University of Liège

Programmation avancée

INFO2050

Projet 3 | Mise en page automatique d'une image

Julien Gustin, Ilias Sebati

May 1, 2020

1 Résolution par recherche exhaustive

Pour générer tout les cas possibles en sachant que chaques points se divisent par après en trois points différents et que la première ligne (la derniere dans notre cas) ne compte pas. Nous avons ainsi $\mathcal{O}(3^{n-1})$, où le '3' provient des trois choix possibles. Cependant toutes ces cases-ci doivent etre traversés ainsi on doit multiplier cette complexité par la hauteur de l'image qui correspond à la hauteur max d'un sillon.
Ce qui nous donne $\mathcal{O}(3n^{n-1})$ comme complexité de cette méthode.

Ainsi on peut conclure que la méthode *brute-force* n'est pas la meilleure dans ce cas-ci vu qu'elle mène à une complexité exponentielle par rapport à la hauteur de l'image.

2 Formulation mathématique de C

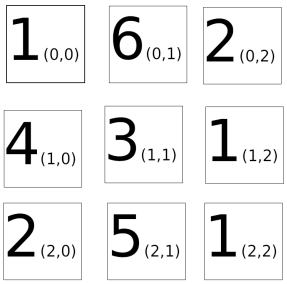
$$\text{C(i, j)} = \begin{cases} E(i,j)\} \ \ si\ i == 1 \\ +\infty\ si\ j == 0\ ou\ j > m - k \\ E(i,j) + min(\begin{cases} C(i-1,j-1)\ \} \\ C(i-1,j)\ \} \\ C(i-1,j+1)\} \end{cases} \ ) \ \ Sinon \end{cases}$$

Dans le code lui meme la fonction récursive est un peu différente, nous ne retournons pas le cout du sillon ou l'énérgie du pixel, mais le sillon associé.
Cela dans un but pratique et d'optimisation. Cependant l'idée de la formulation mathématique est belle et bien utilisée et pour acceder au cout d'un sillon ou l'énérgie associée au pixel il suffit de déréférencer l'adresse.

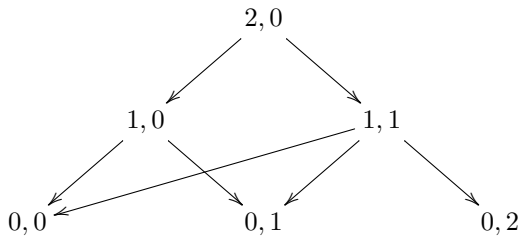
A la place de renvoyer $+\infty$ je renvois NULL, dans les deux cas c'est pour dire que ce sillon a dépacé le borne et qu'on doit donc s'arreter là.

3 Exemple de graphe des appels récursifs

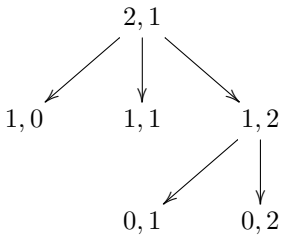
Figure 1: Exemple d'image, l'énergie du pixel (i, j) est écrit en grands



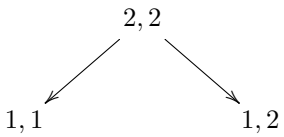
Lors de la première itération, donc lors du calcul du meilleur sillon de (2,0) aucun points n'a été préalablement calculé, la récursivité vaspasser par tout les points possibles en évitant les bornes. Lors de la remontée récursive, les couts des sillons aux points (2,0), (1,0) et (1,1) sont déjà calculés ainsi que l'énérgie de chacun des points.



Deuxième itération, nous passons à la case du milieu, (2,1) comme les couts du meilleur sillon de (1,0) et (1,1) ont déjà été calculés préalablement la fonction n'a qu'a nous renvoyer ce qui à déjà été calculé. Cependant, (1,2) n'étant encore jamais été visité, va regarder les pixels du dessus voisins pour prendre le plus petits des 2 (pas 3 vu que nous somme près d'un bord).



Derniere itération, vu que nous connaissons déjà le meilleurs chemins de (1,1) et (1,2) nous n'avons pas besoin de les recalculer.



4 Pseudo code : calcul le cout du sillon optimal

C(i, j, TS, I, k)

```
1  if j == 0 ∨ j > I.width-k
2        return NIL
3
4  if TS[i][j].nergie == -1 // l'énergie de ce pixel n'a pas encore été caculerée
5        TS[i][j].nergie == énergie(i,j, TS, I, k)
6
7  if i == 1 // cas de base
8        TS[i][j].cout_cumule = TS[i][j].nergie
9        return TS[i][j]
10
11 if TS[i][j].next != NIL // Le chemin du sillon optimal à ce point à déjà été calculé
12       return TS[i][j]
13
14 s = "Sillon"
15 s = min(min(C(i-1, j-1, TS, I, k), C(i-1, j, TS, I, k)),C(i-1, j+1, TS, I, k))
16 // on prend le sillon avec le cout cumulé le moins élévé
17 TS[i][j].next = s // on lie de sillon entre eux pour crée une liste liée de sillon
18 TS[i][j].cout_cumule = TS[i][j].energie + s.cout_cumule
19 return TS[i][j]
```

5 Pseudo code : renvoit de l'image avec réduction de k pixels

reduceImageWidth(I, k)

```
1  TS = "Crée un tableau de sillon de taille image.width * image.height
   dont chacune des case correspond au pixel correspondant de I"
2  for i = 0 to k
3        calcul_cout(I, i, TS)
4        reset_tabSillon(TS, I, i)
5  image2 = "Crée une image à partir de TS, donc avec k sillon supprimé"
6  libere_tab_sillon(TS, I)
7  return image2
```

calcul_cout(I, i, TS)

```
1  tmp = "Un sillon mis à NIL"
2  min = "Un sillon mis à NIL"
3  for j = 1 to I.width-k
4        tmp = C(I.height, j, TS, I, k)
5  if min == NIL or tmp.cout_cumule < min.cout_cumule
6        min = tmp
7  while min != NIL
8        min.nergie = -2.0
9        min = min.next
```

```
RESET_TABSILLON(TS, I, k)
 1  // energie {-1 = energie doit etre recalculée, -2 = sillon à supprimer, -3 = sillon à écraser }
 2  for i = 1 to I.height
 3      for j = 1 to I.width - k
 4          TS[i][j].next = NIL
 5          TS[i][j.cout_cumule = 0
 6          if TS[i][j].energie != -2
 7              if j >1 and TS[i][j − 1].energie == -3
 8                  TS[i][j − 1].energie = TS[i][j].energie
 9                  TS[i][j − 1].pixel = TS[i][j].pixel
10                  TS[i][j].energie = -3
11          else
12              TS[i][j].energie = -3
13              if j < I.width-k
14                  TS[i][j + 1].energie = -1
15              if j > 1
16                  TS[i][j − 1].energie = -1
```

6 Complexité

6.1 Temps

Pour l'initialisation de notre matrice de SILLON, nous avons $\mathcal{O}(n*m)$ ensuite cette boucle ci (voir boucle 1) parcourt toute la matrice en retirant k de *width* à chaque fois qu'on appelle cette fonction, ce qui donne $\sum_{i=0}^{k} \mathcal{O}(n*(m-i))$ que nous bornons à $\mathcal{O}(n*m*k)$ dans cette meme fonction nous parcourons le sillon en hauteur pour mettre l'energie de chacun de ces pixels à -2 pour pouvoirs reconnaitre le sillon qui doit etre supprimé ainsi $\mathcal{O}(n)$

Listing 1: Boucle 1

```
for(size_t j = 0; j < image->width-k; j++){
  tmp = C(image->height-1, j, tab_sillon, image, k);

  if(min == NULL || tmp->cout_cumule < min->cout_cumule)
    min = tmp;
  }
```

Et pour finir, la dernière fonction appelée k fois : RESET_TABSILLON qui supprime le sillon tout en reinitialisant certains champs de la structure, le parcours de la matrice nous donne une complexité de $\mathcal{O}(n*m)$. En faisant l'addition de toutes ces complexités nous pouvons borner celle de REDUCEIMAGEWIDTH à $\mathcal{O}(n*m*k)$

6.2 Espace

Pour la complexité en espace c'est très simple, dès le début du programme nous créons un tableau de SILLON de la taille de l'image originales . Ainsi nous avons une complexité de $\mathcal{O}(n*m)$

Listing 2: structure Sillon

```
typedef struct sillon_t SILLON;

struct sillon_t{
  SILLON *next;
  float energie;
  float cout_cumule;

  PNMPixel pixel;
};
```