Algorísmia QT 2022–2023

Examen Parcial **28 d'octubre de 2022**

Durada: 1h 45min

Instruccions generals:

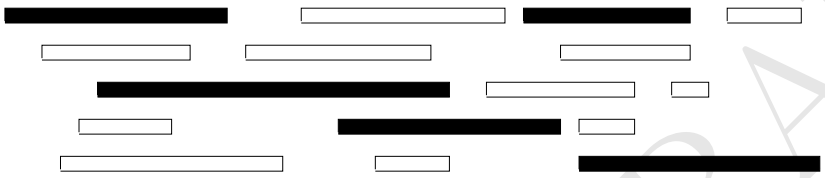
- Entregueu per separat les solucions de cada exercici (Ex 1, Ex 2, Ex 3 i Ex 4).
- Heu de donar i argumentar la correctesa i l'eficiència dels algorismes que proposeu. Per fer-ho podeu donar una descripció d'alt nivell de l'algorisme suficient per tal que, amb les explicacions i aclariments oportuns, justifiqueu que l'algorisme és correcte i té el cost indicat.
- Podeu fer ús d'algorismes que s'han vist a classe, però igualment cal desenvolupar-ne la justificació de la correctesa i calcular-ne o justificar-ne el cost, quan així es demani. Si la solució és una variació d'un algorisme vist a classe, n'haureu de donar també els detalls d'aquesta variació i la seva afectació sobre el cost.
- Es valorarà especialment la claredat i concisió de la presentació.
- La puntuació total d'aquest examen és de **10 punts**.

©Professorat GRAU-A

1. [Camins de recobriment] (2.5 punts):

Donat un conjunt d'intervals $X = \{I_1, I_2, \ldots, I_n\}$ sobre la recta real $\mathbb{R}$, un *camí de recobriment* és un subconjunt $Y \subseteq X$ tal que tot punt $x \in \mathbb{R}$ contingut en algun interval $I_j \in X$ està contingut en algun interval d'Y. Dissenyeu un algorisme que trobi un *camí de recobriment mínim* (és a dir, amb el mínim nombre possible d'intervals) per a un conjunt d'intervals X donat. Justifiqueu la correcció del teu algorisme i calcula'n el cost en funció d'n.

Aquesta figura us mostra un exemple gràfic: els intervals emplenats en negre són un camí de recobriment mínim per al conjunt d'intervals donat (tots els rectangles).



Una solució: El algoritmo voraz que propongo es el siguiente:

Ordenar los intervalos en orden creciente de tiempo de inicio y en caso de empate por orden decreciente de tiempo de finalización.
$S = \{I_1\}$, $\ell = 1$, $k = 1$, $j = k$
while $j \leq n$ **do**
  $y = I_k.y$, $j = k + 1$,
  while $j \leq n$ and $I_j.x \leq y$ **do**
    if $I_j.y > I_k.y$ **then**
      $k = j$,
    $j = j + 1$
  if $k \neq \ell$ **then**
    $S = S \cup \{I_k\}$, $\ell = k$, $y = I_k.y$
  else
    if $j \leq n$ **then**
      $S = S \cup \{I_j\}$, $\ell = j$, $k = j$

El algoritmo selecciona el primer intervalo y, de entre los que empiezan antes o al mismo tiempo de que este acabe, el que termina más tarde. En caso de que no haya intervalos cumpliendo esta condición y queden intervalos sin tratar, se queda con el siguiente y aplica la misma regla voraz.

Para comprobar que el algoritmo es correcto, tenemos que ver que S al final del algoritmo es una solución al problema y que contiene el número mínimo de intervalos posibles. Tenemos que considerar dos casos, el primero en que los intervalos cubren un espacio contiguo de la recta real y el segundo el caso en que esto no ocurre. Basta con demostrar la corrección en el primer caso, ya que así la tendremos para cada uno de los tramos en el segundo caso.

Veamos primero que, cuando los intervalos cubren un espacio contiguo, S al finalizar el algoritmo es una solución. Por contigüidad, siempre hay intervalos con tiempo de inicio $\leq y$ y que acaban después, salvo para el intervalo que acaba el último. Esto garantiza que entre

© Professorat GRAU-A

dos intervalos añadidos a S consecutivamente se cubre todos los valores desde el inicio del primero hasta la finalización del último. Por lo que S cubre todo el espacio cubierto por la entrada.

Para demostrar que $S = \{I'_1, \ldots, I'_k\}$ es una solución óptima, consideremos una solución S^* óptima cualquiera diferente de S. Para comparar las dos soluciones, asumamos que los intervalos en $S^* = \{I^*_1, \ldots, I^*_\ell\}$ están ordenados en orden creciente de tiempo de inicio. Cómo la solución es óptima, $\ell \leq k$, además en S^* nunca hay dos intervalos con el mismo tiempo de inicio, si no el más corto sobraría.

Sea j el primer intervalo en el que S y S^* difieren, si $j = 1$, I^*_1 tiene que empezar a la vez que I'_1 que es uno de los intervalos con el primer tiempo de inicio. Pero $I'1$ acaba igual o más tarde que I^*_1, de acuerdo con el criterio de ordenación. Así podemos reemplazar I^*_1 con I'_1 cubriendo todo el espacio que cubría I^*_1. Si $j > 1$, I^*_j tiene que empezar después de que I^*_{j-1} empiece, si no podríamos eliminar I^*_{j-1} y seguiríamos teniendo una solución con un intervalo menos. Por tanto, de acuerdo con el criterio de elección de nuestro algoritmo, I'_j acaba después o en el mismo instante que I^*_j. Así podemos reemplazar I^*_j por I'_j y seguir teniendo una solución óptima.

Repitiendo este proceso acabaremos teniendo una solución óptima que coincide con S, por lo que S es óptima.

Comentari: El algoritmo simétrico que ordena por orden decreciente de tiempo de finalización y que, de entre los intervalos que acaban al mismo tiempo o después del seleccionado, añade a la solución el que empieza antes también es una solución válida.

©Professorat GPAL-A

2. [Ordenar k-multiconjunts] (2.5 punts):

Un *k-multiconjunt* és un multiconjunt amb k elements diferents, cadascun dels quals apareix exactament n/k cops. Per exemple, $\{1, 1, 2, 2, 3, 3, 4, 4, 5, 5\}$ és un 5-multiconjunt (ordenat) de mida $n = 10$. Doneu un algorisme $\Theta(n \lg k)$ per a ordenar un k-multiconjunt de mida n. Considerem que $n = 2^i$ i $k = 2^j$ per a alguns i i j, $i \geq j$.

Una solució: Sigui S un k-multi conjunt. Considerem el següent algorisme:

- Utilitzar selecció determinista per a trobar la mediana s_m de S.
- partim S al voltant de s_m, treien els elements amb valor s_m que emmagatzemen a un vector Aux.
- Com hi ha n/k elements a S que son iguals a s_m, la partició divideix els elements restants de S en dues parts, cada part té com a molt $n/2$ elements i apareixen com a molt $k/2$ valors diferents. Siguin S_l i S_r les dues meitats,
- Aplicar recursivament l'algorisme a cada meitat (mentre aquestes tinguin grandària $\geq n/k$. Tornarem S_l ordenat, seguit de Aux i seguit de S_r ordenat.

A cada nivell de la recursió, dividim S en dos subconjunts amb grandària $\leq |S|/2$ en $O(n)$ passos. L'algorisme s'aturara quan arribe a subproblemes amb grandària n/k, es a dir $n/2^j$. Per tant l'alçada de l'arbre de recursió serà $j = \lg k$ (fins arribar a un S amb grandària n/k). A cada nivell, el cost de cada crida recursiva és lineal, i les crides es fan sobre conjunts disjunts de valors. Per tant, el cost d'un nivell és $O(n)$. Així tenim cost total $O(n \lg k)$.

Comentari: Un algorisme que ordena el multiconjunt, però no en el temps demanat.

Podem ordenar el multiconjunt donat a un vector A mantenint un vector V ordenat que contingui els valors que trobem al vector i associant a cada valor del vector V una cua de les posicions dels elements del vector d'entrada que contenen aquest valor.

L'algorisme és el següent:

- Guardem $A[0]$ a $V[0]$ i 0 a la cua associada a $V[0]$.
- per $1 \leq i < n$, cerquem $A[i]$ a V amb cerca dicotòmica. Si trobem el valor afegim i a la cua associada. En cas contrari, inserim el valor $A[i]$ en la seva posició a V i inserim i a la cua associada a la posició corresponent de V.
- Finalment, obtenim la sortida recorrent en ordre, una darrera de l'altre, les cues associades als elements d'V.

L'algorisme és correcte, ja que cada llista té posicions amb el mateix valor i es manté ordenada al llarg de tota l'execució.

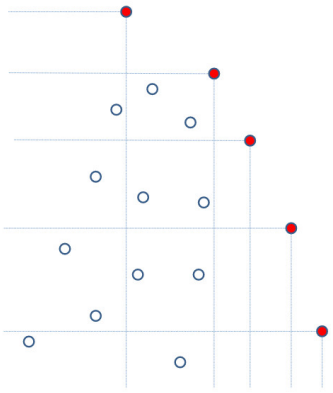
Com V té sempre k o menys elements la cerca dicotòmica té cost $O(\log k)$. Per tant, el cost total de les cerques a V és $O(n \log k)$. Per un altra part, tenim el cost d'inserció, que és com a molt $O(k^2)$. L'algorisme té el cost demanat sempre que $k^2/logk = O(n)$, observem que això no passa sempre, per exemple quan $n = ck$ per alguna constant c.

© Professorat GRAU-A

3. [Òptims de Pareto] (2.5 punts):

Considereu un conjunt $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ d'n punts en el pla Euclidià (2-dimensional). Diem que un punt (x_i, y_i) *domina* un altre punt (x_j, y_j) si és més gran en totes dues coordenades, és a dir, si $x_i > x_j$ i $y_i > y_j$. Diem que un punt és *maximal* si no és dominat per cap altre.

Per exemple, al conjunt de punts mostrat en aquesta figura, els punts sòlids són els punts maximals, mentre que els punts buits són dominats. Les línies discontínues emmarquen quins punts són dominats per cadascun dels punts maximals.



Donat un conjunt P d'n punts, dissenyeu un algorisme eficient per a trobar tots els punts maximals de P (els anomenats *Òptims de Pareto*).

Una solució: [Un voraç] Podem observar que el punt amb coordenada x més gran, i si n'hi ha més d'un, aquell que té cordenad y més gran, sempre forma part de la solució ja que ningú el domina. Observem també que un punt (x, y) maximal dominarà tots els punts (x', y') tals que $x' \leq x$ i $y' \leq y$ i, en conseqüència, els punts dominats segur que no formen part de cap solució.

Tenint en compte aquest principi de suboptimalitat, podem plantejar el següent algorisme voraç:

- Ordena els punts decreixentment per l'abscissa x i en cas d'empat per ordre decreixent d'ordenada y.
- Considereu el primer punt p. Aquest punt és definitivament un punt òptim de Pareto. Guardem el punt, $p_\ell = p$, aquesta variable guardarà l'últim punt afegit a la solucio.
- Aneu recorrent l'entrada fins a trobar un p amb coordenada x menor que la de p_ℓ i coordenada y major que la de p_ℓ. Aquest punt és òptim de Pareto, a més els punts descartats estàn dominats per p_ℓ i no ho son. Fem $p_\ell = p$
- Segueix fent-ho fins a processar tota l'entrada.

El cost total de l'algorisme és el de l'ordenació, ja que la segona part només requereix $O(n)$ passos. Per tant, l'algorisme té cost $O(n \log n)$.

© Professorat GRAU-A

Una solució: [*D&C*:]

function ParetoOptimal (P)
  $P' \leftarrow$ Sort(P) ▷ Creixentment per coordenada x, empats decreix per coordenada y.
  return ParetoOptimalDC(P')

Require: $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ ordenat creixentment pels valors d'x
function ParetoOptimalDC (P)
  if $n == 1$ **then**
    return (x_1, y_1)
  else
    $m \leftarrow \lceil n/2 \rceil$
    $E \leftarrow \{(x_1, y_1), \ldots, (x_m, y_m)\}$
    $D \leftarrow \{(x_{m+1}, y_{m+1}), \ldots, (x_n, y_n)\}$
    $PE \leftarrow$ ParetoOptimalDC(E) ▷ Ordre decreixent d'y
    $PD \leftarrow$ ParetoOptimalDC(D) ▷ Ordre decreixent d'y
    $y_{max} \leftarrow$ coordenada y del primer punt a PE.
    $P = []$ ▷ llista buida
    for $(x_e, y_e) \in PE$ **do**
      if $y_e > y_{max}$ **then** ▷ És un punt maximal esquerre no dominat
        $P = P \cdot \{(x_e, y_e)\}$
    return $P \cdot PD$ ▷ Concatenació de llistes

L'algorisme és correcte, ja que els punts a PE tenen coordenada x menor que els punts a PD. Així tots els punts a PE que tenen coordenada y menor que y_{max} estan dominats pel primer punt de Pareto de PD i els altres no estan dominats per cap punt a PD.

Cost `ParetoOptimalDC`: $T(n) = O(n) + 2T(n/2)$, per tant, $T(n) = O(n \log n)$.

Cost `ParetoOptimal` $T(n) = O(n \log n)$.

Comentari: Un altra solució seria fer D&C però sense ordenar prèviament; fent la cerca de la mediana de cadascuna de les dues parts en temps lineal abans de fer la partició.

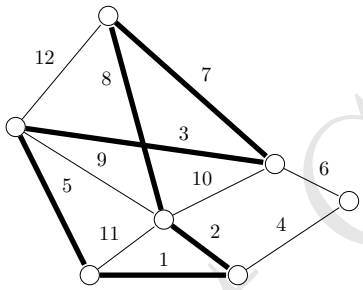
©Professorat GRAU-A

4. [Conjunt de retroalimentació mínim] (2.5 punts):

Donat un graf no dirigit i connex $G = (V, E)$, anomenem *conjunt d'arestes de retroalimentació* a un subconjunt $F \subseteq E$ d'arestes tal que cada cicle de G conté almenys una aresta a F. Com a conseqüència, després d'eliminar de G totes les arestes d'un conjunt de retroalimentació F, obtenim un graf acíclic.

Descriviu i analitzeu un algorisme ràpid per a calcular un conjunt de retroalimentació de pes total mínim d'un graf ponderat, no dirigit i connex $G = (V, E, w)$, amb pesos a les arestes $w : E \to \mathbb{R}$. Com és habitual el pes total d'un conjunt d'arestes ponderades és la suma dels pesos de les arestes del conjunt.

Aquesta figura us mostra un exemple gràfic: les arestes realçades en negreta formen un conjunt d'arestes de retroalimentació de pes mínim del graf ponderat mostrat.



Una solució: Si $F \subseteq E$ és un conjunt de retroalimentació, tenim que el graf $(V, E \setminus F)$ es acíclic. A més $w(F) = w(E) - w(E \setminus F)$. També, podem veure que si F és un conjunt de retroalimentació, qualsevol superconjunt de F es també conjunt de retroalimentació. En particular E és un conjunt de retroalimentació.

Observem que quan tots els pesos són positius, minimitzar $w(F)$ és equivalent a maximitzar $w(E \setminus F)$, ja que els dos valors són no negatius i la seva suma no depèn de la partició. En canvi, si hi ha arestes amb pes negatiu, ens interessa afegir-les al conjunt de retroalimentació, perwuè això disminuirà sempre el pes d'aquest conjunt. Per tant hem de tractar de forma diferent els dos casos.

Analitzem primer el cas en què tots els pesos són no negatius. En aquest cas, el complement d'un conjunt de retroalimentació de pes mínim és un graf acíclic de pes màxim i ha de ser un arbre d'expansió. Si no fos així, com el graf és connex, podríem treure al menys una aresta del conjunt de retroalimentació sense crear cap cicle amb les arestes de fora i mantenir o decrementar el pes del conjunt de retroalimentació. Per tant, és suficient trobar un arbre d'expansió amb pes màxim. Per fer això n'hi ha prou amb multiplicar per -1 els pesos del graf i obtenir un MST sota aquesta ponderació. El conjunt de retroalimentació cercat serà el complement d'aquestes arestes.

Quan el graf té arestes amb pes negatiu, sabem que totes les arestes amb pes negatiu aniran al conjunt de retroalimentació. Així hem de resoldre el problema amb el graf $G^+ = (V, E^+)$ on $E^+ = \{e \in E \mid w(e) \geq 0\}$. Al treure les arestes amb pes negatiu es possible que G' no sigui connex, en aquest cas G' no té un arbre d'expansió. Però el complementari del conjunt de retroalimentació mínim estarà compost per un arbre d'expansió de cost màxim a cada

© Professorat GRAUA

component connexa de G'. Així, per construir F haurem de treure les arestes d'un bosc d'expansió de cost màxim a G^+.

Per trobar el bosc d'expansió amb pes màxim podem fer servir l'algorisme de Kruskal, ordenant les arestes de major a menor pes, això és equivalent a seguir l'ordre dels valor multiplicats per -1, i aturant l'algorisme quan arribem a una aresta amb pes negatiu. Així aquesta part té cost $O(m \log m)$. Finalment, hem de treure les arestes que apareixen al bosc, ho podem fer amb cost $O(m)$ aprofitant que podem obtenir les arestes del bosc d'expansió en ordre decreixent de pes. El cost total és $O(m \log n)$.

©Professorat GRAU-A