector de costos en donde la componente i-ésima se correponde con el arco i-ésimo del digrafo, y un vértice inicial. Ambas rutinas retornan los vectores L y ant, en donde la componente i-ésima de cada uno de ellos se corresponde con el vértice i.

Bibligrafía:

[LMDC] R. Grimaldi. *Matemática Discreta y Combinatoria*. 3ra. edición. Pearson.

[LCO] W. Cook, W. Cunningham, W. Pulleyblank A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience.