Graphes II et réseaux

Projet Binarisation d'images

Thomas Lapierre - Lucas Lelièvre
Master Informatique

Utilisation et exécution du projet

1. Se placer dans la racine du projet : `cd ImageSegmentation/`
2. Exécuter la commande suivante : `java -jar imageSegmentation.jar <sourceDuFichierTxt> <parametre> (--info)`

`<parametre>` : Obligatoire ✅

`--info` : Pas obligatoire ❌

`<parametre>`	Description
-a	Affiche les pixels présents dans les plans un et deux et renvoie la représentation graphique de la segmentation de l'image.
-p	Affiche uniquement la représentation graphique de la segmentation de l'image.

`--info` : Affichage des informations du parsing du fichier texte d'entrée

Exemple 1

```
java -jar imageSegmentation.jar data/deuxlunes.txt -p
```

Résultat

```
FLOT MAXIMUM
2555

AFFICHAGE DES PLANS
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 - - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 - - - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 - - - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 - - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 - - - - - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 - - - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - - - 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - - - - 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - - - - 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - - - - 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - - 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - - - - - 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Exemple 2

```
java -jar imageSegmentation.jar data/demo_sujet.txt -a --info
```

Résultat

```
AFFICHAGE PROBA A
1 1 1 1
1 19 17 1
1 20 20 1
1 1 1 1

AFFICHAGE PROBA B
20 18 18 20
20 1 1 20
20 1 1 18
18 18 20 20

AFFICHAGE PENALITES HORIZONTALES
15 19 17
0 18 5
0 20 5
19 17 19

AFFICHAGE PENALITES VERTICALES
20 8 10 19
20 20 18 17
21 5 0 17

FLOT MAXIMUM
49
PIXEL PREMIER PLAN
6 7 10 11
PIXEL DEUXIEME PLAN
1 2 3 4 5 8 9 12 13 14 15 16
AFFICHAGE DES PLANS
- - - -
- O O -
- O O -
- - - -
```

A noter : Pour éviter tout problème, un dossier "data" est déjà présent. Le plus simple est donc de mettre les fichiers que l'on souhaite traiter dedans et ainsi `<sourceDuFichierTxt>` aura pour valeur : `data/<nomDeVotreFichier>`

Réponses aux questions

1) Q est la somme totale des valeurs “a” et “b” pour chaque sommet du graphe, soit :

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{m} (a_{ij} + b_{ij})$$

Ce qui équivaut à la somme des valeurs “a” et “b” des sommets dans A et des valeurs “a” et “b” des sommets dans B :

$$Q = \sum_{(i,j)\in A} a_{ij} + \sum_{(i,j)\in A} b_{ij} + \sum_{(k,l)\in B} a_{kl} + \sum_{(k,l)\in B} b_{kl}$$

$$Q = \left(\sum_{(i,j)\in A} a_{ij} + \sum_{(k,l)\in B} b_{kl} \right) + \left(\sum_{(i,j)\in A} b_{ij} + \sum_{(k,l)\in B} a_{kl} \right)$$

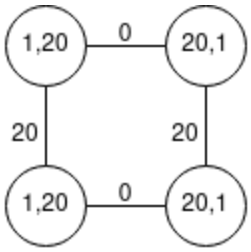
Ceci étant une équation, maximiser la partie de gauche de l'addition revient à en minimiser la partie de droite.

$\sum_{(i,j)\in A,(k,l)\in B \text{ voisins}} p_{ijkl}$ Cette partie de la formule représente la somme des pénalités des sommets voisins d'ensemble distincts, nous l'appellerons psum. Dans la formule q(A,B) l, psum est soustraite de la somme des valeurs a et b des sommets, ainsi, pour que q(A,B) soit maximisée, psum doit être la plus petite possible. Dans q'(A,B), psum est ajoutée à la somme des valeurs a et b des sommets, ainsi pour que q'(A,B) soit minimisée, psum doit aussi être la plus petite possible.

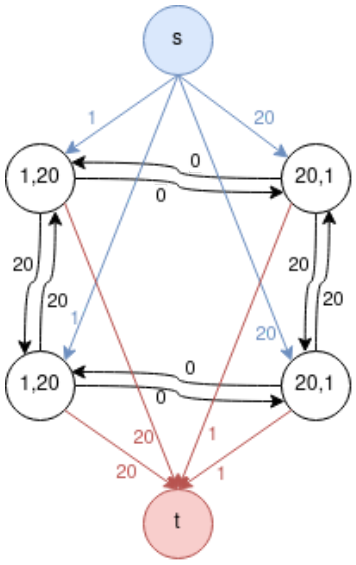
Les différentes parties des deux formules sont équivalentes, et maximiser q(A,B) revient bien à minimiser q'(A,B).

2) Pour une image donnée, on construit un réseau de transport en 4 étapes.
   a) Chaque pixels de l'image est un sommet.
   b) On ajoute une source s et un puit t au graphe.
   c) Pour chaque sommet i du graphe, on ajoute deux arc : un arc de s à i de capacité égale à la valeur “a” de i, et un arc de i a t de capacité égale à la valeur b de i.
   d) Pour chaque pixels i et j voisins dans l'image, on ajoute les arcs (i, j) et (j, i) de capacités égales à la pénalité de i et j.

Image en entrée



Graphe de transport généré



3) Lorsqu'on cherche la coupe de capacité minimum (A U {s}, B U {t}), on cherche une coupe telle que 3 choses soit minimisées :
   - La somme des capacités des arcs de la source s au sommets de B
   - La somme des capacités des arcs des sommets dans A au puit t
   - La somme des capacités des arcs des sommets de A aux sommets de B.

   Cela revient à minimiser la formule suivante, qui est la formule q’(A,B) :

$$\sum_{(i,j)\in A} b_{ij} + \sum_{(k,l)\in B} a_{kl} + \sum_{(i,j)\in A,(k,l)\in B \text{ voisins}} p_{ijkl}$$

   Ainsi, si la pair n'est pas une coupe de capacité minimum, alors la formule q'(A,B) n'est pas minimum, et (A,B) n'est pas une solution de BinIm.

Algorithme général

```
fonction main() :
        informationsDeLImage = parserTexte(adresseFichierTxt)
        graphe = creerGraphe(informationsDeLImage)
        graphe.calculerFlotMaximum()
        graphe.afficherPlans()
FIN
```

Le programme va prendre une adresse de fichier texte en paramètre. Ensuite, il va convertir le fichier texte en classe objet “InformationImage”. Le programme va alors créer le graphe à partir de l’objet “InformationImage”. Une fois le graphe obtenu, on va alors lancer l’algorithme des préflots afin d’obtenir le flot maximum. Une fois le graphe mis à jour, on va calculer la coupe minimum dans ce graphe. Enfin, on affiche le plan pour avoir un rendu graphique.

Fonctions auxiliaires

ConstructionReseau

Détail de la fonction

```
sommets = []

fonction constructionReseau(ImageInfo info):
        nbPixel = nbPixelColonne * nbPixelLargeur
        source = nouveau sommet()
        puit = nouveau sommet()

        POUR i allant de 1 à nbPixel FAIRE
                s = nouveau sommet()
                sommets.ajouter(s)
        FIN POUR

        POUR i allant de 1 à nbPixel FAIRE
                ajouterArc(source,sommets[i])
                ajouterArc(sommets[i],puit)
                SI le pixel possède un pixel droit FAIRE
                        ajouterArc(sommets[i],sommets[i+1])
                FIN SI
                SI le pixel possède un pixel gauche FAIRE
                        ajouterArc(sommets[i],sommets[i-1])
                FIN SI
                SI le pixel possède un pixel au dessus FAIRE
                        ajouterArc(sommets[i],sommets[i-nbPixelLargeur])
                FIN SI
                SI le pixel possède un pixel en dessous
                        ajouterArc(sommets[i],i+nbPixelLargeur)
                FIN SI
        FIN POUR
FIN
```

Explication

La fonction va créer un sommet source et un sommet puit. Elle va ensuite créer un sommet pour chaque pixel de l'image à segmenter. Ensuite, pour chaque pixel p (devenu sommet) on va créer un arc du sommet source vers ce pixel et du pixel au sommet puit. On va également relier, avec un arc, p vers son voisin de droite, de gauche, du haut et du bas, le tout uniquement si le voisin existe.

Complexité

La complexité est assez simple à calculer puisqu'en réalité nous allons juste parcourir l'ensemble des pixels de notre image.

La complexité d'ajout dans une ArrayList est en O(1) (temps constant), l'ajout dans une ArrayList à la position i est en O(n).

La complexité du parcours de tous les pixels est de O(n) avec n = nombre de pixels dans l'image.

Ainsi la complexité est : O(n) avec n = nombre de pixels dans l'image.

CalculCoupeMin

Détail de la fonction

```
fonction calculCoupeMin(): liste de sommets du premier plan
        ensemble = {}
        parcoursEnProfondeur(graphe,ensemble)
        ensemble.supprimer(sommetSource)
        ensemble.supprimer(sommetPuit)
        RETOURNER ensemble
FIN
```

Explication

La fonction va parcourir l'ensemble des sommets accessibles par des arcs qui n'ont pas un flot égal à la capacité. Autrement dit, elle va effectuer un parcours en profondeur et ajouter tous les sommets différents parcourus dans une liste. Cette liste s'agit de la coupe minimale.

Complexité

La complexité au pire à lieu quand la coupe minimum est en faite l'intégralité du graphe. Ainsi, dans ce cas nous devrons parcourir l'ensemble du graphe. La complexité d'un parcours en profondeur d'un graphe est la suivante :

O(|S| + |A|) avec S qui correspond aux sommets et A aux arêtes.

Ensuite, on va ajouter le sommet à chaque fois qu'il s'agit d'un nouveau. L'ajout dans un HashSet se fait en O(1) et le "contains" s'exécute en O(1).

Enfin on supprime le sommet source et le sommet puit qui se fait en O(2*1).

On obtient la complexité suivante : O((|S| + |A|) * (1 + 1) + 2) → O(|S| + |A|)

ResoudreBinIm

Détail de la fonction

```
fonction resoudreBinMin(): liste de listes de sommets du plan 1 et 2
        plans=[]
        premierPlan = calculCoupeMin()
        deuxiemePlan = sommets
        deuxiemePlan.supprimer(premierPlan)
        deuxiemePlan.supprimer(sommetSource)
        deuxiemePlan.supprimer(sommetPuit)
        plans.ajouter(premierPlan)
        plans.ajouter(deuxiemePlan)
        RETOURNER plans
FIN
```

Explication

Cette méthode va créer un ensemble de sommets en appelant la méthode calculCoupeMin. Cet ensemble correspondra au premier plan. Enfin, pour déterminer le second ensemble, elle va créer un ensemble égal à tous les sommets du graphe et va retirer ceux du premier plan. Ainsi, on obtient le second plan. La méthode retourne une liste contenant deux listes : le premier ensemble (premier plan) et le second.

Complexité

Pour cette fonction on appelle la méthode calculCoupeMin dont la complexité est : O(|S| + |A|)

Ensuite on va ajouter un élément dans l'ArrayList ➜ O(1)

Par la suite, on va initialiser un HashSet par l'ArrayList de sommets ➜ O(|S|) (car on a pas de doublon dans la liste de sommets sinon O(|S|^2 si la version de Java <8 sinon O(|S| log |S|))

On va ensuite faire deux suppressions dans un HashSet ➜ O(1)

On va faire un removeAll dans un HashSet ➜ O(|S|) car au pire on supprime tous les sommets

Enfin, on ajoute l'HashSet dans l'ArrayList ➜ O(1)

On obtient la complexité suivante : O(|S| + |A|) *On évite ainsi du |S|^2*

Conclusion

Pour conclure, ce projet a permis de mettre en pratique une notion vue en cours : la maximisation de flot. On a pu voir un cas concret dans lequel cette notion serait utile. Grâce à cette technique de préflot, nous avons réussi à implémenter un module de segmentation d'image en Java. Ce projet nous a permis de se rendre compte de l'importance des flots dans la vie de tous les jours. En effet, nous sommes confrontés au quotidien à plusieurs problèmes qui peuvent être résolus par le biais de cette méthode (GPS, industries). Enfin, ce projet nous a permis de se rendre compte de la performance des algorithmes que l'on peut connaître dans les applications déjà existantes. En effet, la méthode que nous avons implémentée est très vite inutilisable pour des photos dépassant le 150x150. Ainsi, les applications permettant de segmenter des images 4000x4000 en moins de 10 secondes sont des algorithmes précieux, performants et durs à écrire.