UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

DEVOIR 2

PAR

JÉRÉMY BOUCHARD (BOUJ08019605)

JEAN-PHILIPPE SAVARD (SAVJ04079609)

ALEXIS VALOTAIRE (VALA09129509)

DEVOIR PRÉSENTÉ À

M. FRANÇOIS LEMIEUX

DANS LE CADRE DU COURS D'ALGORITHMIQUE (8INF433)

Note : Le code source LaTeX est disponible à l'URL suivant :
`https://github.com/AlexisCode101/algo-devoir-2`

Question 1

Le théorème sur les récurrences que nous avons vu en classe possède une version plus simple dans le cas où la fonction f(n) est un polynôme. Lorsque la récurrence est de la forme:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$

où c et k sont des constantes réelles positives, alors la solution est donnée par:

$$T(n) \begin{cases} \Theta\left(n^{log_b a}\right) & \text{si } a > b^k \\ \Theta\left(n^k lg\ n\right) & \text{si } a = b^k \\ \Theta\left(n^k\right) & \text{si } a < b^k \end{cases}$$

Nous avons vu en classe la démonstration du premier cas (où $a > b^k$). Utilisez le théorème sur les récurrences vu en classe pour démontrer les deux autres cas.

Cas 2:

Nous voulons démonter le cas où:

$$a = b^k$$

Pour ce faire, on commence par mettre le tout en log_b, ce qui donne:

$$log_b(a) = log_b(b^k)$$

Nous savons aussi qu'un log en base 'X' de 'X' équivaut à 1, donc:

$$log_b(a) = k$$

Donc, on peut dire que:

$$n^k = n^{log_b(a)}$$

Ainsi, nous satisfaisons la condition du cas 2 du théorème de récurrence, qui est:

$$f(n) = \Theta\left(n^{log_b(a)}\right)$$

Donc, par le fait même, on peut dire que:

$$T(n) = \Theta\big(n^{log_b(a)}\ lg\ n\ \big) = \Theta\big(n^k\ lg\ n\ \big)$$

Cas 3:

Pour ce cas, nous avons deux conditions à satisfaire:

(a) il existe $\epsilon > 0$ tel que $f(n) = \Omega\big(n^{log_b a+\epsilon}\big)$

(b) il existe $c < 1$ tel que $af(n/b) \leq cf(n)$

Condition a:

Comme dans le cas 2, nous pouvons dire:

$$a < b^k = log_b(a) < k$$

Nous pouvons aussi dire que:

$$log_b(a) < k = 0 < -log_b(a) + k$$

On peut alors supposer que:

$$\epsilon = -log_b(a) + k$$

Si on isole la valeur 'K', nous obtenons:

$$k = log_b(a) + \epsilon$$

Nous pouvons alors dire que:

$$n^k = n^{log_b(a)+\epsilon}$$

Ainsi, nous respectons la première condition qui dit qu'il existe $\epsilon > 0$ tel que $f(n) = \Omega\big(n^{log_b a+\epsilon}\big)$

Condition b:

En divisant toute l'équivalence par b^k, nous obtenons:

$$a < b^k = \frac{a}{b^k} < 1$$

Nous pouvons alors supposer que:

$$c = \frac{a}{b^k}$$

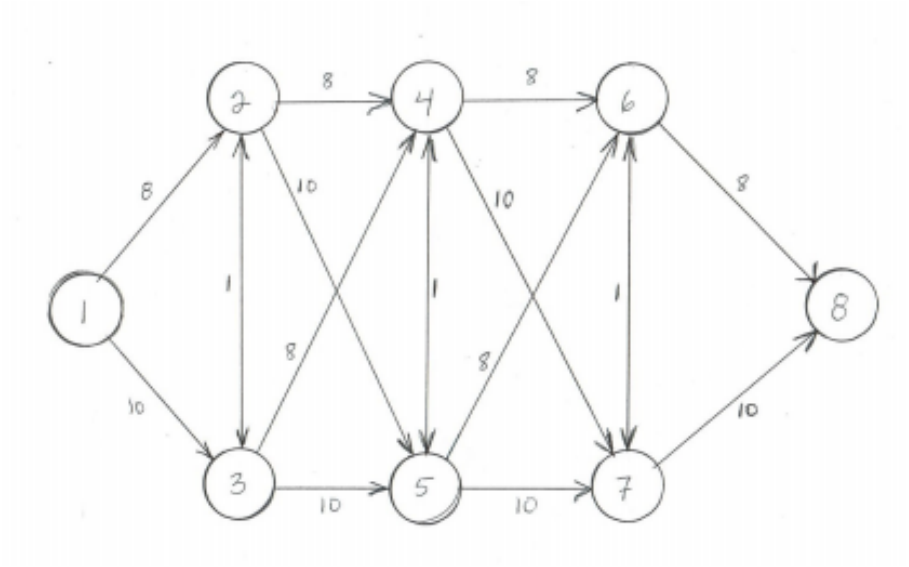
Ainsi, nous respection la deuxième condition qui dit qu'il existe $c < 1$ tel que $af(n/b) \leq cf(n)$

Question 2

L'algorithme de Dijkstra que nous avons vu en classe, nous permet d'obtenir la longueur du plus court chemin entre un noeud source et tous les autres noeuds mais il ne donne pas les chemins eux-mêmes. Comment modifieriez-vous cet algorithme afin d'obtenir cette information? Notez que la modification est mineure si vous choisissez la bonne façon de représenter les chemins.

Question 3

Considérez le graphe dirigé de la figure suivante:



Montrez le contenu du tableau D, de l'ensemble S ainsi que la valeur de la variable v après chacune des itérations de l'algorithme. Décrivez toutes les étapes de l'algorithme Dijkstra. Utilisez le noeud 1 comme source.

Question 4

Résoudre les récurrences suivantes.

a) $T(n) = 2T\left(\frac{n}{2}\right) + 1$

b) $T(n) = T\left(\frac{9n}{10}\right) + n$

c) $T(n) = 16T\left(\frac{n}{4}\right) + n^2$

d) $T(n) = 7T\left(\frac{n}{3}\right) + n^2$

e) $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

f) $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$

g) $T(n) = 3T\left(\frac{n}{2}\right) + 5n$

h) $T(n) = T\left(\frac{n}{2} + 5\right) + 1$

Question 5

Considérez la matrice suivante

$$F = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Soit i et j, deux entiers. Quel est le produit du vecteur $(i,\ j)$ et de la matrice F? Qu'arrive-t-il si i et j sont deux nombres consécutifs de la suite de Fibonacci? Utilisez cette idée pour concevoir un algorithme diviser-pour-régner capable de calculer le nième nombre de Fibonacci en temps $\Theta(log\, n)$ en considérant (faussement) que les multiplications prennent un temps constant. Expliquez le lien entre votre algorithme et l'algorithme fib3 du devoir 1.

Il est possible de constater que le résultat de la multiplication suivante :

$$\begin{pmatrix} f_{n-1} & f_{n-2} \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} f_{n-1} & f_{n-1} + f_{n-2} \end{pmatrix}$$

Il est possible de se rapeller l'identité de la séquence de Fibbonaci :

$$f_n = f_{n-1} + f_{n-2}$$

Il est donc possible de remplacer une identité dans la première équation :

$$\begin{pmatrix} f_{n-1} & f_{n-2} \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} f_{n-1} & f_n \end{pmatrix}$$

On remarque qu'on peut chaîner la multiplication de la matrice. On observe que l'indice de la séquence de Fibonacci augmente à chaque itération. On peut observer cela grâce aux équations suivantes.

$$\begin{pmatrix} f_0 & f_1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} f_0 & f_0 + f_1 \end{pmatrix}$$

$$\begin{pmatrix} f_1 & f_2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} f_1 & f_1 + f_2 \end{pmatrix}$$

En chaînant les multiplications par la matrice, on constate la relation suivante.

$$\begin{pmatrix} f_0 & f_1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} f_n & f_{n+1} \end{pmatrix}$$

En remplacant f_0 et f_1 par leurs valeurs respectives.

$$\begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} f_n & f_{n+1} \end{pmatrix}$$

Question 6

Soit $X[1 \dots n]$ et $Y[1 \dots n]$ deux tableaux, chacun contenant n nombres déjà triés. Donnez un algorithme diviser-pour-régner pour trouver le médian des $2n$ éléments présents dans les tableaux X et Y. Le temps d'exécution de votre algorithme doit être dans $\Theta(log\, n)$ en pire cas.

L'algorithme suivant trouve la médiane de deux tableaux de même taille (taille 'n') de nombres triés. Il est à noter que la fonction 'Mediane' utilisée aux lignes 7 et 8 est une fonction qui s'exécute en temps constant qui retourne la médiane d'un tableau.

Algorithme 1 : TrouverMédiane

Données : $X[1 \cdots n], Y[1 \cdots n], entier\ n$
Résultat : réel med

```
1  si  n = 1 alors
2  |   retourner ((X[1]+Y[1])/2)
3  fin
4  si  n = 2 alors
5  |   retourner ((max(X[1],Y[1])+min(X[2],Y[2]))/2)
6  fin
7  medX ← Mediane(X[1 ··· n]) ;
8  medY ← Mediane(Y[1 ··· n]) ;
9  si medX = medY alors
10 |   retourner medX
11 fin
12 si medX < medY alors
13 |   si n mod 2 = 0 alors
14 |   |   retourner TrouverMédiane(X[n/2 + 1 ... n], Y[1 ... n/2 − 1], n/2 + 1)
15 |   sinon
16 |   |   retourner TrouverMédiane(X[n/2 ... n], Y[1 ... n/2], n/2)
17 |   fin
18 sinon
19 |   si n mod 2 = 0 alors
20 |   |   retourner TrouverMédiane(X[1 ... n/2 − 1], Y[n/2 + 1 ... n], n/2 + 1)
21 |   sinon
22 |   |   retourner TrouverMédiane(X[n/2 ... n], Y[1 ... n/2], n/2)
23 |   fin
24 fin
```