Arbres de recherche
Le plus court chemin

Jerzy KORCZAK
email : jjk@dpt-info.u-strasbg.fr
http://lsiit.u-strasbg.fr/afd http://kia.u-strasbg.fr

Algorithmes de parcours d'un espace de recherche

- Recherche en profondeur d'abord (*DFS*) et en largeur (*BFS*)
- *Hill Climbing* (descente de gradient)
- Recherche heuristique : *Best-First search*
  - *en profondeur ordonnée, en faisceau, du meilleur premier*
- Algorithme A* , IDA*, SMA*
- Recherche concurrentielle : jeux
  - MIN-MAX, élagage α–β, approfondissement progressif

Propriétés
- Complétude
- Complexité en temps
- Complexité en espace
- Optimum

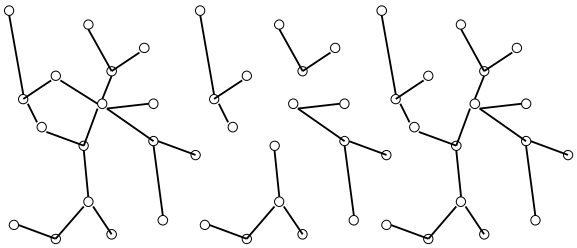
Graphes et arbres : rappel

Un *graphe orienté G* est représenté par un couple *(V,E)* où *V* est une ensemble fini et *E* une relation binaire sur *V*.
S est *l'ensemble des sommets* de *G* et *E* est *l'ensemble des arcs* de *G*.

Dans un *graphe orienté G(V,E),* un *chemin de longueur k* d'un sommet *u* à un sommet *v* est une séquence de sommets.

Un graphe non orienté acyclique est une *forêt* et un graphe non orienté connexe acyclique un est un *arbre*.

Exemples



Graphe contenant un cycle *Forêt* *Arbre*

Graphes et arbres : rappel

Propriétés des arbres
1. Deux sommets quelconques de *G(V,E)* sont reliés par un unique chemin élémentaire.
2. *G* est connexe et *IEI = IVI-1*.
3. *G* est connexe, mais si une arrête quelconque est ôtée de *E*, le graphe résultant n'est plus connexe.
4. *G* est acyclique et *IEI = IVI-1*.
5. *G* est acyclique, mais si une arrête quelconque est ajoutée à *E*, le graphe résultant contient un cycle.

- Un *arbre enraciné* est un arbre dans lequel l'un des sommets se distingue des autres (la *racine*)
- Le *degré* : le nombre de fils du nœud x
- La *profondeur* : la longueur du chemin entre la racine et le nœud x.

Problème de plus courts chemin

On possède en entrée un graphe orienté pondéré *G=(V,E)* de fonction de pondération $w: A \rightarrow \Re$.
Le poids du chemin $p = (v_0, v_1, \ldots, v_k)$ est la somme de ses arcs :

$$w(p) = \sum w(v_{i-1}, v_i), \qquad i=1,2,\ldots k$$

Parcours des graphes

Eviter de parcourir des cycles !

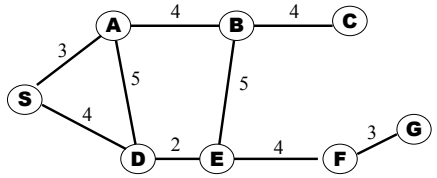
Convention : initialement les sommets sont tous ***blancs*** *; lorsqu'il est rencontré pour la première fois un sommet est peint en* ***gris*** *; lorsque tous ses successeurs dans l'ordre de parcours ont été visités, un sommet est repeint en* ***noir****.*

Parcours en profondeur

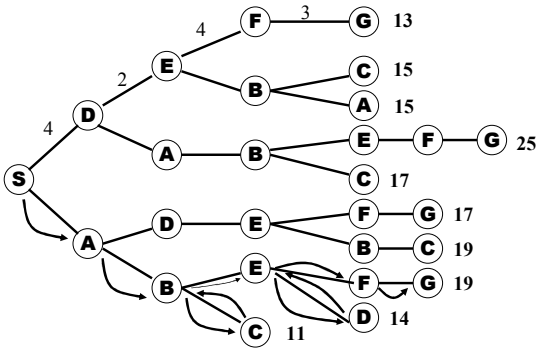
PP(G)
Pour chaque sommet *u* de *G* **faire** *couleur[u]* ← BLANC
Pour chaque sommet *u* de *G* **faire si** *couleur[u]*=BLANC **alors** VISITER-PP(*G,u,couleur*)

VISITER-PP(*G,u,couleur*)
couleur[s] ← GRIS
Pour chaque voisin de *v* de *s* **faire**
si *couleur[v]*=BLANC **alors** VISITER-PP(*G,v,couleur*)
couleur[s] ← NOIR

Exemple: Trouver un chemin de S à G



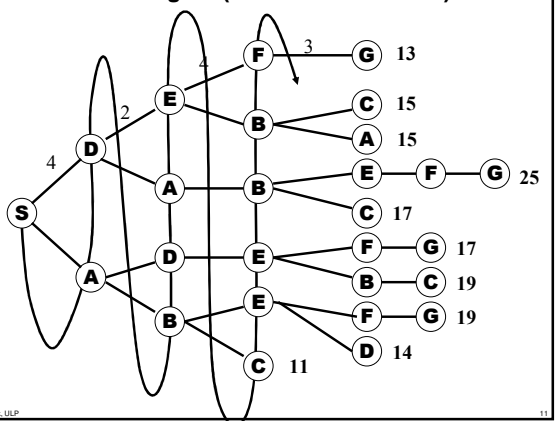
Parcours en profondeur (*depth-first search*)



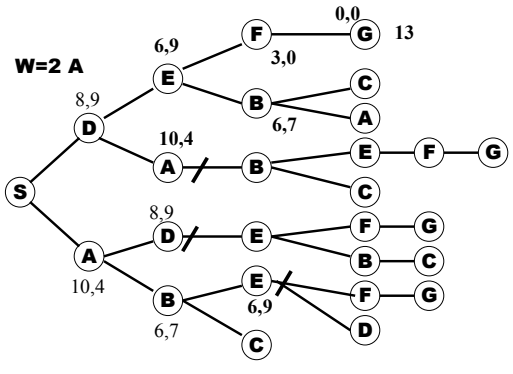
Parcours en largeur

PL(G)
couleur[s] ← GRIS
Pour chaque sommet *u* de *G*, *u*≠*s* **faire** *couleur[u]* ← BLANC
F ← {*s*}
Tant que *F*≠∅ **faire**
u ← SUPRESSION(*F*)
Pour chaque voisin de *v* de *u* **faire**
si *couleur[v]*=BLANC
alors *couleur*[v] ← GRIS
INSERTION (*F,v*)
couleur[s] ← NOIR

Parcours en largeur (*breadth-first search*)



Recherche heuristique (*beam search*)



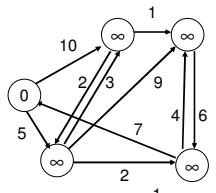
Plus courts chemins à origine unique

But : trouver les plus courts chemins depuis un sommet origine *s* et vers n'importe quel autre sommet.

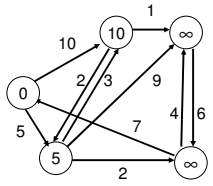
- Algorithme de Dijkstra
- Algorithme de Bellman-Ford
- Algorithme Floyd-Warshall

Algorithme de Dijkstra

L'a.D. maintient à jour un ensemble *V* de sommets de *G* dont le plus court chemin à partir de l'origine *s* est connu et calculé.



A chaque itération, l'algorithme choisit parmi les sommets successeurs – le sommet *u* dont l'estimation de plus court chemin est minimale.



Algorithme de Dijkstra

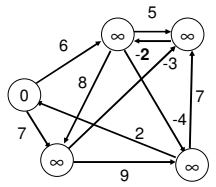
```
SOURCE-UNIQUE-INITIALISATION [G,s]
  pour chaque sommet v de G faire
        d[v]← +∞                    ; la longueur de chemin
        Π[v] ← NIL                  ; le prédécesseur de v
  d[s] ← 0

RELACHER(u,v,w)
  si d[v] > d[u] + w(u,v) alors
        d[v] ← d[u] + w(u,v)
        Π[v] ← u

DIJKSTRA(G,w,s)
  SOURCE-UNIQUE-INITIALISATION [G,s]
  E ← ∅
  F ← V
  Tant que F≠∅ faire
    u ← EXTRAIRE-MIN(F)             ; détermine la complexité → Θ(|F|)=O(|V|)
    E ← E ∪ {u}
        pour chaque arc (u,v) de G faire   ; la complexité O(|V|2+ |A|) = O(|S|2
            RELACHER(u,v,w)
```

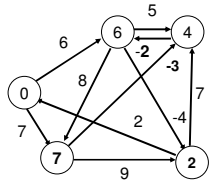

Algorithme de Bellman-Ford

L'a.B-F renvoie un booléen indiquant si le graphe contient ou non un circuit de poids strictement négatif accessible à partir de l'origine.




```
BELLMAN-FORD(G,s, w)
SOURCE-UNIQUE-INITIALISATION [G,s]
Pour i ←1 à |S|-1  faire
  Pour chaque arc  (u,v)∈A  faire
     RELACHER(u,v,w)
  Pour chaque arc (u,v) A faire
     si d[v]> d[u] + w(u,v) alors envoyer FAUX
  renvoyer VRAI
```

A la fin de la i^e itération de la première boucle, les plus courts chemins contenant au plus *i* arcs sont connus.



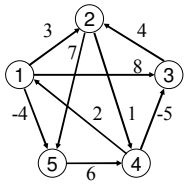
Algorithme de Floyd-Warshall

```
FLOYD-WARSHALL(W)
  n ← lignes(W)
  D(0) ← W
   Pour k ← 1 à n faire
     Pour i ← 1 à n faire
       Pour j ← 1 à n faire
         d_ij^k ← min(d_ij^(k-1) , d_ik^(k-1) + d_kj^(k-1) )
   renvoyer D(n)
```

La complexité : $\Theta(n^3)$

Algorithme de Floyd-Warshall

Principe de la programmation dynamique



$D^{(0)} =$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$D^{(1)} =$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

$D^{(2)} =$

0	3	8	2	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

…