Algorithmique des graphes

David Pichardie

20 Avril 2018

Bilan du CM8

- Plus courts chemins
  - Définitions
  - Algorithme ordinal (en avant/en arrière)
  - Algorithme de Dijkstra

Algorithme de Dijkstra

- on se restreint à un graphe sans poids négatifs

```
DIJKSTRA(G,s)=
  dist ⟵ [+oo, …, +oo ]
dist[s] ⟵ 0
S ⟵ {}
S’ ⟵ tous les sommets de G
tant que S’ non vide
   i ⟵ choisir i∈S’, tel que dist[i] minimal
   S’ ⟵ S’-{i}
   S ⟵ S+{i}
   pour tout j∈Adj[i]
        si dist[i]+w(i,j)<dist[j]
        alors
           dist[j] ⟵ dist[i]+w(i,j)
           pred[j] ⟵ i
```

Quel algorithme, que avons nous déjà rencontré, ressemble fortement à Dijkstra ?

Algorithme de Prim

- On colorie toutes les arêtes de G en gris. Puis on va progressivement colorier en noir les arêtes de ACM. A chaque étape, le sous-graphe noir est un arbre. On colorie en noir les sommets reliés par des arêtes noirs.

- On choisie un sommet de départ.

- On considère la coupure partitionnant les sommets noirs des autres. On choisi un arc traversant minimal pour cette coupure et on le colorie en noir. On colorie en noir le sommet associé qui ne l'était pas encore.

- On répète jusqu'à avoir colorié tous les sommets en noir.

Algorithme de Prim

G est non-orienté cette fois

```
Prim(G)=
  dist ⟵ [+oo, …, +oo ]
  dist[s] ⟵ 0
  S ⟵ {}
  S' ⟵ tous les sommets de G
  tant que S' non vide
     i ⟵ choisir i∈S', tel que dist[i] minimal
     S' ⟵ S'-{i}
     S ⟵ S+{i}
     pour tout j∈Adj [i]
          si j∈S' et w(i,j)<dist[j]
          alors
              dist[j] ⟵ w(i,j)
              pred[j] ⟵ i
```

Exercice : justifiez cette implémentation

Implémentation

Comment implémenter
efficacement ce choix ?

```
Prim(G)=
  …
     i ⟵ choisir i∈S', tel que dist[i] minimal
  …

DIJKSTRA(G,s)=
  …
     i ⟵ choisir i∈S', tel que dist[i] minimal
  …
```

File de priorité

- Collection d'éléments avec un poids
  - Insérer un élément
  - Supprimer l'élément de plus petit poids
  - Diminuer le poids d'un élément de la file
  - Créer une file vide

Algorithme de Dijkstra

- on se restreint à un graphe sans poids négatifs

```
DIJKSTRA(G,s)=
  S ⟵ {}
S’ ⟵ file de priorité avec tous les sommets de G au poids +oo
diminuerPoid(S’,s,0)
tant que S’ non vide
   i ⟵ extraireMin(S’) // choisir i∈S’ de poids minimal
  S ⟵ S+{i}
   pour tout j∈Adj[i]
        p_i ⟵ poidsCourant(S’,i)
        p_j ⟵ poidsCourant(S’,j)
        si p_i+w(i,j)<p_j
        alors
           diminuerPoid(S’,j,p_i+w(i,j))
           pred[j] ⟵ i
```

Comparaison des algorithmes de calcul de plus courts chemins à origine unique

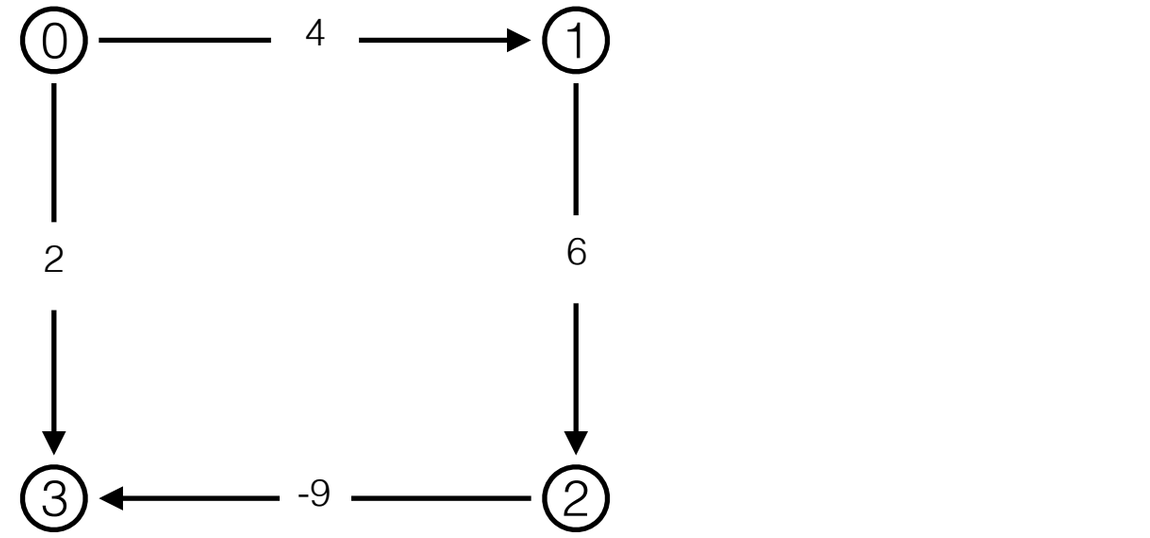
- Ordinal
  - pour les graphes acycliques
  - coût **O**(S+A)
- Dijkstra
  - pour les graphes avec des poids positifs
  - coût **O**(A·log(S) + S·log(S))

Avec une implémentation de file par tas binaire

appels à DiminuerPoids

appels à extraireMin

Dijkstra ne fonctionne pas avec les poids négatifs



Dijkstra pense que 0-3 est un plus court chemin de 0 à 3

Comparaison des algorithmes de calculs de plus courts chemins à origine unique

- Dans le cas général ?
  - décider si un chemin est un plus court chemin dans un graphe orient pondéré quelconque est NP-complet
    - on n'est même pas sûr qu'un plus court chemin existe
- Graphes sans cycles de poids négatif
  - on sait les détecter en coût **O**(A·S)
  - on sait y calculer des plus courts chemin en coût **O**(A·S) avec l'algorithme de Bellman-Ford

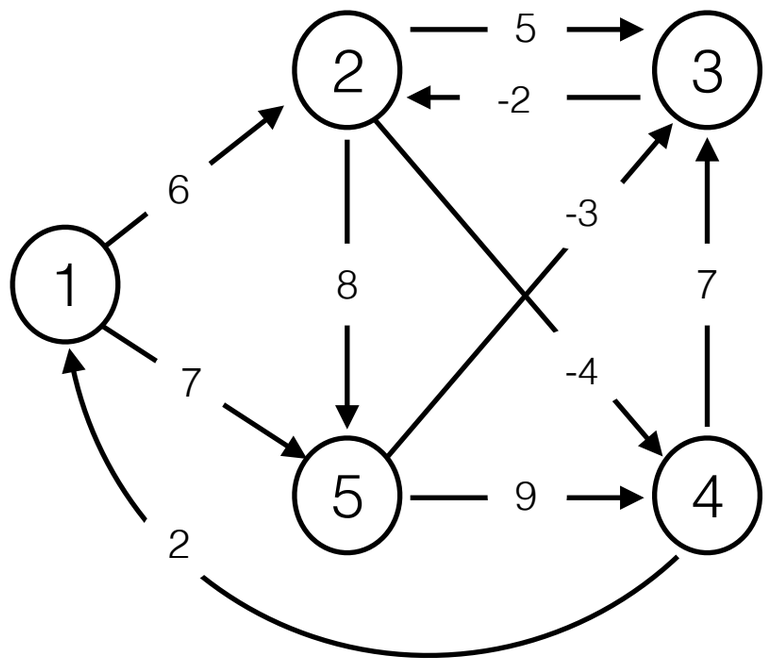
Algorithme de Bellman-Ford

On itère |S| fois l'idée de l'algorithme ordinal, mais sans tri !

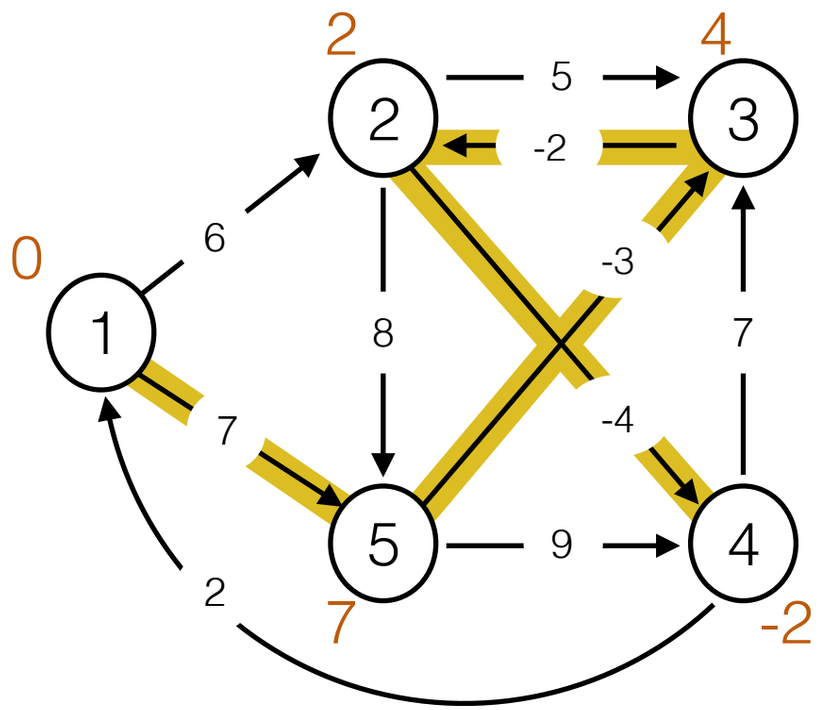
```
Bellman-Ford(G)=
    dist ⟵ [+oo, …, +oo ]
    dist[s] ⟵ 0
    pour k=1 à |S| faire
      pour tout (i,j)∈A
            si dist[i]+w(i,j)<dist[j]
            alors
                dist[j] ⟵ dist[i]+w(i,j)
                pred[j] ⟵ i
```

coût : O(A·S)

Exemple



Exemple



Algorithme de Bellman-Ford

```
Bellman-Ford(G)=
  dist ⟵ [+oo, …, +oo ]
  dist[s] ⟵ 0
  pour k=1 à |S| faire
    pour tout (i,j)∈A
          si dist[i]+w(i,j)<dist[j]
          alors
              dist[j] ⟵ dist[i]+w(i,j)
              pred[j] ⟵ i
pour tout (i,j)∈A
    si dist[i]+w(i,j)<dist[j]
    alors CycleNegatif()
```

Détecte s'il y a un cycle de poids négatif accessible depuis s

Modifiez cet algorithme pour éviter de considérer trop souvent des arcs (i,j) qui ne modifient pas dist[j]

Algorithme de Bellman-Ford

```
Bellman-Ford(G)=
  dist ⟵ [+oo, …, +oo ]
  dist[s] ⟵ 0
  W ⟵ {s}
tant que W non vide faire
  i ⟵ extraire un élément de W
  pour tout j∈Adj[i]
        si dist[i]+w(i,j)<dist[j]
        alors
           dist[j] ⟵ dist[i]+w(i,j)
           pred[j] ⟵ i
           W ⟵ W + {j}
```