Université de Liège

Programmation avancée

INFO2050

Projet 3 : Mise en page automatique d'une bande dessinée

Noémie Lecocq **(s130165)**
Andrew Sassoye **(s160135)**

2017-2018

1 Répartition des cases

a)

Une approche exhaustive consisterait à calculer toutes les possibilités de répartition des images puis de chercher celui avec le coût minimal.

Si il y a 1 image a répartir, il n'y a qu'une possibilité.

- 1 image sur la première ligne

Si il y a 2 images à répartir, il y a deux cas possible.

- 2 images sur la première ligne
- 1 image sur la première ligne et une sur la seconde.

Si il y a 3 images à répartir, il y a 4 possibilités.

- 3 images sur la première ligne
- 2 images sur la première ligne et une sur la seconde
- 1 images sur la première ligne et 2 sur la seconde
- 1 image par ligne

À chaque ajout d'une image, on multiplie par deux le nombre de possibilités de répartitions. Dans une cas plus générale on peut dire que la complexité est de $2^n - 1$ avec n le nombre d'images à répartir.

b)

$$c(i) = \begin{cases} C(0,0) & i = 0 \\ min_{0 \leq k \leq i}(c(k) + C(k,i)) & i > 0 \end{cases}$$

où $C(i,j)$ est le coût pour mettre une image i et j sur la même ligne.

c)

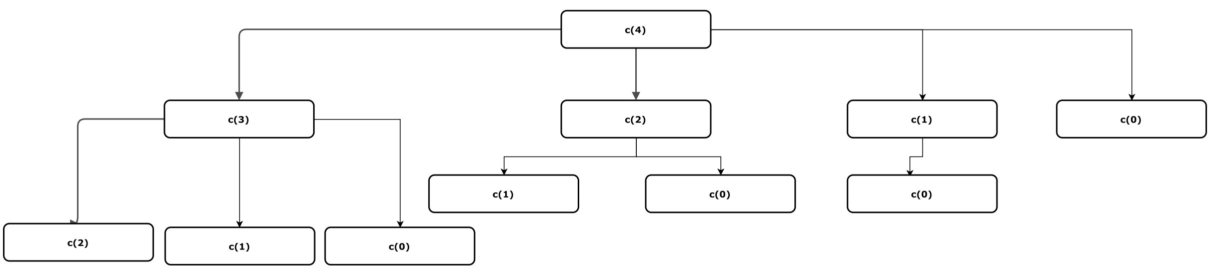

FIGURE 1 – Graphe des appels récursifs

d)

```
OPTIMAL(j, nbImages)
if j < nbImages + 1
    if j == 0
        optimalCost[j] = 0
    else
        optimalCost[j] = +∞
        for i = 1 to j
            if (optimalCost[i − 1] + costMatrix[i − 1][j − 1] < optimalCost[j])
                optimalCost[j] = optimalCost[i − 1] + costMatrix[i − 1][j − 1]
                mem[j] = i
    optimal(j + 1, nbImages)
return mem
```

e)

La complexité temporelle est $O(n^2)$ et la complexité spatiale dépends de costMatrix

1 Calcul de la couture d'énergie minimale

a)

Une approche exhaustive consisterait à calculer le coût de tous les chemins possibles puis de chercher celui avec le coût minimal. Pour atteindre le pixel (i, j), i étant la dernière ligne :

Si la hauteur = 1, 1 chemin possible.
Si la hauteur = 2, 3 chemins possibles. [1]
Si la hauteur = 3, $3 * 3 = 9$ chemins possibles.
Si la hauteur = n, $3 * 3 * ... = 3^{n-1}$ chemins possibles.
L'approche exhaustive est donc bien de complexité exponentielle.

b)

Cas de base, $i = 0$, $j \in [0, largeur - 1]$;
$C(i, j) = E(i, j)$

$\forall i > 0, j \in [0, largeur - 1]$ tels que $C(i-1, j-1)$, $C(i-1, j)$ et $C(i-1, j+1)$ sont définis :
$C(i, j) = E(i, j) + min(C(i-1, j-1), C(i-1, j), C(i-1, j+1))$ [2]

1. Si le pixel est au bord de l'image alors il n'y a que 2 chemins possibles, mais on ne va pas considérer ce cas ici pour plus de simplicité
2. Si j - 1 ou j + 1 dépassent les limites de l'image, on n'en tient pas compte dans le calcul du minimum.

c)

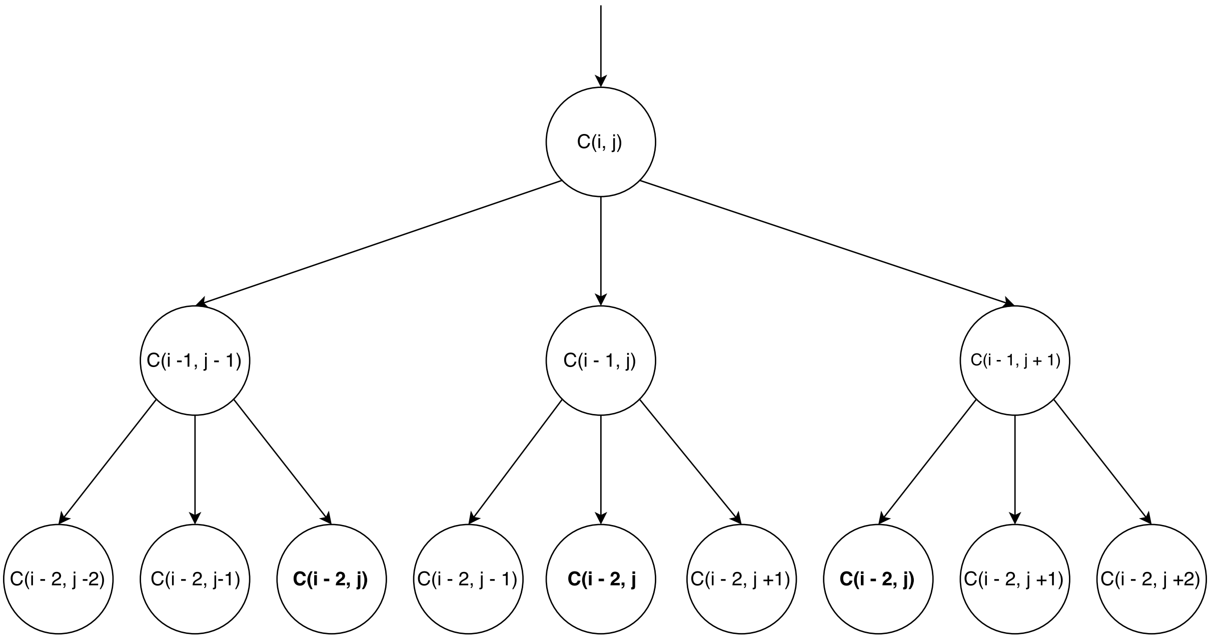

FIGURE 2 – Graphe des appels récursifs

On peut voir dans la figure 2 que l'on effectue plusieurs fois le même appel (en gras). Afin de ne pas faire inutilement des calculs, on tâchera de retenir les valeurs déjà calculées.

d)

```
COST(energy)
for j = 1 to width
    cost[1][j] = energy[1][j]
for i = 2 to heigth
    for j = 1 to width
        if j − 1 > 1
            left = cost[i − 1][j − 1]
        mid = cost[i − 1][j]
        if j + 1 < width
            right = cost[i − 1][j + 1]
        // Si left ou right n'est pas défini, on n'en tient pas compte
        cost[i][j] = energy[i][j] + min(left, mid, right)
return cost
```

energy est un tableau de taille $heigth * width$ contenant l'énergie de chaque pixel.

e)

Pour une image de taille n * m, la complexité est $\Theta(n * m)$.
L'espace mémoire utilisé est constant.

2 Fonctions de réduction et d'élargissement d'une image

a) Implémentation

Pour chacune des fonctions, on utilise deux tableaux de même taille que l'image : *energy* et *sum*. On effectue une boucle k fois :

On calcule l'énergie de chaque pixel et on la place dans *energy*. Ensuite on calcule le coût de chaque pixel, et on enregistre les résultats dans *sum*. Finalement on cherche le minimum dans la dernière ligne de *sum* et parcourt *sum* du bas vers le haut à partir de ce point afin de reconstituer la couture d'énergie minimale que l'on enregistre dans *seam*.

Dans le cas d'un élargissement, on note dans un tableau *marked*, de même taille que l'image, les pixels de la couture d'énergie minimale.

Ensuite on recopie l'image en enlevant les pixels de la couture d'énergie minimale, on a donc une image réduite d'un pixel en largeur.

Après cette boucle le programme se termine et renvoie la nouvelle image réduite k fois dans le cas d'une réduction.

S'il s'agit d'un élargissement, on recopie l'image originale et pour chaque pixel marqué on ajoute un pixel à sa droite comme expliqué dans l'énoncé. On retourne ensuite cette image de k pixels plus large.

b) Complexités

Pour les deux fonctions la complexité en espace est constante puisqu'on modifie directement dans les images et les tableaux. La complexité de **reduceImageWidth** est $\Theta(k * n * m)$.

La complexité de **increaseImageWidth** est $\Theta(k * n * m)$ si k est inférieur à 20% de la largeur originale de l'image. Si k est supérieur à 20%, on devra appeler n fois la fonction pour ne pas dépasser $k > 20\% * m$.