Modélisation

emmanuel.jeandel@loria.fr

Le devoir doit être effectué par groupe de 1 *à* 2 *personnes, chaque personne appartenant à exactement un groupe.*

Travail à effectuer

Description

Le but est de réaliser un logiciel permettant de diminuer la taille d'une image en supprimant progressivement des colonnes "inutiles", comme présenté en TD.

Le travail est en deux parties :

— La première partie consiste à produire un logiciel qui enlève 50 pixels par ligne, en enlevant colonne par colonne.

— La seconde partie consiste à produire un logiciel qui enlève 50 pixels par ligne, en enlevant deux colonnes par deux colonnes.

— Chacune des deux parties peut être accompagnée d'améliorations, décrites dans ce document.

L'intérêt de la deuxième partie est purement théorique. La plupart des logiciels implémentent des variantes de la première méthode. L'algorithme mis en place dans la deuxième partie est bien plus compliqué, et prend plus de temps pour un résultat assez similaire.

Rendu

La première partie est à rendre pour le 2 février. Elle doit être constituée des éléments suivants :

- Le logiciel demandé. Le logiciel doit être programmé en Java et utiliser les classes Java fournies. Il doit fonctionner sur une machine Linux, distribution Debian 8, disposant de 4 processeurs Intel i5 à 3.20 Ghz et où les paquets `openjdk-8-jdk` et `openjdk-8-jre` sont installés.
- Un rapport (un `.txt` est suffisant) expliquant
  — Comment la répartition du travail s'est effectuée entre les différents membres d'un même groupe. Cette partie n'est facultative que pour les groupes d'une personne et sera prise en compte dans la notation.
  — Une description de l'utilisation du logiciel, et des modifications apportées par rapport au sujet.
- Deux images au format PGM, la deuxième correspondant au résultat de l'utilisation du logiciel sur la première. L'image doit être différente des images fournies par l'enseignant.

La deuxième partie est à rendre pour le 2 mars. Elle doit être constituée des éléments suivants :

- Le logiciel demandé (mêmes remarques que ci-dessus).
- Un rapport (mêmes remarques que ci-dessus). Ce rapport doit en particulier mentionner les améliorations effectuées.
- Deux images au format PGM (mêmes remarques que ci-dessus)

Le projet étant un projet de modélisation et non pas de développement logiciel, la majeure partie de la notation portera sur l'implémentation correcte des deux algorithmes. Un logiciel en mode texte est amplement suffisant. Pour obtenir une note conséquente, il est nécessaire de réaliser une ou plusieurs des améliorations proposées.

1 Première partie

Le fichier `SeamCarving` contient une fonction permettant de lire un fichier au format pgm. Dans ce format, chacun des pixels est représenté par un niveau de gris, entre 0 et 255.

Q 1) Ecrire une fonction `writepgm(int[][] image, String filename)` qui écrit un fichier pgm.

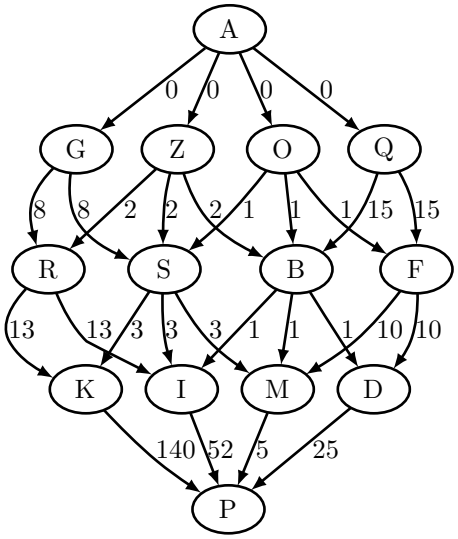
Q 2) Ecrire une fonction `int[][] interest (int[][] image)` qui prend une image et qui renvoie un tableau de la même taille, contenant, pour chaque pixel, son facteur d'intérêt. Le facteur d'intérêt est défini comme suit :

- Le facteur d'intérêt est la différence (en valeur absolue) entre la valeur du pixel, et la moyenne des valeurs de ses voisins de gauche et de droite
- Si un pixel n'a pas de voisin de droite (pixels au bout de la ligne), c'est la différence (en valeur absolue) entre le pixel et le pixel précédent.
- Si un pixel n'a pas de voisin de gauche (pixels en début de ligne), c'est la différence (en valeur absolue) entre le pixel et le pixel suivant.

Par exemple, voici une image 4×3, et le tableau intérêt correspondant :

Image				Intérêt			
3	11	24	39	8	2	1	15
8	21	29	39	13	3	1	10
200	60	25	0	140	52	5	25

Q 3) Ecrire une fonction `Graph tograph(int[][] itr)` qui crée le graphe correspondant au tableau `itr`.
Voici par exemple le graphe correspondant au tableau précédent :



Q 4) Ecrire une fonction `Dijkstra(Graph g, int s, int t)` qui renvoie le plus court chemin entre les sommets s et t dans le graphe g.

Note : On a vu en TD que le graphe en question est un DAG, et qu'on peut donc faire mieux que Dijkstra. Cependant il sera nécessaire pour la deuxième partie d'avoir une implémentation de Dijkstra.

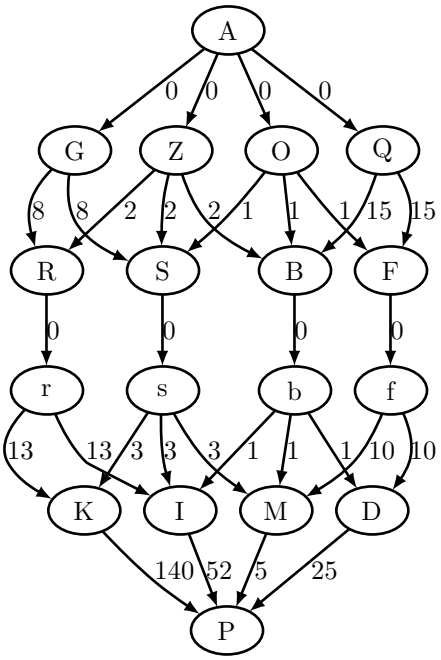
Q 5) Utiliser les fonctions précédentes pour écrire le logiciel demandé.

2 Deuxième partie

Dans cette partie, on va chercher à supprimer deux pixels par ligne d'un coup en cherchant deux plus courts chemins. L'algorithme donné ici est due à Suurballe. On prouvera la validité de cet algorithme à la fin du cours.

La première chose à faire est de transformer le graphe pour remplacer tous les sommets correspondant aux lignes intérieures (toutes les lignes sauf la première et la dernière) par deux sommets reliés par une arête de poids nul.

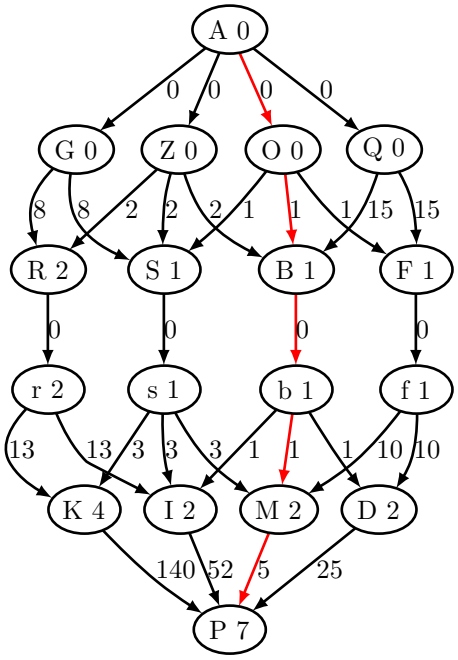
Dans l'exemple on obtient le graphe suivant :



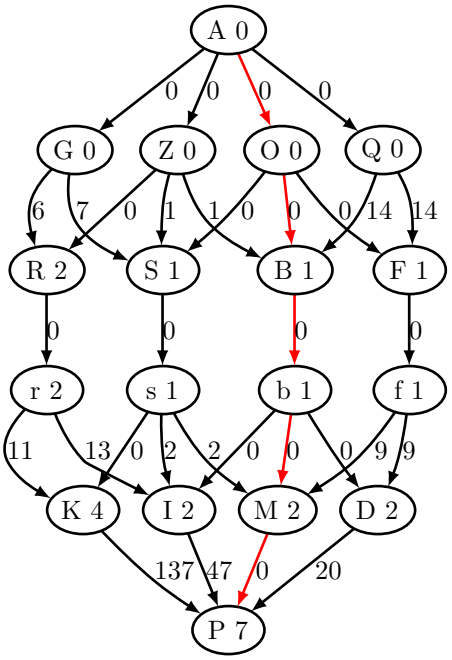
(On pourrait aussi dupliquer les sommets G, Z, O, Q et dupliquer les sommets K, I, M, D, mais ce n'est pas nécessaire)

Trouver les deux chemins les plus courts dans le graphe de départ sans sommets en commun revient dans ce nouveau graphe à trouver les deux chemins les plus courts sans arêtes en commun.

Pour ce faire, on commence par calculer le chemin de coût minimal entre le sommet de départ (A dans l'exemple) et tous les autres sommets. On marque en rouge sur le dessin les sommets faisant partie du plus court chemin entre le sommet de départ et le sommets d'arrivée (P). La valeur du chemin de coût minimal est indiquée sur chaque sommet.

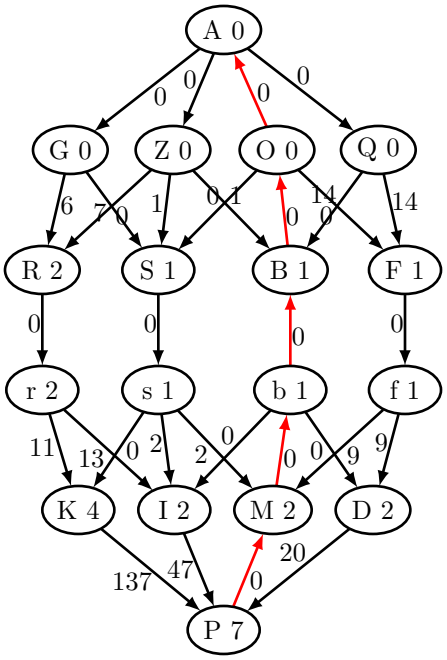


On ajoute alors à chaque arête (u, v) la différence entre le coût du plus court chemin arrivant à u et celui arrivant à v. Par exemple, pour l'arête entre s et M, qui valait 3, on ajoute $1 - 2 = -1$, et elle passe donc à 2 :

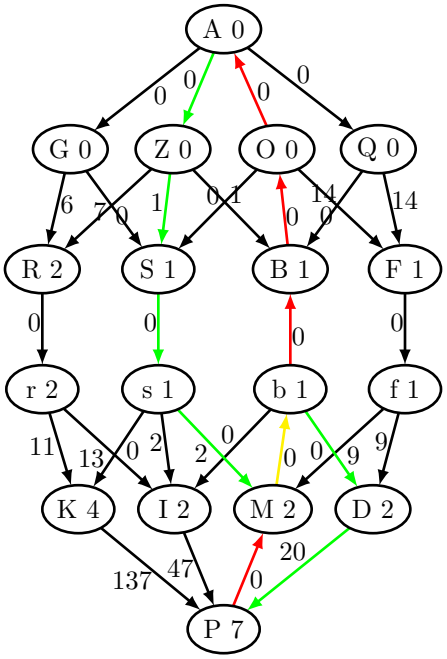


(Noter que tous les coefficients restent positifs)

On inverse ensuite les arêtes du plus court chemin :

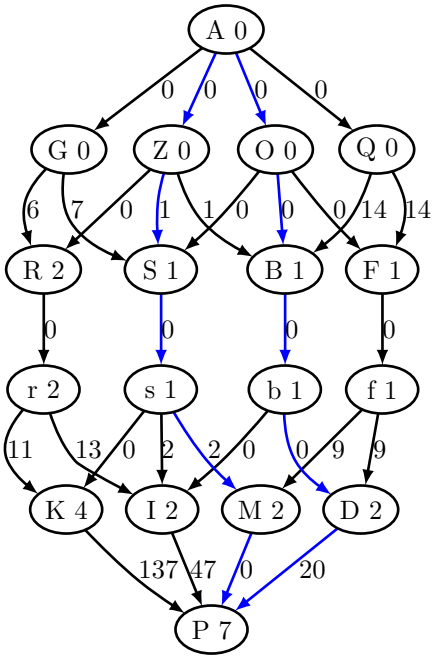


On cherche ensuite le chemin le plus court dans ce nouveau graphe (qui n'est plus un DAG, mais où les poids sont tous positifs, donc on peut utiliser Dijkstra) :



(Les arêtes en jaunes correspondent aux arêtes qui sont rouges et vertes)

Finalement, les deux chemins qui nous intéressent sont ceux dans le graphe de départ correspondant aux arêtes qui sont dans l'un des deux plus courts chemins, mais pas dans les deux (donc on enlève les arêtes jaunes) :



Q 1) Modifier la fonction `tograph` de la première partie pour qu'elle produise un graphe comme représenté ci-haut. (Note : si c'est plus simple à programmer, l'algorithme fonctionne également si on dédouble aussi les autres sommets)

Q 2) Ecrire une fonction `twopath(Graph g, int s, int t)` qui calcule les deux plus courts de s à t passant par des arêtes différentes en utilisant la méthode précédente.

3 Améliorations possibles

3.1 Fonction d'énergie avant

On peut obtenir un meilleur résultat en changeant la façon de construire le graphe. Au lieu que les trois arêtes qui partent d'un sommet aient le même coût, on va leur mettre des coûts différents. On note $M[i, j]$ l'intensité (le niveau de gris) du pixel de coordonnées (i, j). On pose $M[i, j] = 0$ si le pixel n'existe pas (si on déborde sur la gauche ou la droite)

On relie alors :

— Le sommet (i, j) au sommet $(i + 1, j)$ avec comme coût $|M[i, j + 1] - M[i, j - 1]|$

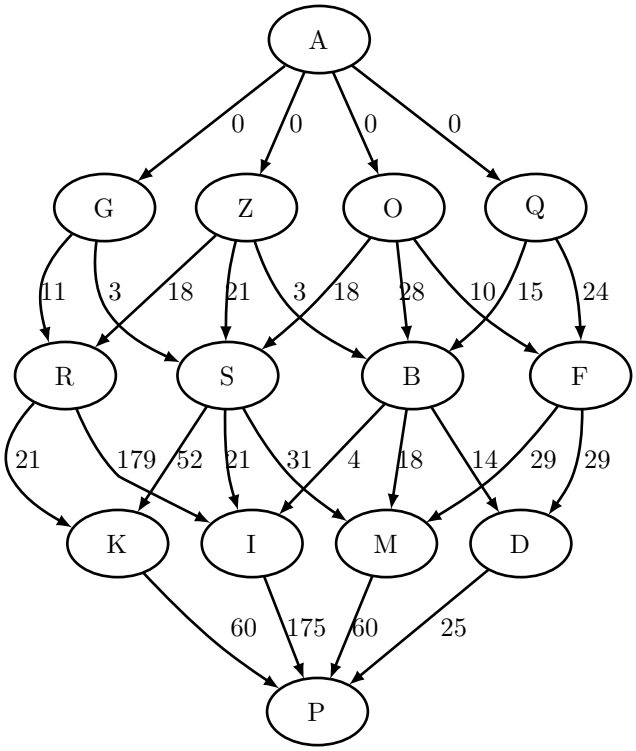
— Le sommet (i, j) au sommet $(i + 1, j + 1)$ avec comme coût $|M[i, j + 1] - M[i + 1, j]|$

— Le sommet (i, j) au sommet $(i + 1, j - 1)$ avec comme coût $|M[i, j - 1] - M[i + 1, j]|$

Si on reprend l'exemple de l'image suivante :

3	11	24	39
8	21	29	39
200	60	25	0

on doit obtenir le résultat :



Par exemple, pour le sommet Z (qui correspond au deuxième pixel en haut, de valeur 11)

— L'arête vers R a comme coefficient $|21 - 3|$

— L'arête vers S a comme coefficient $|24 - 3|$

— L'arête vers B a comme coefficient $|24 - 21|$

Pour les arêtes partant des sommets de la dernière ligne, il s'agit de la différence entre le pixel précédent et le pixel suivant. Pour le sommet I, on fait donc $|200 - 25|$.

On trouvera une comparaison des résultats avec les deux méthodes à la fin de ce document.

3.2 Autres modifications possibles

— Implémentation de l'algorithme vu en TD à la place de Dijkstra ;
— Suppression de lignes et de colonnes ;
— Gestion de la couleur (chercher sur Internet le format des fichiers ppm P3) ;
— Gestion de pixels qu'il ne faut surtout pas supprimer ;
— Gestion de pixels qu'il faut forcément supprimer.
— Augmentation de la taille d'une image.

4 Fichiers fournis

4.1 Images

Les images pour tester sont au format pgm P2. Si on dispose d'un fichier jpeg,png, ou autre, on peut le convertir en format pgm P2 avec la commande :

```
convert -compress none toto.jpg toto.pgm
```

On peut afficher un fichier pgm en utilisant la commande `display`, en utilisant un gestionnaire de fichiers ou un navigateur.

Les images `ex1.pgm` et `ex2.pgm` sont dans le domaine public. L'image `ex3.pgm` a été fournie par Alexandre Buisse.

L'image de format 4x3 fournie correspond à l'exemple donné dans l'énoncé.

4.2 Graph

La classe `Graph` permet de représenter des graphes dont les sommets sont numérotés par des entiers. Le constructeur `Graph(int N)` crée un graphe dont les sommets sont les entiers entre 0 et $N-1$. On peut ensuite ajouter des arêtes au graphe en utilisant la fonction `addEdge(Edge e)`. Les arêtes sont définies dans la classe `Edge` et ont une source (`e.from`), une destination (`e.to`) et un coût (`e.cost`).

La classe `Graph` dispose d'une fonction `writedot` qui permet de transformer le graphe en un fichier `.dot`. Il est ensuite possible de visualiser le fichier .dot en utilisant la bibliothèque `graphviz`, ou un site web comme `http://sandbox.kidstrythisathome.com/erdos/`.

Une utilisation de la classe Graphe est fournie dans la classe `Test`. Elle construit un graphe sous forme de grille et lance un parcours en profondeur sur ce graphe.

4.3 Heap

La classe Heap est une implémentation d'une file de priorité en Java, qui peut stocker des entiers ayant des priorités.

Voici les différentes fonctions :

- Le constructeur `Heap(int N)` crée une file de priorité contenant initialement tous les entiers de 0 à $n-1$, chacun avec priorité $+\infty$.
- La fonction `int pop()` (qu'on a appelé `extractMin` en cours) supprime de la file de la priorité l'élément de priorité la plus faible et le renvoie. On peut trouver sa priorité en utilisant la fonction `priority(int x)` (Note : la fonction `priority(x)` peut être appelée même si l'élément x n'est plus dans la file)
- La fonction `decreaseKey(int x, int p)` change la priorité de l'élément x à la valeur p. C'est une erreur d'appeler cette fonction avec une valeur de p plus grande que la priorité de l'élément x.
- La fonction `boolean isEmpty()` permet de savoir si la file de priorité est vide.

Notez qu'il n'y a aucune fonction pour ajouter un élément à la file de priorité : Ils sont tous ajoutés à l'initialisation.

5 Exemples

FIGURE 1 – Fichier initial

FIGURE 2 – Résultat en utilisant la méthode de la première partie puis en utilisant la méthode par fonction d'énergie avant

FIGURE 3 – On peut aussi utiliser des pandas roux

FIGURE 4 – Paysage