

UNIVERSITÉ DE LIÈGE

PROGRAMMATION AVANCÉE

INFO2050

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

Noémie Lecocq
s130165
Andrew Sassoie

2017-2018



1 Répartition des cases

- a)
- b)
- c)
- d)
- e)

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 , $3 * 3 * \dots = 3^{n-1}$ 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)

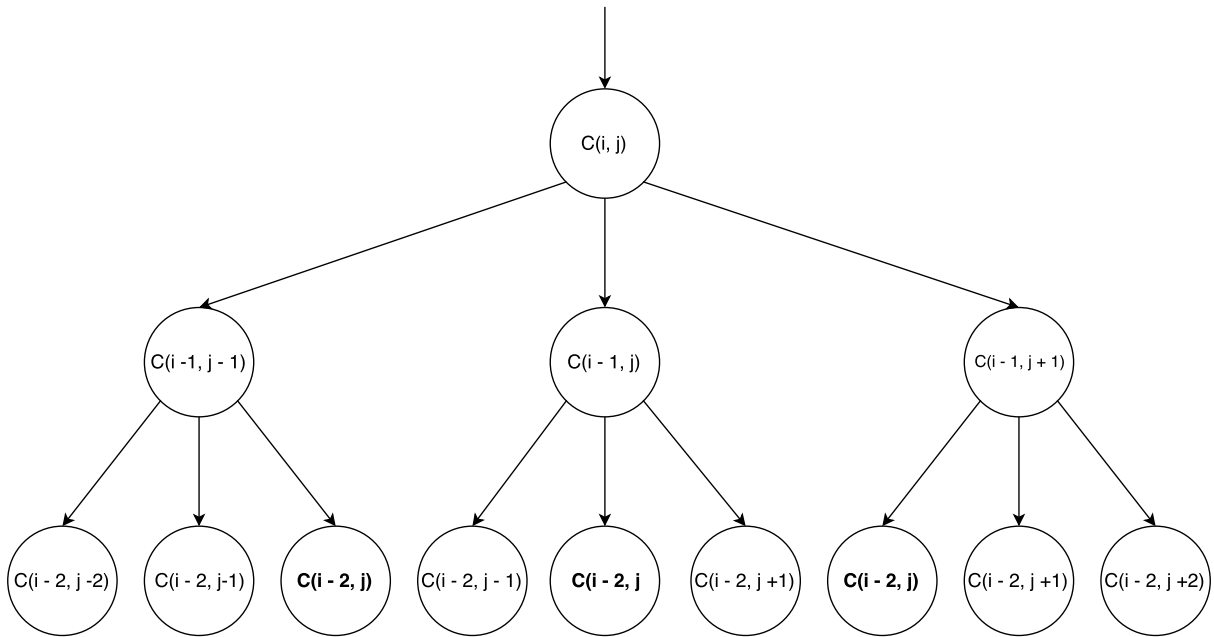


FIGURE 1 – Graphe des appels récursifs

On peut voir dans la figure 1 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)

$\text{COST}(\text{energy})$

```

1  for  $j = 1$  to  $\text{width}$ 
2     $\text{cost}[1][j] = \text{energy}[1][j]$ 
3  for  $i = 2$  to  $\text{height}$ 
4    for  $j = 1$  to  $\text{width}$ 
5      if  $j - 1 > 1$ 
6         $\text{left} = \text{cost}[i - 1][j - 1]$ 
7       $\text{mid} = \text{cost}[i - 1][j]$ 
8      if  $j + 1 < \text{width}$ 
9         $\text{right} = \text{cost}[i - 1][j + 1]$ 
10     // Si left ou right n'est pas défini, on n'en tient pas compte
11      $\text{cost}[i][j] = \text{energy}[i][j] + \min(\text{left}, \text{mid}, \text{right})$ 
12  return  $\text{cost}$ 
  
```

energy est un tableau de taille $\text{height} * \text{width}$ 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 $k > 20\% * m$.