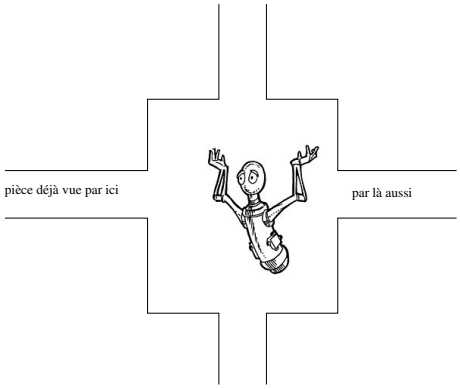
IREM de Clermont-Ferrand
Formation ISN
Algorithmique

13 décembre 2013
Alex Esbelin
Malika More

Parcours

1 Généralités

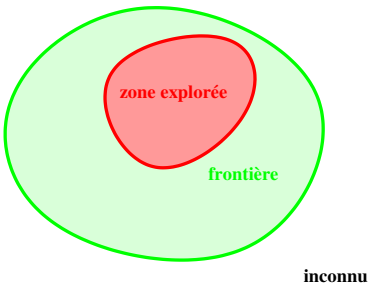
Le robot explorateur Un robot a pour mission d'explorer un bâtiment d'importance archéologique. Ce bâtiment a été englouti par une coulée de boue. Au cours des âges, la boue a séché et forme des blocs compacts difficiles à percer.



Les balises Le robot dispose de petites balises pour marquer chaque pièce explorée et/ou aperçue. Ainsi depuis la pièce où il se trouve, il peut détecter quels couloirs mènent à une pièce balisée.
Question Quelle stratégie pourrait-il adopter afin d'explorer entièrement le bâtiment en minimisant le nombre de couloirs traversés (puisqu'il faut déblayer la boue avant) ?

Parcours d'un graphe
Stratégie On garde en mémoire l'ensemble des sommets connus, partitionné en deux :
- d'une part, ceux qu'on a complètement visités, qui forment la *zone explorée* ; et,
- d'autre part, ceux qu'on n'a pas encore entièrement visités, ou seulement aperçus, qui forment la *frontière* de la zone explorée.

Méthode On se place sur un sommet s de la frontière :
- ou bien tous les voisins de s sont dans la zone explorée ou dans la frontière
  - ⋆ et on peut ajouter s à la zone explorée ;
- ou bien s a au moins un voisin t dans l'inconnu
  - ⋆ et on peut ajouter t à la frontière.



Comment organiser la frontière ?

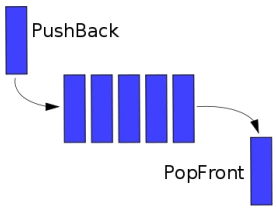
- **Le plus simple :** On prend un sommet *au hasard* dans la frontière, on traite la frontière comme un *ensemble*.
  - ⋆ Si on marque l'arête $\{s, t\}$ lorsqu'on ajoute le voisin t de s dans la frontière pendant le parcours,
  - ⋆ on obtient un *arbre recouvrant* quelconque.
- **Premier arrivé = premier servi :** On fait attendre un sommet le moins possible dans la frontière, on traite la frontière comme une *file d'attente*.
  - ⋆ On obtient un arbre tel que le niveau d'un sommet dans l'arbre correspond à la distance au sommet de départ.
  - ⋆ Application : distance à la source.
  - ⋆ On parle de *parcours en largeur*.
- **Dernier arrivé = premier servi :** Ou bien au contraite, on traite la frontière comme une *pile d'attente*.
  - ⋆ On obtient un arbre couvrant tel que tout arête du graphe qui n'est pas dans l'arbre va forcément depuis un sommet vers un de ses ancêtres
  - ⋆ Application : tri topologique
  - ⋆ On parle de *parcours en profondeur*.

En résumé

- Parcourir un graphe connexe induit un *arbre recouvrant*.
- Différentes méthodes selon la structure de données choisie pour stocker la frontière.
  - ⋆ File d'attente
    - Parcours en largeur
    - Distance à la source
  - ⋆ Pile d'attente
    - Parcours en profondeur
    - Tri topologique
    - Calcul des blocs

Définition d'une file. Les primitives permettant de gérer une file sont :

- Ajouter un sommet (toujours derrière) la file (`pushback`)
- Enlever un sommet (toujours devant) la file (`popfront`) ; et,
- Vérifier si la file est vide (`empty?`).
- On peut aussi juste regarder le sommet devant la file sans l'enlever (`CheckFront`)



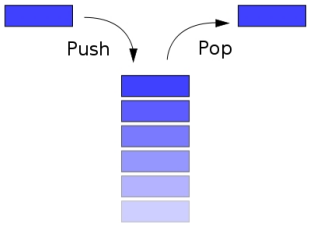
Les files en pratique

- Peut se coder facilement avec des pointeurs
- Des librairies implémentant les files existent dans la majeure partie des langages de programmation
- En anglais : FIFO (*First In First Out*)

Exemple. File d'attente associée à une imprimante.

Définition d'une pile. Les primitives permettant de gérer une pile sont :

- Ajouter un sommet (toujours devant la file) (`push(front)`)
- Enlever un sommet (toujours devant la file) (`pop(front)`) ; et,
- Vérifier si la file est vide (`empty?`).
- On peut aussi juste regarder le sommet devant la file sans l'enlever (`CheckFront`)



Les piles en pratique

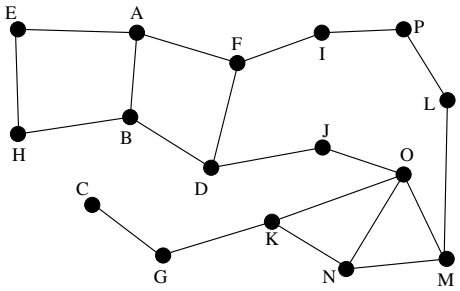
- En anglais : *Stack* ou bien LIFO *(Last In First Out)*
- Peut se coder facilement avec des pointeurs
- Des librairies implémentant les piles existent dans la majeure partie des langages de programmation

Exemple.

- On peut utiliser une pile si on veut implémenter un *undo*
- Il y a aussi la pile des *appels récursifs*

2 Parcours en largeur

Exemple



Algorithme du parcours en largeur

Entrées

- G non orienté
- s source

Variables locales

- F file (frontière)
- S ensemble des sommets connus
- u, v des sommets

Initialisation

- Ajouter s à F
- Ajouter s à S

Calcul

Tant que F n'est pas vide, répéter

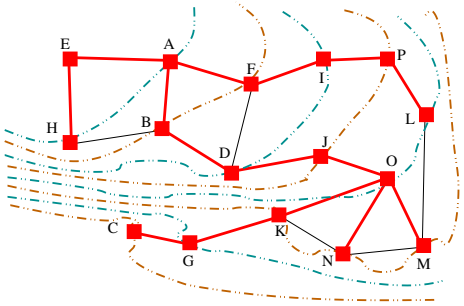
- regarder le sommet v sur le devant de F,
- si v possède des voisins encore inconnus
  - ⋆ choisir un voisin inconnu u de v
  - ⋆ ajouter u à la fin de F
  - ⋆ ajouter u à S
- sinon enlever v de la file

Algorithme 1 : Parcours en largeur

Données G *un graphe* s *un sommet de* G
Variables locales
S *un ensemble* `/* sommets connus */`
F *une file* `/* la frontière */`
u, v *deux sommets*
début
  initialisation
  `add(`s`,`S`)`
  `pushBack(`s`,`F`)`
  tant que `not empty?(`F`)` **faire**
    v :=`checkFront` (F)
    si *il existe* $u \notin S$ *adjacent à* v **alors**
      `add(`u`,`S`)` `/* première visite de` u `*/`
      `pushBack(`u`,`F`)`
    sinon
      `popFront(`F`)` `/* dernière visite de` v `*/`
    fin
  fin
fin

Propriétés d'un arbre de parcours en largeur :

Théorème (Distance et Niveau)**.** *Le niveau d'un sommet dans l'arbre est égal à la distance à la source dans le graphe. Autrement dit, les distances à la source dans l'arbre et dans le graphe sont les mêmes.*



Preuve du théorème

- Si un sommet est au niveau d de l'arbre, alors il existe un chemin de longueur d de ce sommet à la source (par l'unique branche remontant à la source). Donc la distance est plus petite ou égale au niveau.
- On peut montrer l'inégalité inverse par récurrence sur le nombre de niveaux/les étapes de l'algorithme.
  - ⋆ à la première étape, les voisins de la source sont ajoutés au niveau 1 de l'arbre.
  - ⋆ supposons l'égalité vraie au rang n. Un sommet u à distance $n+1$ est voisin d'un sommet v à distance n. Or le sommet v est au niveau n de l'arbre par hypothèse et u ne peut pas avoir été ajouté avant v. L'algorithme du parcours en largeur va donc ajouter u au niveau $n+1$.
- Conclusion : on a bien distance=niveau

Calcul des distances à la source
Entrées

- G non orienté
- s source

Variables locales

- F file (frontière)
- S ensemble des sommets connus
- u, v des sommets adjacents

Sortie
- *dist* le tableau des distances à la source s

Initialisation
- Ajouter s devant la file
- Ajouter s à S
- dist[s] :=0

Calcul

Tant que la file n'est pas vide, répéter
- prendre le premier sommet v de la file d'attente,
- ajouter chaque voisin inconnu u de v à l'arrière de la file, ajouter u à S et d[u] :=d[v]+1
- enlever v de la file

Algorithme 2 : Parcours en largeur + distance à la source

```
Variables locales
L tableau des distances à la source                    /* indexés par les sommets */
début
    initialisation
    add(s,S)
    pushBack(s,F)
    L[s] := 0
    tant que not empty?(F) faire
        v :=checkFront (F)
        si il existe u ∉ S adjacent à v alors
            add(u,S)                                   /* première visite de u */
            pushBack(u,F)
            L[u] := L[v] + 1
        sinon
            popFront(F)                                /* dernière visite de v */
        fin
    fin
fin
```

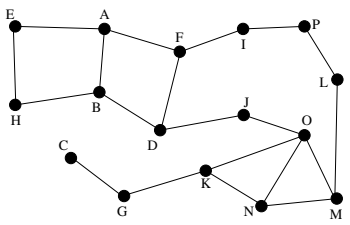
Sorties : L, tableau des distances à la source s

Arbre de parcours en largeur

Propriété. On considère un graphe connexe et un arbre de parcours en largeur. Tout arête du graphe est :

1. soit une arête de l'arbre de parcours en largeur
2. soit si ce n'est pas le cas, une arête entre deux sommets dont les niveaux diffèrent d'au plus 1

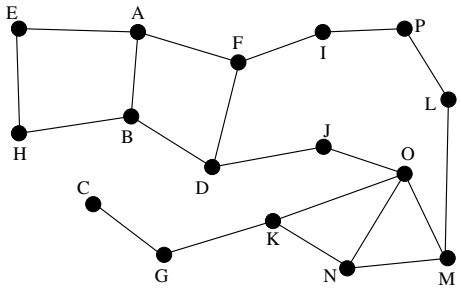
Exercice 1.



Effectuez un parcours en largeur depuis O, en choisissant toujours le plus petit sommet dans l'ordre alphabétique. Indiquez à chaque étape l'état de la file et de l'arbre de parcours. En déduire les distances entre O et les autres sommets.

3 Parcours en profondeur

Exemple



Algorithme du parcours en profondeur
Entrées
- G non orienté
- s source

Variables locales
- P pile (frontière)
- S ensemble des sommets connus
- u, v des sommets

Initialisation
- Ajouter s à P
- Ajouter s à S

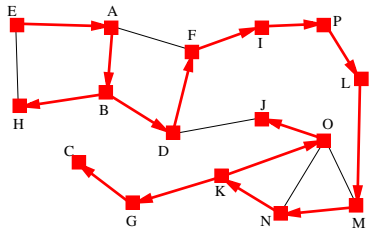
Calcul
Tant que P n'est pas vide, répéter
- regarder le sommet v au sommet de P
- si v possède des voisins encore inconnus
  - ⋆ choisir un voisin inconnu u de v
  - ⋆ ajouter u sur P
  - ⋆ ajouter u à S
- sinon enlever v de la pile P

Algorithme 3 : Parcours en profondeur

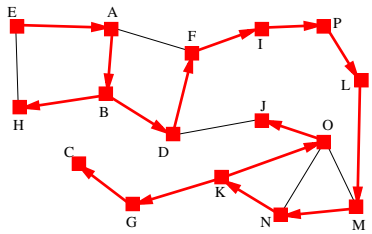
Données *G un graphe et s un sommet de G*
Variables locales *S un ensemble* `/* sommets connus */`
F une pile `/* la frontière */`
u, v deux sommets
début
  initialisation
  `add(`s`,`S`)`
  `pushFront(`s`,`F`)`
  tant que `not empty?(`F`)` **faire**
    v :=`checkFront` (F)
    si *il existe* $u \notin S$ *adjacent à* v **alors**
      `add(`u`,`S`)` `/* première visite de` u `*/`
      `pushFront(`u`,`F`)`
    sinon
      `popFront(`F`)` `/* dernière visite de` v `*/`
    fin
  fin
fin

Propriétés d'un arbre de parcours

Remarque. Dans le cas d'un arbre de parcours, on part de la source. On a donc un arbre enraciné. Cela revient à considérer un arbre orienté par le sens du parcours depuis la source vers les feuilles. On peut donc parler d'ancêtre d'un sommet.



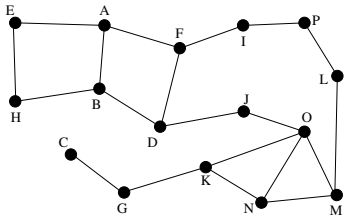
Propriétés d'un arbre de parcours en profondeur



Observation Tout arête de G est :

1. soit une arête de l'arbre,
2. soit si ce n'est pas le cas, une arête entre deux sommets tels que l'un est l'ancêtre de l'autre (arc arrière).

Exercice 2.



Effectuez un parcours en profondeur depuis O, en choisissant toujours le plus petit sommet dans l'ordre alphabétique. Indiquez à chaque étape l'état de la pile et de l'arbre de parcours.

Temps de découverte des sommets
Problème

- Pour chaque sommet du graphe,
  - ⋆ on voudrait savoir à quelle étape le sommet a été ajouté à la frontière (c'est-à-dire la première fois qu'il a été visité) ; et,
  - ⋆ à quelle étape il en est sorti (c'est-à-dire la dernière fois qu'il a été visité)
- On ajoute deux tableaux `Début` et `Fin` à notre algorithme pour stocker ces informations

Algorithme 4 : Parcours en profondeur + début et fin

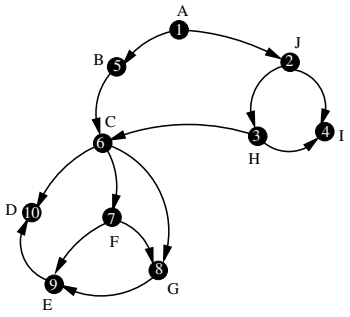
```
Variables locales
Debut et Fin deux tableaux                          /* indexés par les sommets */
début
    initialisation
    x=0
    add(s,S)
    pushFront(s,F)
    Début[s] := x
    tant que not empty?(F) faire
        x := x + 1
        v :=checkFront (F)
        si il existe u ∉ S adjacent à v alors
            add(u,S)                                /* première visite de u */
            pushFront(u,F)
            Début[u] := x
        sinon
            popFront(F)                             /* dernière visite de v */
            Fin[v] := x
        fin
    fin
fin
```

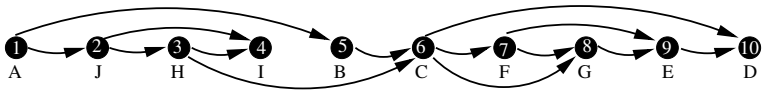
Ordre de la visite d'un sommet
Observation On termine toujours l'exploration d'un sommet avant de terminer l'exploration de ses ancêtres. (terminer l'exploration d'un sommet= enlever ce sommet de la frontière)

Théorème (Début et Fin)**.** *Si x est un voisin de y tel que x est visité pour la première fois avant y, alors y est visité pour la dernière fois avant que x ne soit visité pour la dernière fois.* $E(x,y)\&Debut(x) < Debut(y) \Rightarrow Fin(y) < Fin(x)$

Ordre topologique

Définition. Soit $\overrightarrow{G}$ un graphe orienté sans cycles dirigés. Un ordre total $<$ sur les sommets est dit topologique lorsqu'il est compatible avec les arcs du graphe, i.e. pour tout arc (u, v) de $\overrightarrow{G}$, on a $u < v$.





Calcul d'un ordre topologique
Le problème

- Entrée : un graphe $\overrightarrow{G}$ sans cycle
- Problème : calculer un ordre topologique

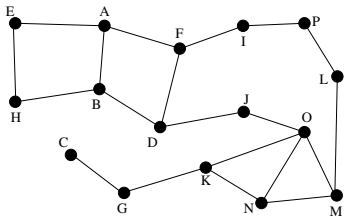
Remarque Un tel graphe représente typiquement *un problème d'ordonnancement de tâches* : un sommet correspond à une tâche et un arc (u, v) indique la contrainte de précédence "*la tâche u doit être terminée avant que la tâche v ne puisse commencer*".
Solution On prend les sommets dans l'ordre inverse de leur dernière visite dans un parcours en profondeur.

Concrètement
Méthode

- Ajouter un sommet source ayant un arc vers tous les autres (si nécessaire)
- Faire un parcours en profondeur en notant pour chaque sommet le temps de dernière visite
- Ce temps est la priorité du sommet dans le problème d'ordonnacement correspondant
- Plus la priorité est élevé, plus la tâche doit être exécutée tôt, donc l'ordre topologique est l'ordre décroissant des priorités

Attention Il y a plusieurs parcours possibles en général et donc plusieurs ordres topologiques possibles.

Exercice 3.



1. Orientez les arcs du graphe selon l'ordre alphabétique. Justifiez que le graphe orienté obtenu $\overrightarrow{G}$ est acyclique.
2. Ajouter un sommet qui précède tous les sommets de $\overrightarrow{G}$. Faire un parcours en profondeur du graphe orienté à partir de ce nouveau sommet.
3. En déduire un ordre topologique pour $\overrightarrow{G}$.

Exercice 4. Au cours de la rénovation de son habitation, la famille M. a décidé de transformer en cuisine une ancienne remise. Le tableau ci-dessous énumère les différentes tâches et les contraintes d'antériorité entre ces tâches.

Désignation	Libellé	Tâches préalables
A	Plomberie	-
B	Gaines électriques	-
C	Isolant plafond	F
D	Isolant murs	E
E	Rails de fixation des plaques de plâtre des murs	A,B
F	Rails de fixation des plaques de plâtre du plafond	A,B
G	Plaques de plâtre des murs	D,M
H	Plaques de plâtre du plafond	C
I	Carrelage	G
J	Peinture	G,H
K	Meubles et plans de travail	I,J,L
L	Prises électriques et interrupteurs	G
M	Fenêtres	-

1. Construire le graphe des tâches associé.
2. Effectuer un parcours en profondeur sur le graphe des tâches. En cas d'ambiguité, on choisira les sommets dans l'ordre alphabétique. Pour chaque sommet, on précisera l'instant de découverte et l'instant de clôture.
3. En déduire un ordre topologique de réalisation des tâches qui respecte les contraintes d'antériorité.

Exercice 5. Un dandy doit s'habiller. Pour être fin prêt, il doit mettre :

1. son caleçon,
2. ses chaussettes,
3. ses chaussures,
4. sa chemise,
5. ses boutons de manchettes,
6. sa ceinture,
7. sa cravate,
8. son épingle à cravate,
9. son pantalon,
10. son gilet,
11. sa montre de gousset,
12. sa veste,
13. sa pochette (son vilain petit mouchoir),
14. et son chapeau.

Il ne peut toutefois pas mettre ses vêtements dans n'importe quel ordre :
- il doit mettre son caleçon avant son pantalon et ses chaussures ;
- ses chaussures ne peuvent être enfilées avant les chaussettes ;
- la chemise doit être mise avant la ceinture, les boutons de manchettes, le chapeau et la cravate ;
- la cravate doit être nouée avant de pouvoir être fixée par l'épingle à cravate ;
- la cravate doit être nouée avant de fermer le gilet ;
- la montre-à-gousset se porte sur le gilet
- la veste ne peut être mise avant la montre de gousset
- la ceinture et les chaussures ne peuvent pas être mises avant le pantalon.

Proposez-lui une solution.