EISTI
Ecole Internationale des Sciences du Traitement de l'Information

Algorithmique et Conteneurs

Le conteneur Graphe

Utilité des graphes

- Comme les arbres, de nombreux systèmes sont naturellement des graphes : réseau de routes, WEB : réseau de machines, réseau de chemin de fer, …
- Problèmes de flot maximum (lien avec la loi de conservation de l'énergie : loi de Kirshoff).
- Processus stochastiques : Chaînes de Markov
- …

Rappels : Définitions d'un graphe

- Un graphe est une paire d'ensembles (S, A) où A est une famille de paires (orientées ou non) d'éléments de S.
- Dans le cas d'un graphe orienté, on utilisera le terme arc plutôt qu'arête.
- Le degré d'un sommet s, noté d(s), est le nombre d'arêtes incidentes à s, c'est à dire contenant s.
- Une chaîne est une suite finie de sommets (s_1, ...,s_n) reliés entre eux par une arête ((s_i ,s_{i+1}) est une arête).
- Un chemin, dans un graphe orienté, est une chaîne où les arêtes (s_i ,s_{i+1}) sont des arcs.
- Un cycle est une chaîne qui revient à son point de départ ($s_1 = s_n$).
- Un graphe G est dit connexe si pour toute paire de sommets(s,s') de G, il existe une chaîne de premier terme s et de dernier terme s' .
- Un graphe G est dit fortement connexe si toute paire de sommets (s,s') de G, il existe une chemin de premier terme s et de dernier terme s' et un autre chemin de premier terme s' et de dernier terme s .

Types de graphes

- Un graphe est complet si tout sommet est relié à tous les autres.
- Un graphe est trivial s'il n'est qu'un sommet isolé.
- Un graphe est k-régulier si tout sommet est de degré k.
- Un graphe G(S,A)est bipartites si
  - $S = S_1 \cup S_2 \;\; S_1 \cap S_2 = \varnothing$
  - $\forall\, s_1 \in S_1 \;\; \exists\, s_2 \in S_2 \,/\, (s_1, s_2) \in A$
- Un graphe est planaire si les arcs ne coupent pas

Parcours de graphes

- Comme pour les arbres (ou arborescences), on pourra faire des parcours en largeur ou en profondeur.
- La difficulté dans un parcours de graphe est de repasser par un sommet.
- Pour régler ce problème, on introduit une notion de marquage de sommet pour se souvenir du passage par un sommet donné
- Principes généraux du marquage :
  - Au début de l'algorithme, on démarque tous les sommets
  - Dès qu'on passe par un sommet, on le marque.
  - On n'étudie plus les sommets marqués

Parcours de graphes en largeur

```
Parcourir en Largeur G
    Démarquer tous les sommets de G
    Pour chaque sommet s de G
        Si s n'est pas marqué
            Créer un file f
            Enfiler s dans f
            Marquer s
            Tantque f n'est pas vide
                Récupérer s' la tête de f
                { Traiter s' }
                Enfiler dans f, tous les successeurs de s' non marqués
            Fin Tantque
        Fin Si
    Fin Pour
```

Parcours de graphes en profondeur

```
Parcourir en profondeur G
    Démarquer tous les sommets de G
    Pour chaque sommet s de G
        Si s n'est pas marqué
            Parcourir en profondeur G à partir de s
        Fin Si
    Fin Pour

Parcourir en profondeur G à partir de s
    Marquer s
    { Traiter s }
    Pour tous les sommets s' successeur de s
        Si s' n'est pas marqué Alors
            Parcourir en profondeur G à partir de s'
        Fin Si
    Fin Pour
```

Les type Arete et AreteValuee

- Le type Arete utilise le type Element
  Méthode de construction et d'initialisation
  - fonction creerArete(orig : Element, dest : Element) : Arete

  Méthodes d'accès
  - fonction recOrigine(ar : Arete) : Element
  - fonction recDestination(ar : Arete) : Element

- Le type AreteValuee utilise les types Arete, Element et Reel
  AreteValuee hérite de Arete // On récupère les méthodes d'accès
  Méthode de construction et d'initialisation
  - fonction creerAreteValuee(orig : Element, dest : Element, val : Reel) : AreteValuee

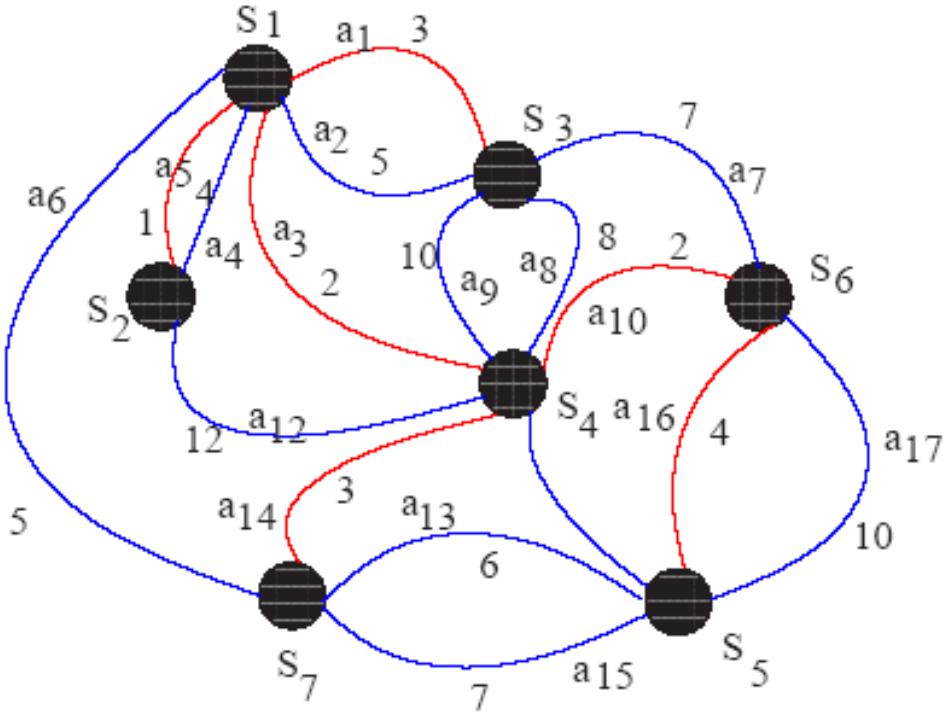
  Méthodes d'accès
  - Fonction recValuation(ar : Arete) : Reel

Méthodes de bases du type Graphe

- Ce type utilise les types Element et Arete (donc AreteValuee)
- Méthodes de construction et d'initialisation
  - fonction grapheVide() : Graphe
- Méthodes de mise à jour
  - fonction ajouterSommet(gr : Graphe, e : Element) : Graphe
  - fonction ajouterArete(gr : Graphe, a : Arete) : Graphe
  - fonction supprimerSommet(gr : Graphe, e : Element) : Graphe
  - fonction supprimerArete(gr : Graphe, a : Arete) : Graphe
- Méthodes d'accès
  - fonction recNbSommets(gr : Graphe) : Entier
  - Fonction recSommets(gr : Graphe) : Vecteur
  - Fonction recAretes(gr : Graphe) : Vecteur
  - fonction recSuccesseurs(gr : Graphe, e : Element) : Vecteur
  - fonction recPredecesseurs(gr : Graphe, e : Element) : Vecteur

Arbre couvrant de poids minimum

- Données : graphe valué, connexe et non orienté avec arêtes valuées (coûts de transport, probabilités, poids, longueurs)
- Objectif : trouver un arbre partiel de coût minimum avec les mêmes sommets que le graphe initial



Algorithme de Kruskal

Principes de l'algorithme

- Créer un graphe (sans arêtes) avec les n sommets du graphe connexe
- Choisir une arête de poids minimum
- Répéter la procédure parmi les arêtes restantes de poids minimum ne faisant pas de boucles avec les précédentes

Etapes de l'algorithme

Trier les arêtes par ordre croissant
Tantqu'on n'a pas sélectionné n − 1 arêtes Faire
    considérer la première non sélectionnée et non rejetée (ordre du tri)
    Si elle ne fait pas une boucle,
        la sélectionner
    Sinon
        la rejeter
Fin Tantque

Implémentation de l'algorithme de Kruskal

Kruskal de G // On note n le nombre de sommets

On crée un graphe G' avec les mêmes sommets que G mais sans arêtes

On associe un indice i à chaque sommet.

On trie les arêtes par ordre croissant de valuation

On construit un tableau cc de composantes connexes de G' indicés par les sommets

// Au départ cc(i) = i car il n'y a pas d'arêtes dans G' $\Rightarrow$ autant de composantes connexes que de sommets

Tant que le nombre d'arêtes sélectionnées n'est pas égale à (n-1)

prendre la prochaine arête // x et y sont les indices des sommets

Si cc(x) <> cc(y) Alors

Sélectionner cette arête et l'ajouter au graphe G'

Fusionner les composantes connexes de x et de y (*)

Fin Si

Retourner G'

(*) $\forall$ i $\in$ { 1, …, n} Si cc(i) = cc(y) alors cc(i) $\leftarrow$ cc(x)