Théorie des graphes

L. Sais

Algorithmique & Programmation 5

7 avril 2011

1. Graphe et algorithme : présentation
   - Introduction
   - Définitions et terminologie
   - Représentation
     - Matrice d'adjacence
     - Listes d'adjacence
2. Parcours, numérotation et descendance
3. Connexité et forte connexité
4. Graphes sans circuit
5. Problème du plus court chemin

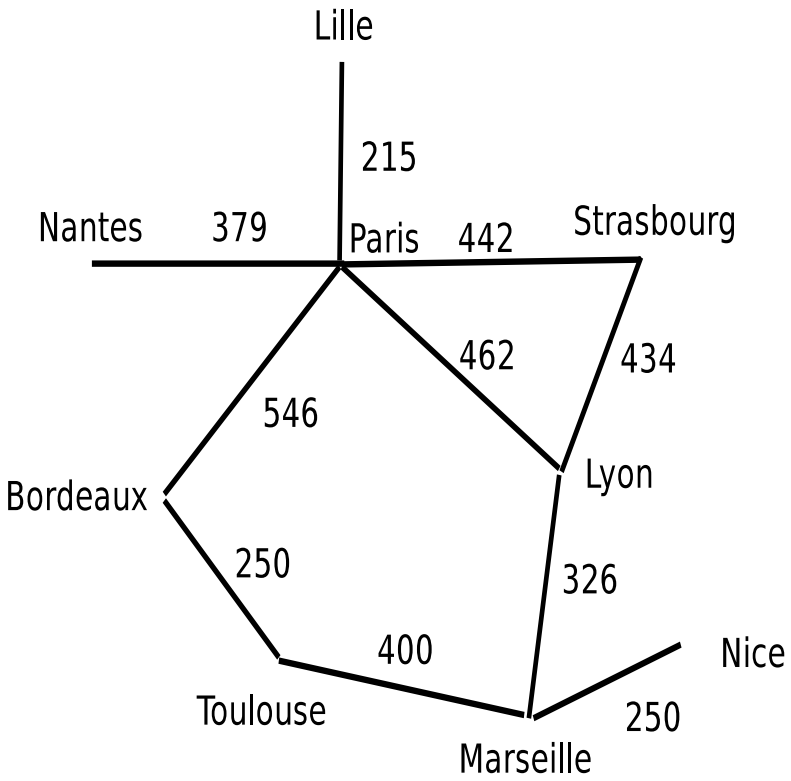
Graphes et algorithme : présentation

Introduction

De nombreux problèmes courants, en informatique, en ingénierie, en sciences sociales, en intelligence artificielle, peuvent se représenter en termes de relations (binaires) entre objets.

On peut citer par exemple :

- réseaux de communications : routiers, ferroviaires, machines en réseaux, circuits intégrés, ...

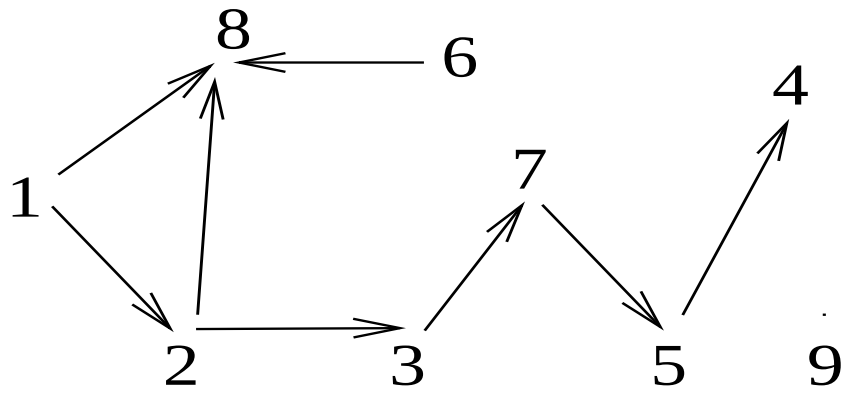


On distingue les sommets (villes, carrefours, soudures) et les arcs ou arêtes (communication entre sommets) qui mettent en relation les sommets. On associe parfois aux arcs des caractéristiques, qui permettent d'exprimer des distances, des coûts, des flux, ...

Le type des problèmes que l'on peut se poser alors concerne par exemple la recherche d'itinéraires optimaux (problème classique du voyageur de commerce : visiter toutes les villes avec un cheminement optimal).

- relations sociales : familiales, hirarchiques, d'incompatibilité, ...
  Ici, les sommets correspondent aux individus, les arcs aux relations entre individus.

- ordonnancement de tâches : atelier, systmes d'informations, ...
  Une première possibilité consiste représenter les instants de début et de fin d'activité par les sommets, et les activités par les arcs ; on peut également représenter les tâches par les sommets et les relations de précédance par les arcs.



1 : caleçon	4 : veste	7 : chemise
2 : pantalon	5 : cravate	8 : chaussures
3 : ceinture	6 : chaussettes	9 : montre

- représentation d'algorithmes (organigrammes) :
  Les sommets correspondent aux instructions ou tests, les arcs correspondent aux enchanements possibles entre instructions ou tests.
  On retrouvera également :
  * les emplois du temps
  * les automates à états finis
  * les problèmes de coloration de cartes (2 pays voisins doivent avoir 2 couleurs différentes)
  * etc.

Remarque

Le formalisme des graphes permet de décrire de nombreux problèmes souvent de manière simple, mais qui peuvent s'avérer difficile à résoudre.

Dans le cadre de ce cours, on traitera des problèmes classiques où de bons algorithmes sont connus (il existe des problèmes où l'on ne connait pas de bons algorithmes) :

- étant donné un graphe, vérifier s'il possède certaines propriétés
- étant donné un graphe, déterminer une partie possédant certaines propriétés.
- construire un, plusieurs, ou tous les graphes possédant certaines propriétés.

Graphes et algorithme : présentation

Définitions et terminologie

Définition

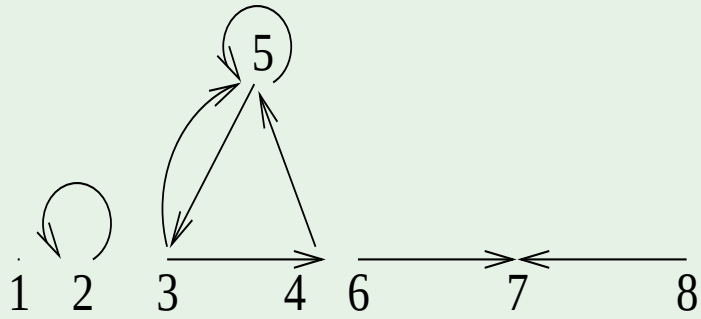
Un graphe orienté est un couple $G=(S,A)$ où :
- S est un ensemble fini d'éléments appelés sommets
- A est un ensemble fini de couples de sommets appelés arcs

on a $A \subset S \times S$

Remarque

S'il existe plusieurs arcs entre deux sommets, on parle d'un multigraphe (arcs multiples)

Exemple



S={1..8}
A={(2,2),(3,5),(4,5),(3,4),(5,5),(5,3),(6,7),(8,7)}

Définitions

Etant donné un graphe G=(S,A) et $x \in S$

- si $(x, y) \in A$, alors y est un successeur de x
- si $(y, x) \in A$, alors y est un prédécesseur de x
- x et y sont adjacents si y est un prédécesseur et/ou un successeur de x
- l'ensemble des successeurs de x est noté $\Gamma^+(x)$
  $\Gamma^+(x) = \{y | (x, y) \in A\}$
- l'ensemble des prédécesseurs de x est noté $\Gamma^+(x)$
  $\Gamma^-(x) = \{y | (y, x) \in A\}$
- l'ensemble des voisins de x est noté $\Gamma(x)$
  $\Gamma(x) = \Gamma^+(x) \cup \Gamma^-(x)$

- le degré intérieur de x est noté $d^-(x)$
  $d^-(x) = |\Gamma^-(x)|$
- le degré extérieur de x est noté $d^+(x)$
  $d^+(x) = |\Gamma^+(x)|$
- le degré de x est noté $d(x)$
  $d(x) = |\Gamma(x)|$

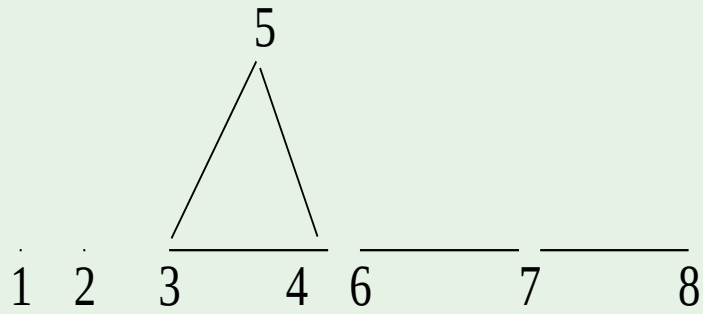
Définition

Un graphe non orienté est un couple G=(S,A) où :
- S est un ensemble fini d'éléments appelés sommets
- A est un ensemble fini de paires de sommets appelés arêtes

Une arête est notée $\{x,y\}$

Dans le cas de ces graphes non-orientés, on parle simplement de sommets voisins et de degré.

Exemple



S={1..8}
A={{3,5},{4,5},{3,4},{6,7},{8,7}}

Définition (graphe complet)

- Un graphe orienté G=(S,A) est dit complet si $A=S^2$
- Un graphe non orienté G=(S,A) est dit complet si toute paire de sommets figure dans A

Définition

Etant donné un graphe orienté G=(S,A) :

- le graphe réciproque de G est le graphe $G^{-1}=(S,A^{-1})$ où $A^{-1}=\{(x,y) \in S \times S \mid (y,x) \in A\}$
- le graphe symétrisé associé à G est le graphe $G_S=(S,A \cup A^{-1})$

Définition

Un graphe valué est un graphe orienté ou non orenté G=(S,A) auquel on associe une fonction de coût c : A->R , un tel graphe est noté (S,A,c)

Exemple

Voir l'exemple sur le réseau routier (villes)

Notations

Si G=(S,A) On a |S|=card(S)=n et |A|=card(A)=m

Rapports etre n et m : (cas des graphes orientés)

- Si G=(S,A) n'est pas un multigraphe, on a $A \in S^2$
- graphes tels que $m=n^2$: graphes complets avec boucles
- graphes complets sans boucle : $m=n(n-1)$

Relations entre d^+, d^-, d et m :

- pour le cas des graphes orientés sans boucles :

$$\sum_{x \in S} d^+(x) = \sum_{x \in S} d^-(x) = \frac{1}{2} \sum_{x \in S} d(x) = m$$

- pour le cas des graphes non orientés :

$$\sum_{x \in S} d(x) = 2m$$

Définition

Etant donné un graphe orienté G=(S,A) et un sous-ensemble de sommets X, le sous-graphe de G induit (engendré) par X est G(X)=(X,E)
où E=$\{(x, y) \in A | x, y \in X\}$

La définition est identique dans le cas non orienté

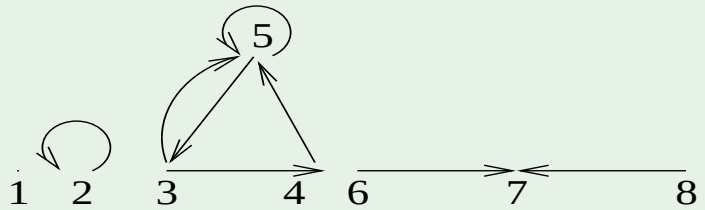
Définition

Etant donné un graphe G=(S,A) et un sous ensemble d'arcs ou d'arêtes E, le graphe partiel de G induit par E est le graphe G=(S,E)

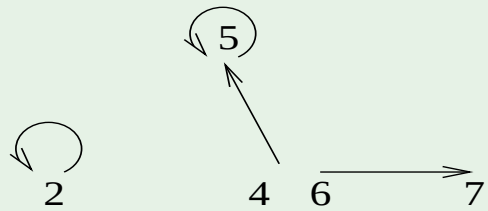
Exemple

Un graphe, le sous-graphe induit par X ={2,4,5,6,7}
et le graphe partiel induit par l'ensemble d'arcs
E={(2,2),(5,3),(4,5),(3,4),(8,7)}

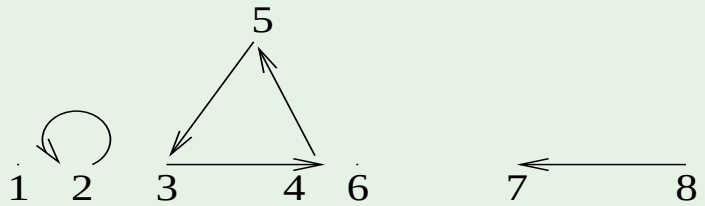
graphe



sous-graphe



graphe partiel



Définition

Etant donné un graphe G=(S,A), une partition p=$\{S_1,...,S_p\}$ de S,
le graphe quotient induit par P est la graphe (V,E) où :

- V=$\{s_1,...,s_p\}$ (à chaque partie S, on associe un sommet S_i)
- E=$\{(s_i,s_j)$ / $\exists$ (x,y)$\in$A , x$\in S_i$ et y$\in S_j\}$

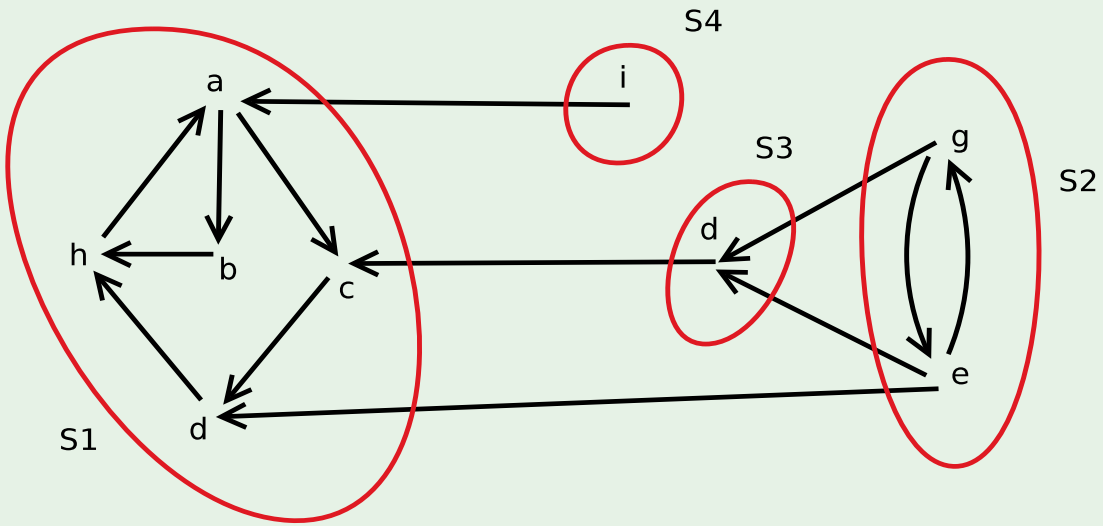
Rappel

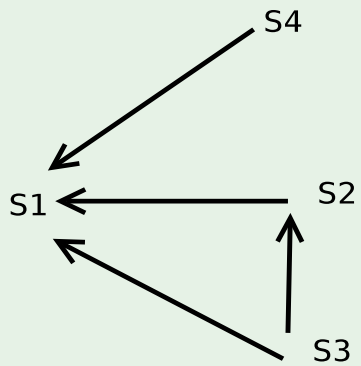
p=$\{S_1,...,S_p\}$ est une partition de S ssi

$$\bigcup_{1\leq i\leq p} S_i = S \quad \text{et} \quad \forall i \neq j,\ S_i \bigcap S_j = \emptyset$$

Exemple

Un graphe et le graphe quotient induit par la partition p=$\{s_1,\ldots,s_4\}$





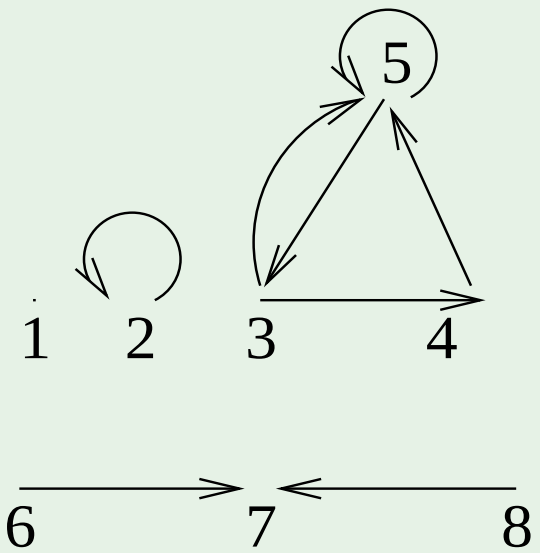
Graphes et algorithme : présentation

Représentation

Matrice d'adjacence

Le graphe est représenté par un tableau à deux dimensions indexé sur l'ensemble des sommets

Graphe et matrice d'adjacence



X	1	2	3	4	5	6	7	8
1	F	F	F	F	F	F	F	F
2	F	V	F	F	F	F	F	F
3	F	F	F	V	V	F	F	F
4	F	F	F	F	V	F	F	F
5	F	F	V	F	V	F	F	F
6	F	F	F	F	F	F	V	F
7	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	V	F

Les éléments du tableau peuvent être différents types selon le graphe considéré

- booléens : un élément [i,j] est vrai si l'arc (i,j) figure dans le graphe. Dans le cas non orienté si l'élément [i,j] est vrai, alors l'élément [j,i] est vrai
- réels : Dans le cas de graphes valués, on représente la fonction de coût, ainsi pour valeur c(i,j) et si l'arc (i,j) ne figure pas dans le graphe [i,j] prend une valeur conventionnelle.
- entiers : Dans le cas du multigraphe, [i,j] aura pôur valeur le nombre d'arcs (i,j) figurant dans le graphe.

en C :

```
#define N_MAX

typedef struct{
  int g[N_MAX][N_MAX];
  int n;
}grapheM;
```

en pseudo-langage :

```
constante N_MAX = ... (* nombre maxi de sommets *)

type sommet = 1..N_MAX
     graphe = structure
                a:= tableau[sommet,sommet] de booleens
                n: 0..N_MAX
              fin
```

Avantages

- Accès direct aux arcs : tester l'existence d'un arc, ajouter, ou supprimer un arc sont des opérations réalisables en temps constant ($\theta(1)$)
- L'écriture des algos est généralement simplifiée :
  l'accès aux successeurs d'un sommet est aisé

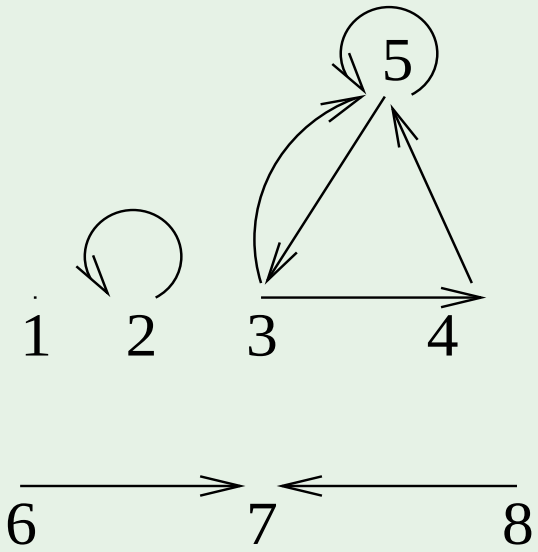
Inconvénients

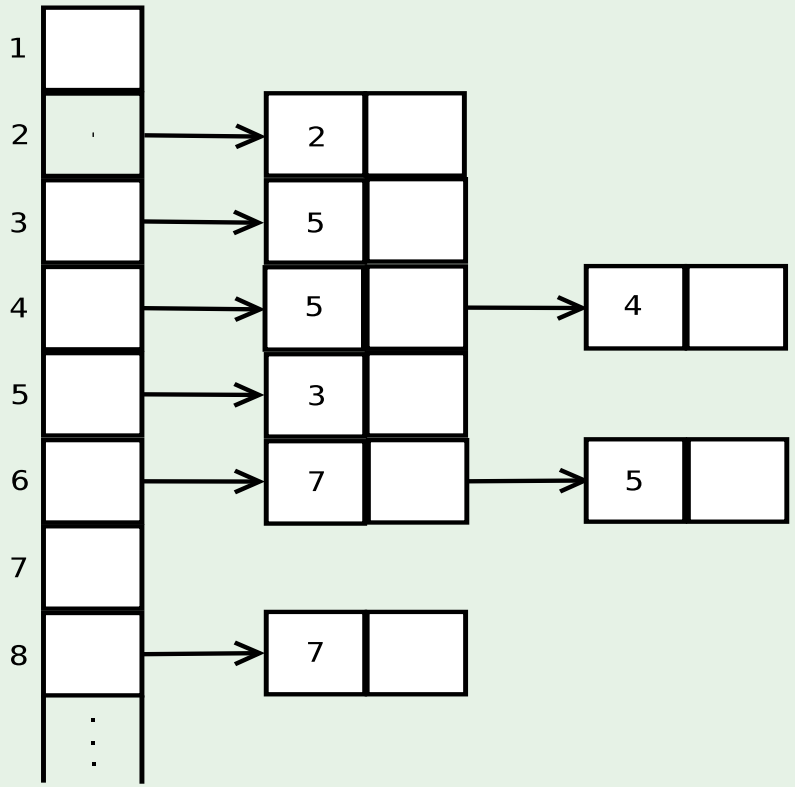
- encombrement maximal de la mémoire :
  $\forall$ le nombre d'arcs figurant dans le graphe, la place mémoire nécessaire est en $\theta(\text{N\_MAX}^2)$; cas génant : $m \leq n^2$
- coût de l'initialisation : $\theta(n^2)$
- coût de la recherche des successeurs d'un sommet : $\theta(n)$, même s'il y'a pas de successeurs.
  l'accès aux successeurs d'un sommet est aisé

Listes d'adjacence

C'est généralement la représentation la plus utilisée, celle où l'on dispose des algos les plus performants.

Exemple





en C :

```
typedef struct cel{
  int st;
  struct cel* suiv;
}cellule;

typedef cellule* Liste;

typedef struct{
  Liste a[N_MAX];
  int n;
}grapheL;
```

en pseudo-langage :

```
type Liste = ^cellule
     cellule = structure
               st : sommet
               suiv : Liste
               fin
     grapheL = structure
               a := tableau [sommet,sommet] de Liste
               n :0..N_MAX
               fin
```

Avantages

- encombrement minimal de la mémoire : θ(N_MAX+m)
- l'accès au successeur d'un sommet x est en $\theta(d^+(x))$

Inconvénients

- cout d'accès à un arc l'origine x : $\theta(d^+(x))$
- l'accès aux prédécesseurs d'un sommet x impose le parcours de toutes les listes d'adjacence : θ(n+m)

1 Graphe et algorithme : présentation

2 Parcours, numérotation et descendance

3 Connexité et forte connexité

4 Graphes sans circuit

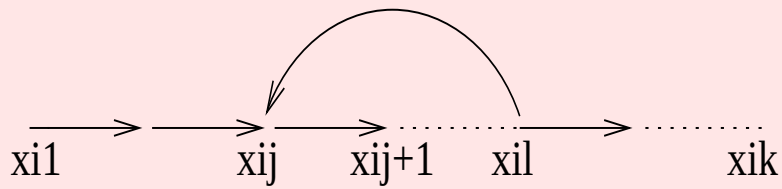
5 Problème du plus court chemin

Définition (Graphe orienté)

- un chemin de G=(S,A) est une séquence de sommets x_{i_1}, x_{i_2},...,x_{i_k}
  $k \geq 1$, $\forall j\ 1 \leq j \leq k\text{-}1$, $(x_{i_j},x_{i_{j+1}}) \in A$
  un chemin $x_{i_1},x_{i_2},...,x_{i_k}$ sera noté $[x_{i_1},x_{i_2},...,x_{i_k}]$
- un chemin $[x_{i_1},x_{i_2},...,x_{i_k}]$ est dit <u>élémentaire</u> si
  $\forall\ u,v\ ,\ 1 \leq u \leq v \leq k\ ,\ x_{i_u} \neq x_{i_v}$
  (ne contient pas deux fois le même sommet)
- la <u>longueur d'un chemin</u> est égale au nombre de sommets moins un.
  le chemin constitué du seul sommet x_{i_1} noté $[x_{i_1}]$ est de longueur nulle.

Propriété

si $c=[x_{i_1},...,x_{i_k}]$ est un chemin de G, alors il existe une sous-suite de c qui est un chemin élémentaire de x_{i_1} à x_{i_k}.



Preuve :

Par récurrence sur r, le nombre de répétitions de sommets dans c.

- si r=0, c'est un chemin élémentaire
- Supposons que la propriété est vraie au rang r-1
  Soit un chemin c=$[x_{i_1},\ldots,x_{i_k}]$ possédant r répétitions.
  Il existe u et v $1 \leq u \leq v \leq k$ tels que x_{i_u} à x_{i_v}.

  Soit c' la sous-séquence de c obtenue en supprimant de c la sous-suite $x_{i_u+1} \ldots x_{i_v}$.

  Donc c'=$[x_{i_1},\ldots,x_{i_u},x_{i_v+1},\ldots,x_{i_k}]$ est un chemin ayant au plus c-1 répétitions, d'où par hypothèse de récurrence c' qui est une sous-séquence de c posède une sous-séquence qui est un chemin élémentaire de x_{i_1} à x_{i_k} :

$$c=[x_{i_1},\ldots,(x_{i_u},\ldots,x_{i_v}),x_{i_v+1},\ldots,x_{i_k}]$$

Définitions (Graphe non orienté)

- une chaîne de G=(S,A) est une séquence de sommets $x_{i_1},\ldots,x_{i_k}$ $k \geq 1$ avec $\forall$ j $1 \leq u \leq v \leq k$

$$(x_{i_j},x_{i_{j+1}}) \in A \text{ ou } (x_{i_{j+1}},x_{i_j}) \in A$$

- une chaîne $(x_{i_1},\ldots,x_{i_k})$ est dite élémentaire si

$$\forall\ u,v \quad 1 \leq u \leq v \leq k \quad x_{i_u} \neq x_{i_v}$$

(ne contient pas deux fois le meme sommet)

- la longueur d'une chaine est égale au nombre de sommets moins un. La chaine constituée du seul sommet x_{i_1} sera notée (x_{i_1}) est de longueur nulle.

Définition

- un circuit de G=(S,A) est un chemin $[x_{i_1},\ldots,x_{i_k}]$ de G tel que $\forall k \geq 2$ et $x_{i_1}=x_{i_k}$.
- un cycle de G=(S,A) est une chaine $(x_{i_1},\ldots,x_{i_k})$ de G tel que $\forall k \geq 2$ et $x_{i_1}=x_{i_k}$.

Définition

Etant donné un graphe G=(S,A) et x∈S, on définit les ensembles de descendants et ascendants d'un sommet x :

- $desc_G(x)=\{y \in S \mid \exists$ [x,y] dans G$\}$
- $asc_G(x)=\{y \in S \mid \exists$ [y,x] dans G$\}$

=> $\forall$ x,y∈S, $y \in desc_G(x)$ <=> $x \in asc_G(y)$

Propriété

Soit G=(S,A) un graphe

$$\forall x \in S,\ asc_G(x)=desc_{G^{-1}}(x)$$

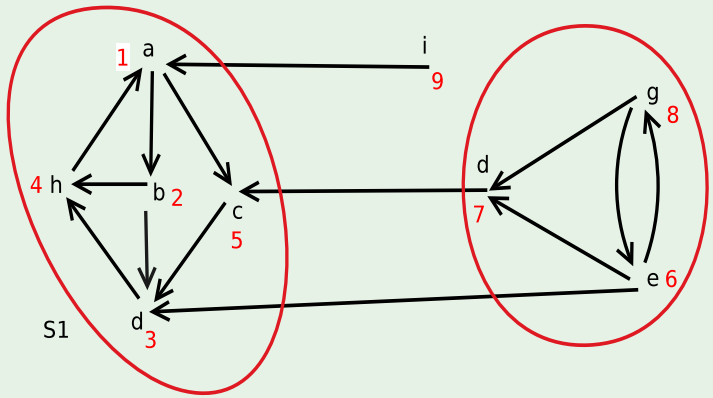
Définition

Une racine de G=(S,A) est un sommet r de S tel que $S=desc_G(r)$

Parcours en profondeur :

Exemple

Parcours en profondeur d'un graphe : les numéros correspondants aux sommets donnant l'ordre dans lequel les sommets sont visités.



algorithme :

```
procedure parcoursProfondeur(g : graphe);
    variable atteint : tableau[sommet] de booleens
    (atteint[x] <=> le sommet x a ete atteint)
    x : sommet

debut
   pour tout sommet x de g faire
      atteint[x]:=faux
   pour tout sommet x de g faire
      si non atteint[x] alors RechercheProf(x)
fin
```

```
procedure RechercheProf(x : sommet);
{specifications : etant donnee un sommet x non atteint, cette
procedure marque x, et tous les sommets y descendants de x
tels qu'il existe un chemin [x,y] dont aucun sommet n'est
marque}

   variable y : sommet

debut
   atteint[x]:=vrai
   pour tout successeur y de x faire
      si non atteint[y] alors RechercheProf(y)
fin
```

Complexité :

- La phase d'initialisation du tableau atteint est en $\theta(n)$.
- L'itération de l'algorithme principal est réalisé exactement en n étapes, pour chacune il y'a au moins un test réalisé.
- $\theta(n+m)$, soit $\theta(max(n,m))$ dans le cas des listes d'adjacences.
- $\theta(n^2)$ dans le cas des matrices d'adjacence.

Ordre de traitement-Numérotation : (parcours en profondeur)

L'objet des parcours de graphes concerne des traitements que l'on souhaite opérer sur les graphes. Les traitements s'opèrent parfois sur les sommets visités. Il est alors possible d'opérer un traitement avant ou après la visite du sommet. Il y'a donc soit un traitement en préOrdre ou ordre préfixé, soit en postOrdre ou ordre postfixé. Ceci se traduit par une modification de la procédure de recherche en profondeur.

traitement en préOrdre

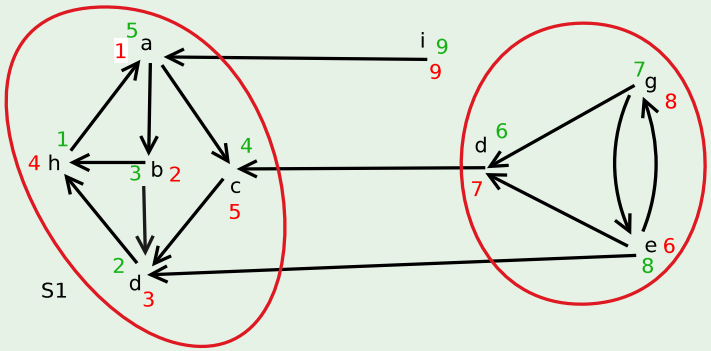
```
var y : sommet
debut
   atteint[x]:=vrai
   <traiter x>
   pour tout successeur y de x faire
      si non atteint[y] alors rechercheProf(y)
fin
```

traitement en postOrdre

```
var y : sommet
debut
    atteint[x]:=vrai
    pour tout successeur y de x faire
        si non atteint[y] alors rechercheProf(y)
    <traiter x>
fin
```

Exemple

numérotation en pré-ordre(rouge) et en post-ordre(vert) pour un parcours en profondeur



Mise en oeuvre en C du parcours en profondeur :

```
typedef struct cel{
  int st;
  struct cel* suiv;
}cellule

typedef cellule* Liste;

typedef struct{
  Liste a[N_MAX];
  int n;
}grapheL;

typedef int atteint[N_MAX];

/* variables globales déclarées:
   grapheL g;
   atteint m: */
```

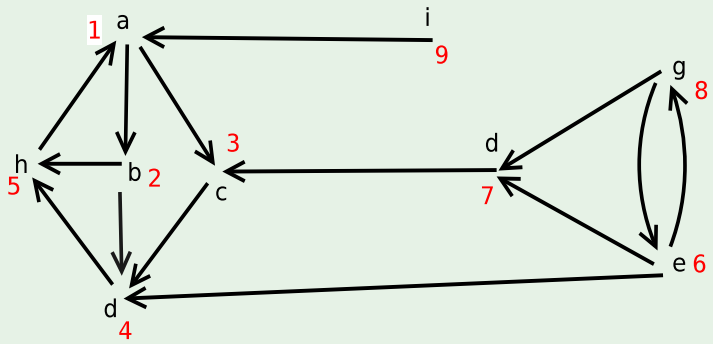
```
void parcoursProf(int x){
  Liste p;
  m[x]=1;
  p=g.a[x];
  while(p!=NULL){
    if(!m[p->st]) parcoursProf(p->st);
    p=p->suiv;
  }
}

void main(){
  int x;
  for(x=1;x<g.n;x++) m[x]=0;
  for(x=1;x<g.n;x++){
    if(!m[x]) parcoursProf(x);
  }
```

Parcours en largeur :

Dans ce parcours, lorsqu'un sommet x est atteint, tous ses successeurs y sont visités avant de visiter les autres descendants de x. Si l'on se réfère à la notion de distance, partant de x, le parcours va d'abord visiter tous les sommets situés à la distance 1 de x, puis tous les sommets situés à la distance 2 de x, et ainsi de suite jusqu'à ce que tous les sommets atteignables soient visités.

Exemple (parcours en largeur)

La numérotation indiquée correspond à un ordre de visite lors d'un parcours en largeur



principe de l'algo :

Il repose sur la notion de file.
Lors de la visite d'un sommet s, tous ses successeurs non encore atteint vont être rangés dans la file de manière à conserver la priorité liée aux distances depuis le sommet origine.

Mettons en oeuvre cet algo ! ! !

```
typedef struct{
  Liste tete,queue;
}File;

void enfiler(int x,File* f);
int defiler(File* f);
int fileVide(File* f);
```

```
void parcoursLarg(int x){
  Liste p;

  initFileVide(&f);
  enfiler(x,&f);
  m[x]=vrai;
  while(!fileVide(f)){
    x=defiler(f);
    p=g.a[x];
    while(p!=NULL){
if(!m[p->st]){
 m[p->st]=vrai;
enfiler(p->st,f);
}
      p=p->suiv;
    }
  }
}
```

```
void main(){
  int x;
  for(x=1;x<g.n;x++) m[x]=0;
  for(x=1;x<g.n;x++){
    if(!m[x]) parcoursLarg(x);
  }
}
```

complexité de l'algo :

- identique à celle du parcours en profondeur
- $\theta(n+m)$ pour les listes d'adjacence
- $\theta(n^2)$ pour les matrices d'adjacence

Quelques applications des parcours

- Obtention des descendants par un parcours en profondeur. Il suffit d'insérer dans un ensemble, les sommets dès qu'ils sont atteints. Outre le tableau atteint, on utilise donc une structure de données d'ensemble.

```
Type desc : ensemble de sommets
var atteint : tableau[sommet] de bools

procedure descProf(x : sommet,desc : ens de sommets);
   var y : sommet;
   debut
      atteint[x]:=vrai
      desc:=desc U {X}
      pour tout successeur y de x faire
         si non atteint[y] faire descProf(y,desc)
   fin
```

complexité de l'algo :
identique à celle du parcours en profondeur du sous-graphe induit par les descendants de x, soit avec n_x et m_x le nombre de sommets et d'arcs pour ce sous-graphe.
- $\theta(n_x+m_x) -> $ listes d'adjacence pire cas : $O(n+m)$
- $\theta(n_x.m) -> $ matrice d'adjacence pire cas : $O(n^2)$

- Obtention des ascendants pour un parcours en profondeur.
  La recherche des ascendants d'un sommet peut se faire en utilisant la procédure descProf avec pour paramètre le sommet considéré et le graphe réciproque de G, soit G^{-1}

Application globale : Fermeture reflexo-transitive d'un graphe

Définition

Soit G=(S,A) un graphe, la fermeture réflexo-transitive de G est notée $G^t=(S,A^t)$ et elle vérifie la propriété suivante :

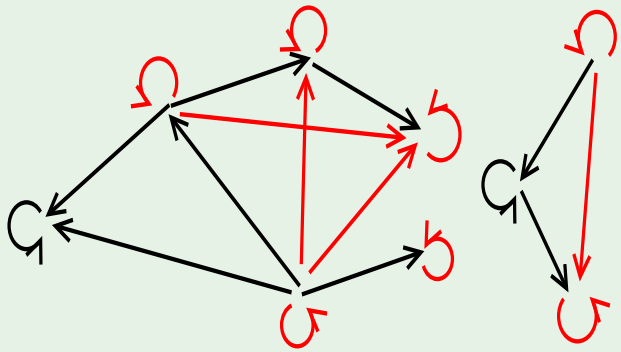
$$(x,y)\in A \Leftrightarrow x=y \text{ ou } \exists\ [x,y] \text{ dans } G$$

On peut vérifier que l'on a bien les propriétés suivantes :

- $(x,y)\in A \Rightarrow (x,y)\in A^t$
- $(x,y)\in A$ et $(y,z)\in A \Rightarrow (x,z)\in A^t$
- $\forall x\in S\ (x,x)\in A \Rightarrow (x,y)\in A^t$

Exemple

les arcs ajoutés par réfléxivité et transitivité



Calcul de G^t à partir de G :

principe :

Il s'agit d'ajouter les arcs (x,y) pour tout y tel que [x,y] est dans G ie pour tout élément y descendants de x. Il en découle que dans G^t les successeurs de x sont les descendants de x dans G.
On utilise ainsi la procédure de calcul des descendants.

```
procedure fermetureRefTrans(g : graphe,gt : graphe)
var atteint : tableau[sommet] de booleens
    desc    : ens de sommets
    x,y     :  sommet

debut
    {initialisations}
    pour tout sommet x de G faire atteint[x]:=faux
    desc:=ens_vide
    pour tout sommet x de G faire
        descProf(x,g,desc)
        pour tout sommet y de desc faire
            ajouterArc(x,y,gt)
            desc:=desc-{y}
            atteint[y]:=faux
        fin pour
    finpour
fin
```

complexité :

Nous donnons ici seulement une majoration de la complexité.
On étudie d'abord le cas d'une représentation matricielle :
on a vu que le cout de la procédure descProf(x,g,desc) est en $\theta(n_x.m)$. On réalise ce traitement n fois.
On obtient ainsi une majoration par $O(n^3)$

Dans le cas des listes d'adjacence, on obtient une majoration par $O(n(n+m))$

1 Graphe et algorithme : présentation

2 Parcours, numérotation et descendance

3 Connexité et forte connexité
- Composantes connexes
- Forte connexité

4 Graphes sans circuit

5 Problème du plus court chemin

Le problème de la recherche de composante connexe est un problème d'un intérêt pratique majeur dans de nombreuses applications liées à toutes sortes de réseau : on veut savoir quels sont les sommets qui sont reliés sans chercher à savoir explicitement comment.

La notion de connexité est liée à l'existence de chaines, celle de forte connexité à celle de chemins.

Connexité et forte connexité

Composantes connexes

Définition

Soit G=(S,A) un graphe

On définit la relation de connexité sur S, notée R_c par :

$$xR_cy <=> x=y \text{ ou } \exists \text{une chaine } (x_{i1}=x,\ldots,x_{ik}=y) \text{ dans } G$$

Propriété

R_c est une relation d'équivalence

preuve :

Par définition des chaines, R_c est bien une relation qui est :

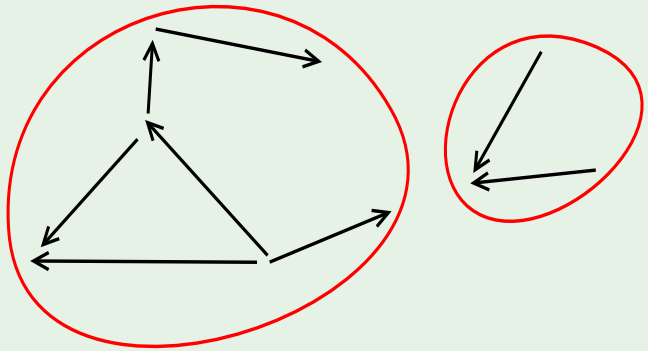
- réflexive : xR_cx
- symétrique : $xR_cy <=> yR_cx$
- transitive : xR_cy et $yR_cz => xR_cz$

Définition

Soit G=(S,A) un graphe. Une classe d'équvalence de S modulo R_c est appellée composante connexe de G.
Le graphe G est dit connexe s'il ne possède qu'une seule composante connexe.

Exemple

Graphe avec deux composantes connexes



Recherche de composantes connexes

La notion de connexité est non orientée, on va donc travailler sur G_S, le graphe symétrisé de G=(S,A) qui est défini par G_S=(S,A∪A^{-1}).
On s'appuie par ailleurs sur les deux propriétés suivantes :

Propriété

Les composantes connexes d'un graphe G sont les mêmes que celles de G_S

preuve :

∃ une chaine (x_{i1}=x,...,x_{ik}=y) dans G <=> ∃ une chaine (x_{i1}=x,...,x_{ik}=y) dans G_S

Propriété

Etant donné un graphe G=(S,A)
∀x∈G $desc_{G_s}$(x)=composante connexe de x dans G_S

preuve :

trivial (montrer la double inclusion)

Un schéma d'algorithme découle immédiatement de cette dernière propriété

algo :

On utilise un marquage des sommets avec la structure suivante :
marqué : tableau[sommet] de booléens avec marqué[x] <=> le sommet x a été traité

1) calcul de G_S
2) pour tout sommet x de G faire marqué[x] :=faux finpour
3) tant que $\exists s \in S$/marqué[s]=faux faire
   marqué[x] :=vrai
   calculer $desc_{G_S}(x)$ il s'agit d'une nouvelle composante connexe
   pour tout sommet y de $desc_{G_S}(x)$ faire marqué[y] :=vrai
   fintq

La justification de cette algorithme découle naturellement des propriétés précédantes.

complexité :

- Le coût de la symétrisation d'un graphe est en $\theta(n+m)$
- La seconde étape est linéaire dans le nombre de sommets du graphe : $\theta(n)$
- Considérons la troisième étape : soit n_k et m_k le nombre de sommets et le nombre d'arcs dans la k ième composante connexe calculée : pour chaque composante connexe, le coût est de l'ordre de (n_k+m_k) comme pour le calcul des descendants d'un sommet.
  On obtient aussi globalement un coût de l'ordre :

$$\sum_{1 \leq k \leq |C|} (n_k+m_k) = n+m$$

  (C : l'ensemble des composantes connexes)

D'où la complexité $\theta(n+m)$ pour les listes d'adjacence

De manière similaire $=>$ $\theta(n^2)$ pour les matrices d'adjacence.

Connexité et forte connexité

Forte connexité

Définition

Soit G=(S,A) un graphe

On définit la relation de forte connexité sur S, notée R_{fc} par :

$$xR_{fc}y \Leftrightarrow x=y \text{ ou } \exists \text{ des chemins } [x,y] \text{ et } [y,x] \text{ dans } G$$

Propriété

R_{fc} est une relation d'équivalence

preuve :

La même que pour la notion de connexité R_c (relation d'équivalence)

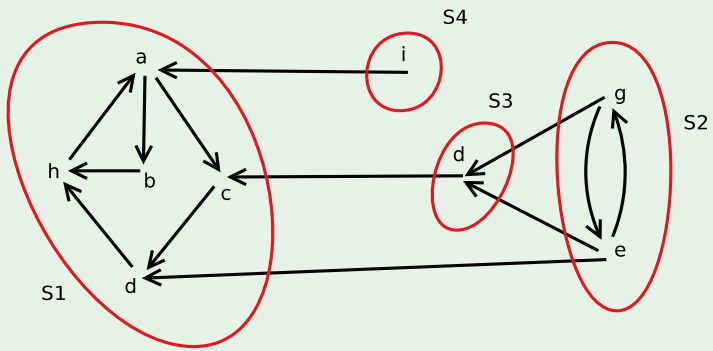
Définition

R_{fc} induit sur G=(S,A) une partition de S.
On dit que S est partitionnée en composantes fortement connexes. (cfc)

Définition

Un graphe est dit fortement connexe s'il possède une seule composante fortement connexe.

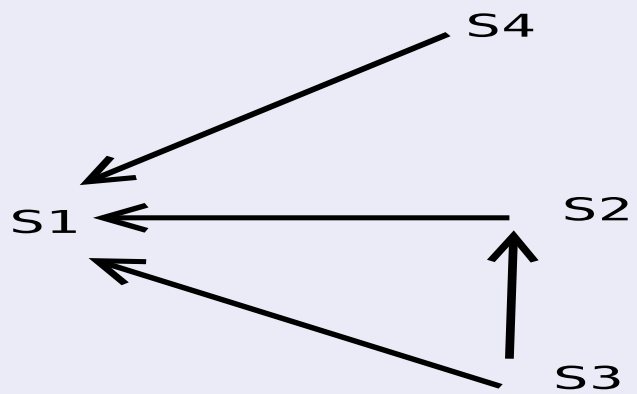
Exemple



Remarques

- Dans le cas d'un graphe modélisant un réseau de flux d'informations on pourra constater que :
  - à l'intérieur d'une cfc, il y'a une bonne circulation de l'information
  - entre cfc, l'information ne transite au mieux que dans un sens
- Les cfc d'un graphe G sont les mêmes que celles de G^{-1}
- Les cfc d'un graphe forment une partition de l'ensemble des sommets de G. On peut en déduire un graphe quotient, nécessairement snas circuit.

Graphe quotient induit par les cfc de G :



Recherche de composantes connexes :

1) Solution brutale

Cette solution est basée sur la propriété suivante :

Propriété

Etant donné un graphe G=(S,A)
si cfc(x) est la composante fortement connexe qui inclut x alors

$$cfc(x) = desc_G(x) \cap asc_G(x)$$

preuve :

Par définition de R_{fc}, $desc_G(x)$, et $asc_G(x)$ (fin preuve)

On en déduit un schéma d'algorithme utilisant un ensemble de sommets (classé) initialisé à vide, et devant à la fin de l'algorithme, contenir l'ensemble des sommets du graphe.

algorithme :

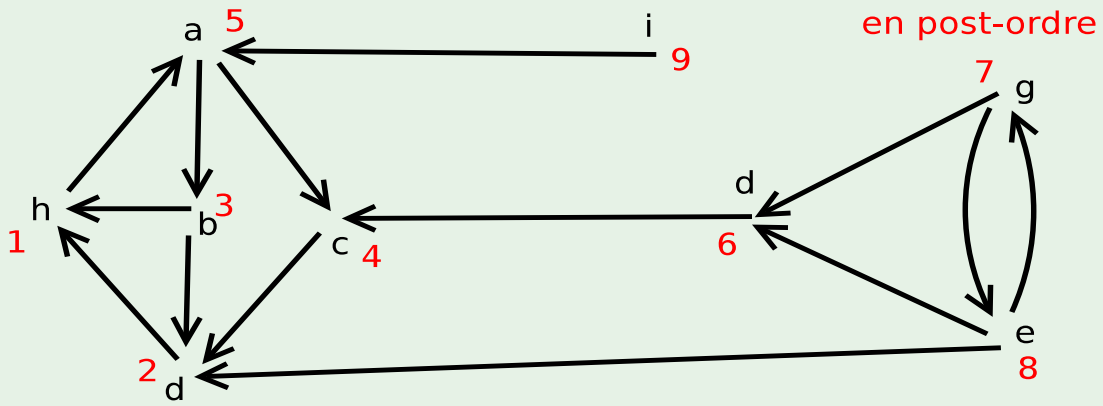
1) Calcul de G_{-1} ; classé := ; i :=0
2) tant que $\exists$ x$\in$S / x$\notin$classé faire
3) i :=i+1
4) S_i :=$desc_G$(x) $\cap$ $desc_{G^{-1}}$(x)
5) classé :=classé $\cup$ S fintq

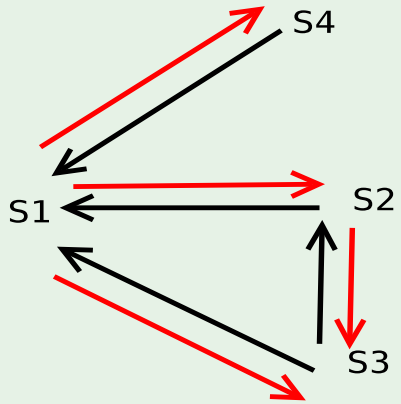
complexité :

- Dans le cas d'une représentation par liste d'adjacence, l'étape 1) est réalisable en θ(n+m)
  L'instruction 4) coûte θ(n+m) à cause des calculs de descendants, tandis que l'étape 5) possède un coup majoré par le nombre de sommets.
  L'instruction peut-être executée n fois, ce qui fait que l'on arrive à un coût global de θ(n(n+m))
- En suivant le même raisonnement, on obtient un coût global de $\theta(n^3)$ avec une représentation par matrices d'adjacence.

2) Solution linéaire basée sur le parcours de graphe

Exemple





1. Graphe et algorithme : présentation
2. Parcours, numérotation et descendance
3. Connexité et forte connexité
4. Graphes sans circuit
   - Définitions
   - Quelques exemples classiques
   - S-séquence d'un graphe - Décomposition par niveaux
5. Problème du plus court chemin

Graphes sans circuit

Définitions

Exemple

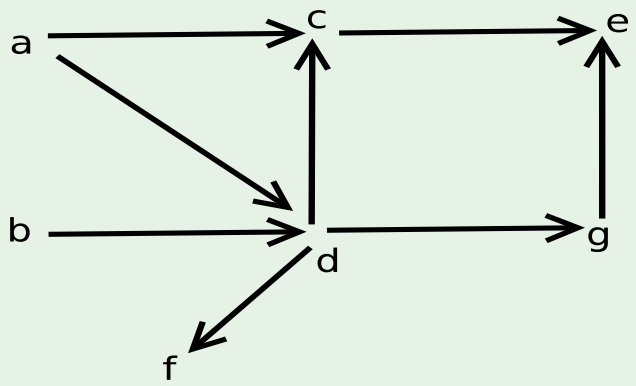
Un graphe sans circuit, mais cyclique.
On peut remarquer que la notion de circuit est liée aux graphes orientés, tandis que la notion de cycle ne considère pas l'orientation des arcs. Ceci est naturellement du aux définitions de chemin et de chaîne sous-jacentes à celles de circuit et de cycle.

Rappels

Un circuit de G=(S,A) est un chemin $[x_{i0},x_{i1},\ldots,x_{ik}]$ tel que $x_{i0}=x_{ik}$
Un cycle de G=(S,A) est une chaîne $(x_{i0},x_{i1},\ldots,x_{ik})$ tel que $x_{i0}=x_{ik}$

Exemple

Graphe orienté sans circuit



il existe plusieurs cycles dans ce graphe, mais aucun circuit

cycles : (a,c,d,a) ;(e,c,d,g,e) ;...

circuit : $\emptyset$

Propriété

Tout sous-graphe ou graphe partiel d'un graphe sans circuit (sans cycle) et sans circuit (sans cycle).

preuve :

démo triviale par contraposée.

Définition

Un circuit (cycle) élémentaire est un circuit (cycle) qui ne possède qu'une seule répétition, le premier et le dernier sommet.

Propriété

De tout circuit, on peut extraire un circuit élémentaire.

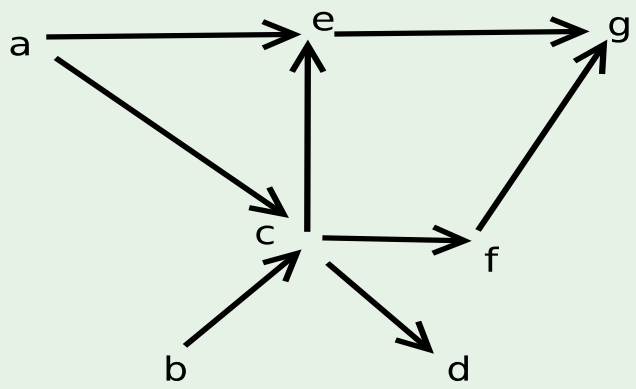
preuve :

Voir celle pour les chemins.

Définition

Etant donné un graphe G, une source (resp un puits) de G est un sommet de G n'ayant aucun prédécesseur (resp aucun successeur).
L'ensemble des sources est noté sources(G)=$\{s \in S \ / \ d^-(s)=0\}$
L'ensemble des sources est noté puits(G)=$\{s \in S \ / \ d^+(s)=0\}$

Exemple



sources(G)=a,b
puits(G)=d,g

Propriété

Tout graphe sans circuit possède au moins une source et un puits.

preuve :

Considérons un chemin c de G qui soit maximal au sens suivant :
$c=[x_{i1},x_{i2},\ldots,x_{ik}]$ et il n'existe pas de sommet y de G tel que $[y,x_{i0},x_{i1},\ldots,x_{ik}]$ ou $[x_{i0},x_{i1},\ldots,x_{ik},y]$ soient de schemins de G.
Un tel chemin existe puisque G est sans circuit.
Cela signifie que x_{i1} est une source et x_{ik} est un puits.

Propriété

G est sans circuit ssi G^{-1} est sans circuit.
Les sources (resp les puits) de G sont les puits (resp les sources) de G^{-1}.

preuve :

trivial (juste utiliser les définitions)

Graphes sans circuit

Quelques exemples classiques

Graphe d'une relation d'ordre strict

Une relation d'ordre strict est une relation :

- transitive
- irréflexive

Etant donné R une relation d'ordre strict sur un ensemble fini S, on peut associer à R un graphe G=(S,A) avec $A=\{(x,y)\in S^2 \mid xRy\}$

Propriété

Un tel graphe est nécessairement sans circuit.

preuve : par l'absurde

Si $c=[x_{i1},\ldots,x_{ik-1},x_{ik}]$ est un circuit élémentaire, on a donc $x_{i1}Rx_{i2}R\ldots Rx_{ik}$
On en déduit par transitivité de R que $x_{i1}Rx_{ik}$, d'où $x_{i1}Rx_{i1}$ (car $x_{i1}=x_{ik}$)

Contradiction ! ! ! (car R irreflexive)

Graphe de précédence

Lors du début de ce chapitre, nous avons vu un exemple de modélisation de taches soumises à des contraintes de précédence.
(certaines taches doivent s'exécuter avant d'autres)

L'existence d'un circuit dans un tel graphe signifie l'impossibilité qu'il y'a d'accomplir la tache.

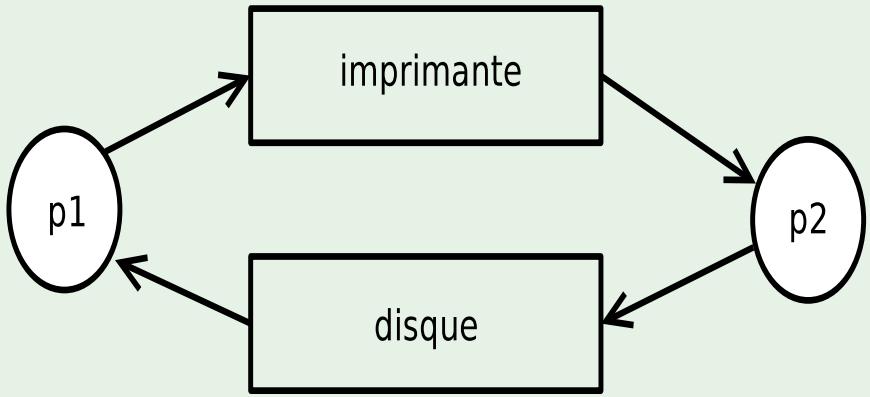
Allocation de ressources

On peut modéliser certains problèmes d'allocations de ressources dans des systèmes multi-processus par un graphe d'allocation dont les sommets sont de deux types :

- processus
- ressources (une mémoire centrale partagée, un disque, une imprimante,...) et les arcs sont de deux types :
  - (p,r) qui signifie que le processus p est en attente de la ressource r.
  - (r,p) qui signifie que la ressource r est allouée au processus p.

L'existence d'un circuit dans ce graphe est équivalent à la présence d'une situation d'interblocage.

Exemple



Graphes sans circuit

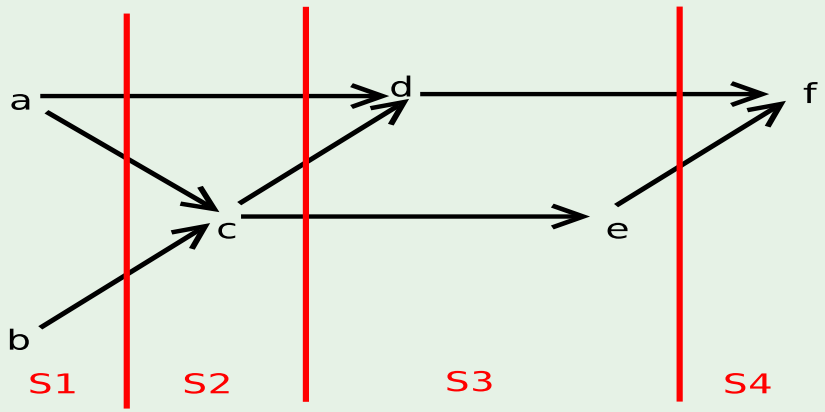
S-séquence d'un graphe - Décomposition par niveaux

Définition

Soit un graphe G=(S,A), la S-séquence de G est l'ensemble $\{S_1,S_2,...,S_k\}$ des parties non vides de G tel que :

- S_1=sources(G)
- S_i=sources(G\(S-$S_1 \cup S_2 \cup ... S_{i-1}$))$\neq \emptyset$ pour $2 \leq i \leq k$
- S_{k+1}=sources(G\(S-$S_1 \cup S_2 \cup ... S_k$))=$\emptyset$

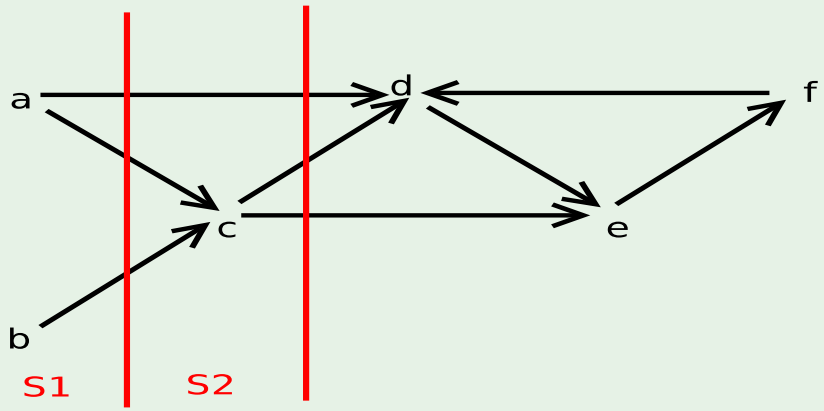
Exemple(partition en S-séquence)



Remarque

La partition en S-séquence des sommets d'un graphe avec circuit est impossible.

Exemple



sources($G \setminus (S - S_1 \cup S_2)$)=$\emptyset$
$S_1 \cup S_2 = \{a,b,c\}$

Propriété

Le graphe G=(S,A) est sans circuit ssi
la S-séquence de G est une partition de S

preuve :

=>

Par construction, les S_k sont disjoints 2 à 2.
Il reste à prouver que S est la réunion des S_k.
On considère le sous-graphe induit par S-$S_1 \cup ... \cup S_k$.
Comme G est sans circuit, ce graphe l'est aussi.
S'il n'est pas vide, il a donc au moins une source.
Or S_{k+1}=sources(G\(S-$S_1 \cup ... \cup S_k$))=$\emptyset$
=> S-$S_1 \cup ... \cup S_k$=$\emptyset$
=> S=$S_1 \cup ... \cup S_k$

<=

La S-séquence de G est une partition de S.
Supposons su'il existe un circuit [x_1,...,x_j=x_{i1}]
Si $x_j \in S_{p_j}$, on a necessairement $p_1 < p_2 < ... < p_j$ et $p_1 < p_1$
donc p_j=p_1
et donc on a : $p_1 < p_2 < ... < p_1$ => p_1=p_1
Contradiction ! ! !

Calcul d'une S-séquence d'un graphe

L'idée de l'algorithme est en fait basée sur la définition même de la S-séquence :
les sommets de S_i figurent parmi les successeurs de S_{i-1}, il suffit à chaque étape de supprimer les sommets de S_{i-1} en mettant à jour le degré intérieur des sommets succeseurs (de S_{i-1}), et de considérer ensuite comme sommet de S_i, les sommets de degré intérieur nuls. Les informations calculées sont la S-séquence.

```
type S-sequence = structure
     S:tableau[sommet] de ens de sommets {S[i]=Si}
     k:entier {le nb de parties de la S-sequence}

fonction sources(g:graphe):ens de sommets
{specif : a pour valeur l'ens des sources de G}
```

```
procedure calcul_s_seq(g:graphe,circuit:booleen,
       s-seq:s_sequence);

{specif : en sortie, si circuit est vrai alors le
graphe G possede au moins un circuit ;
sinon circuit est faux, et s-seq est la S-seq de G}

variable x,y:sommet
i,nbsom:entier {nbsomm=nb de sommets deja traites}
deg:tableau[sommet] de 0..n {pour le degre interieur}
```

```
debut <calcul des degr
   pour i allant de 1 .n faire s-seq[i]:=ens_vide
   s-seq.S[1]:=sources(g)
   s-seq.k=1
   nbsom:=|s-seq.S[1]|
   tant que (s-seq.S[s-seq.k]!=0)faire
      pour tout sommet x de s-seq.S[s-seqk] faire
         pour tout succeseur y de x faire
            deg(y):=deg(y)-1
            si deg(y)=0 alors
               s-seq.S[s-seq.k+1]=s-seq.S[s-seq.k+1]U{y}
               nbsom:=nbsom+1
            finsi
         fpour
      fpour
      s-seq.k:=s-seq.k+1
   ftq
   circuit:=(nbsom<g.n)
```

1 Graphe et algorithme : présentation

2 Parcours, numérotation et descendance

3 Connexité et forte connexité

4 Graphes sans circuit

5 Problème du plus court chemin
- Introduction
- Cas des valuations positives : algorithme de Dijkstra
- Cas des valuations quelconques et graphes sans circuits : algorithme de

Problème du plus court chemin (PLC)

Introduction

Problème :
Dans cette partie, nous traitons des problèmes du plus court chemin. Pour modéliser les situations réelles à la base de ce type de problèmes ; nous introdusons la notion de graphe orienté.

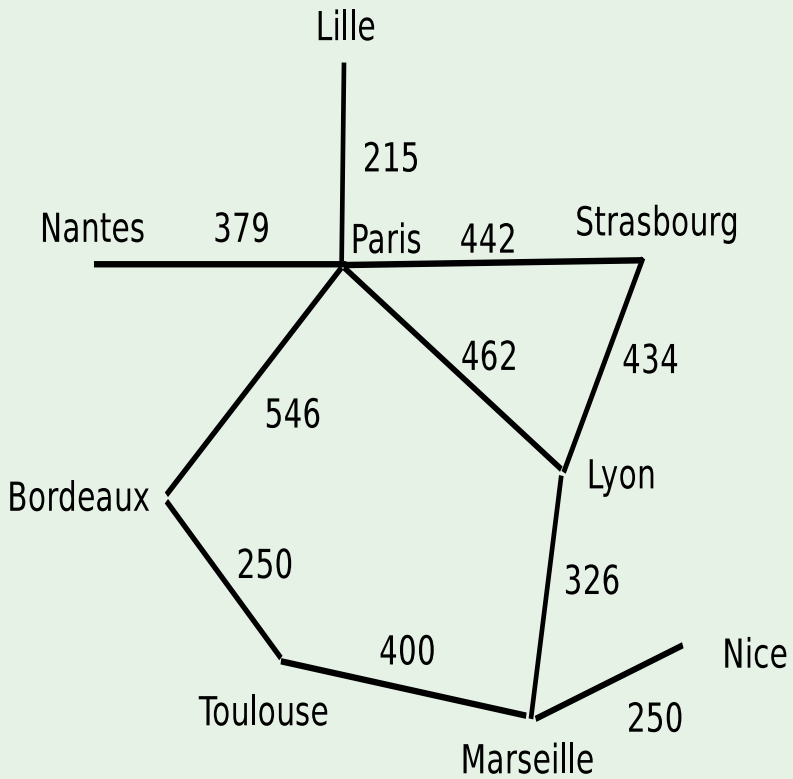
Définition

Un graphe valué est noté G=(S,A,v) où :

- (S,A) est un graphe orienté
- v est une fonction de coût (on dit aussi un étiquetage, un poids, une valuation des arcs) qui à tout arc a=(x,y) associe un réel v : A$\longrightarrow \mathbb{R}$

Exemple

Un réseau de transport avec les coûts (ici les distances) entre chaque ville.



Définition

Le coût cumulé (ou valeur) d'un chemin c, que l'on note coût(c) est égal à la somme des coûts de chacun des arcs qui le composent.

Exemple

En reprenant l'exemple précédant, si c=[Lille,Paris,Lyon,Strasbourg]
coût(c)=215+462+434=1111

Remarque

Il est préférable d'étendre le domane de définition de v à S^2, en supposant pour les couples de sommets ne figurant pas dans A que leur valuation vaut $+\infty$ (ou $-\infty$) ce qui revient à définir une extension v' de v :

$$v' : S^2 \rightarrow \mathbb{R}$$
$$(x,y) \rightarrow v'(x,y)=v(x,y) \text{ si } (x,y) \in A$$
$$\text{et } +\infty \text{ sinon}$$

Plusieurs problèmes (d'optimisation) peuvent alors se poser :

- problème 1 : recherche d'un chemin de cout minimum (ou maximum) entre deux sommets x et y donnés
- problème 2 : recherche de l'ensemble des chemins de cout minimum (ou maximum) entre un sommet x et les autres osmmets du graphe
- problème 3 : recherche de l'ensemble de chemins de cout minimum (ou maximum) entre tous les couples de sommets

Le problème 1 est généralement appelé "problème du plus court chemin".

Définition

On appelle plus court chemin (plc) de x à y, un chemin de valeur minimale. La distance entre deux sommets x et y est égale au cout du plus court chemin entre x et y.

Question :

A quelle condition le problème 1 admet-il une solution ?

- Il faut déjà qu'il existe un chemin entre x et y :
  ceci peut être résolu par le calcul des descendants de x.
- Dans le cas de valuations négatives, le problème 1 ne peut pas posséder de solution :
  soient $c=[x_1, ..., x_i, ..., x_j=x_i, x_{j+1}, ..., x_k]$
  $c'=[x_i,...,x_j=x_i]$ un sous-chemin de c qui est un circuit
  et $c''=[c=[x_1,...,x_i,x_{j+1},...,_k]$
  on a alors :

$$cout(c)=cout(c')+cout(c'')$$

  si cout(c')¡0, il n'existe pas de plus court chemin de x à y.

  Puisque c peut etre étendu à tout chemin empruntant plusieurs fois le circuit c', et ce infiniment, en obtenant à chaque fois un chemin de cout inférieur. Les circuits tels que c' (cout(c')<0) sont appelés circuits absorbants.

Remarque

Dans le cas des graphes possédant des circuits absorbants, on pourrait ramener le problème 1 à celui de la recherche de chemins élémentaires de coût minimum, mais on ne connait pas d'algorithme de complexité raisonnable (polynomiale) pour ce dernier.

=> Dans le cadre de cette partie du cours, on se limitera aux graphes sans circuits absorbants.

Application : choix d'une politique optimale :

Il s'agit ici de définir par une entreprise, une politique de remplacement d'équipements informatiques qui soit la meilleure en terme de cout.

"Une entreprise a besoin d'une nouvelle machine pour remplir un contrat de 5 ans. Le cout d'une machine neuve est de :

- 300 unités monétaires (UM) aux années 0 et 1
- 350 UM aux années 2 et 3
- 400 UM à l'année 4

Deux facteurs sont également à considérer :

- le cout de maintenance
- le prix de revente

nb années utilisation	prix de revente	cout maintenance cumulé
1	200	0
2	150	50
3	100	100
4	50	200
5	0	300

Question :

Quelle est la politique optimale (la moins couteuse) pour avoir pendant 5 années une machine en état de fonctionnement ?

Modélisation par un graphe valué :

G=(S,A,v)

- S={0,1,2,3,4,5} les années
- A={(i,j) | i<j}
- v(i,j)=<cout d'achat l'année i>
  +<cout de maintenance pendant j-i années>
  -<prix de revente après j-i années d'utilisation>

exemple :

v(2,4)=350+50-150=250

Un plus court chemin correspond à une politique optimale.
On trouve plusieurs politiques optimales d'un cout de 550 UM.

Par exemple [0,2,5] est un chemin optimal, son cout 550.
On peut l'interpréter :

- achat d'une machine à l'an 0 $\longrightarrow$ 300
  - 2 années de maintenance $\longrightarrow$ $+50 = 350$
  - revente à l'année 2 $\longrightarrow$ -150 $= 200$
- achat d'une machine à l'année 2 $\longrightarrow$ $+350 = 550$
  - 3 années de maintenance $\longrightarrow$ $+100 = 650$
  - revente à l'année 5 $\longrightarrow$ -100 $= 550$

Le chemin [0,1,5] est un également optimal, (cout : 550).
On peut l'interpréter :

- achat d'une machine à l'an 0 $\longrightarrow$ 300
  - 1 année de maintenance $\longrightarrow$ $+0 = 300$
  - revente à l'année 1 $\longrightarrow$ -200 $= 100$
- achat d'une machine à l'année 1 $\longrightarrow$ $+300 = 400$
  - 4 années de maintenance $\longrightarrow$ $+200 = 600$
  - revente à l'année 5 $\longrightarrow$ -50 $= 550$

Problème du plus court chemin

Cas des valuations positives : algorithme de Dijkstra

Cet algorithme repose sur l'hypothèse suivante :

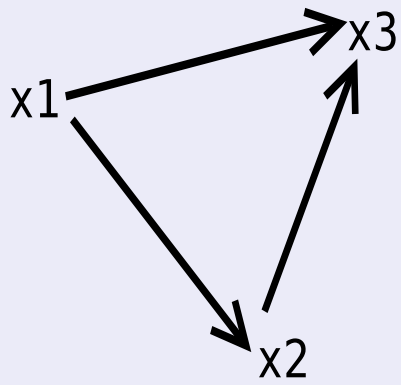
La valuation de chaque arc est positive.
L'absence d'arc entre deux sommets se traduit par une valuation infinie. Le problème traité est le problème 2 :
On recherche les chemins optimaux entre un sommet x_i et tous les autres.
En fait, ceci est justifié par la remarque suivante :

"on ne sait pas résoudre le problème 1 plus efficacement que le problème 2"

Principe de l'algorithme :

Remarque

Si on considère trois sommets x_1, x_2 et x_3 tels que (x_1,x_3), (x_1,x_2) et (x_2,x_3) sont des arcs du graphe :

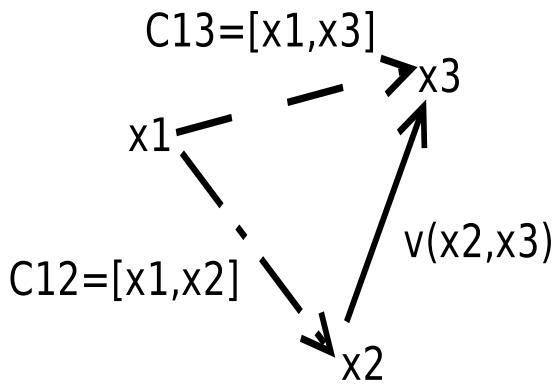


Si l'on veut le plus court chemin entre x_1 et x_3, et si on a :

$$v(x_1,x_3) > v(x_1,x_2) + v(x_2,x_3)$$

alors le plus petit chemin est $[x_1,x_2,x_3]$

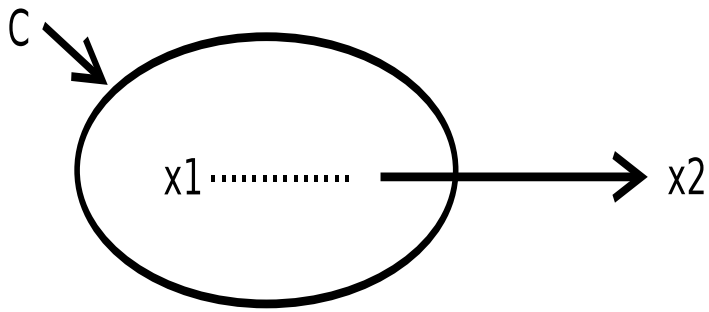
Cette remarque peut se généraliser à des chemins :
$C_{12}=[x_1,x_2]$ et $C_{13}=[x_1,x_3]$



et si l'on a $Cout(C_{13})>Cout(C_{12})+v(x_2,x_3)$
alors le plus court chemin est le chemin C_{12} augmenté de (x_2,x_3) soit $[x_1,x_2,x_3]$

L'algorithme de Djikstra repose sur la remarque précédante et sa généralisation :
On procède itérativement en chosissant à chaque étape un nouveau sommet.

- à chaque étape, un sommet est chosi puis inséré dans C :
  il s'agit d'un sommet pour lequel on connait la distance de x_1 à ce sommet.
- on appelera chemin special, un chemin de x_1 vers un autre sommet x_2, qui ne passe que par des sommets de C

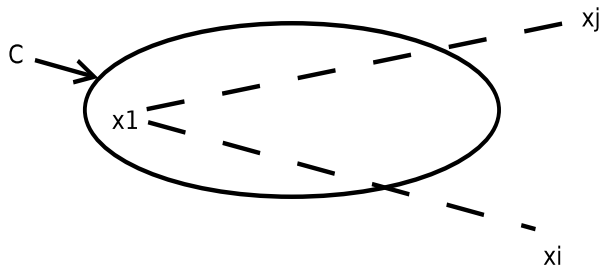


- à chaque étape de l'algorithme un tableau D (D pour distance) mémorise le coût du plc special de x_1 vers chaque sommet du graphe.
- quand un nouveau sommet est ajouté à C, le plc spécial de x_1 à ce sommet est aussi un plc.

Donc, quand tous les sommets ont été insérés dans C, on connait le coût de chaque plc spécial, donc de chaque plc.

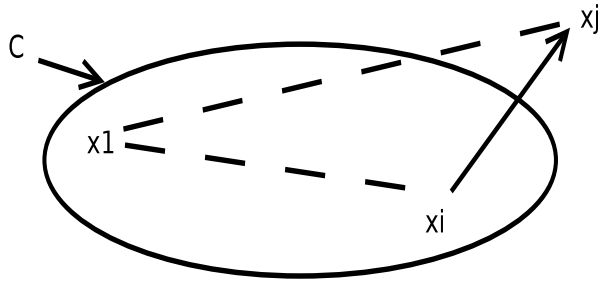
A chaque étape :

- choix d'un nouveau sommet à insérer dans C :
  sommet $x_i \in$ S-C, dont la valeur $D[x_i]$ est minimum



- La valeur de $D[x_j]$ est mise à jour pour tous les sommets x_j extérieurs à C, pour le cas où l'ajout à C de x_i engendre un chemin spécial de coût inférieur ie :

$$D[x_j] > D[x_i] + v(x_i, x_j)$$

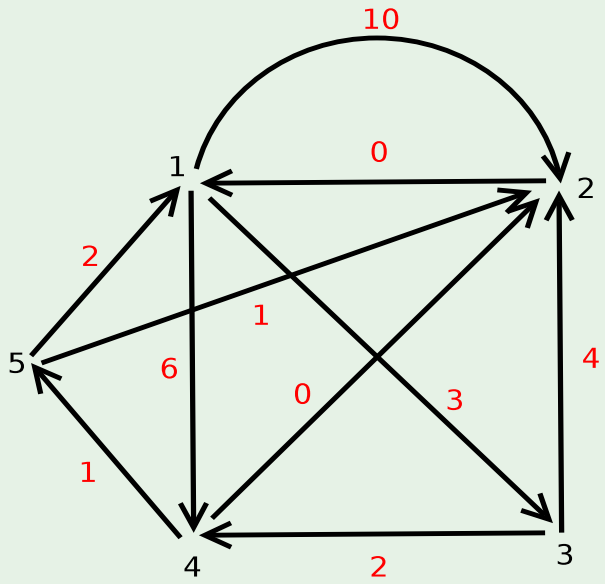


- on choisit par défaut 1 comme sommet initial.

Algorithme de calcul des distances :

```
variables i,x,y:entiers
          D:tableau[sommet] de reel
          C:ens de sommet

debut
   1) C<-{1}
   2) pour tout sommet x de G faire D[x]=v(1,x) fpour
   3) pour i allant de 2 a n-1 faire
   4)       choisir x dans S-C en minimisant D[x]
   5)       C<-C U {x}
   6)       pour tout sommet y de S-C faire
   7)          D[y]<-min(D[y],D[x]+v(x,y))
            fpour
      fpour
```

Exemple



- D=[0,10,$\underline{3}$,6,$+\infty$]
  C={1}
  => choix de 3 d'où C={1,3}
- D=[0,3+4,3,3+2,$+\infty$]=[0,7,3,$\underline{5}$,$+\infty$]
  C={1,3}
  => choix de 4 d'où C={1,3,4}

- D=[0,5+0,3,5,5+1]=[0,5,3,5,6]
  C={1,3,4}
  =>choix de 2 d'où C={1,3,4,2}
- D=[0,5,3,5,6]

preuve de l'algo :

On montre la validité de cet algorithme en considérant les propriétés suivantes comme invariants de boucle :

1. si un sommet x est danc C, alors D[x] est le coût du plus court chemin de 1 à x
2. si un sommet x n'est pas dans C, alors D[x] est le coût du plus court chemin spécial de 1 à x

Initialement :

La propriété (1) est satisafaite car C=$\{1\}$ puisque D[1]=0

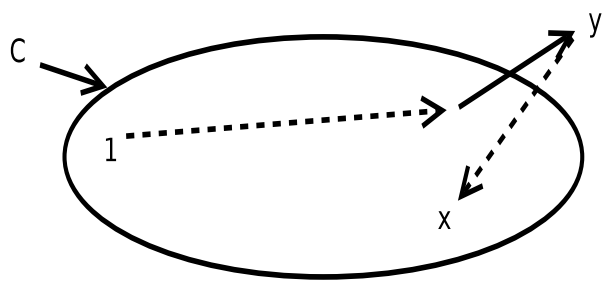
Pour la propriété (2), les chemins spéciaux sont réduits aux arcs (1,x), donc, compte tenu de l'initialisation, cette assertion est vérifiée

Quand le sommet x est considéré (étape x) :

propriété (1) :

Cette propriété est trivialement vérifiée pour les sommets de C avant ajout de x car la valeur de D n'est pas modifiée pour ces sommets.
Considérons x ; on doit vérifier que D[x] est le coût d'un chemin optimal.
Par hypothèse, on sait que D[x] est le coût du plus court chemin spécial de 1 à x. Il suffit de montrer que le plus court chemin de 1 à x est dans C (c'est un chemin spécial). Ceci revient à montrer que le plus court chemin de 1 à x ne passe par aucun sommet situé hors de C ; on le montre par l'absurde.

Soit y le premier sommet hors de C par lequel passe le plus court chemin de 1 à x :



Il est clair que [1,y] est un chemin spécial ; par ailleurs, c'est un plus court chemin spécial car le chemin [1,...y,...x] considéré est un plus court chemin de 1 à x. On a donc :

$$\text{coût}([1,...y,...x]) \geq \text{coût}([1,y])$$

car par hypothèse pour l'algorithme de Dijkstra, $\forall i, j$, v(i,j) ≥ 0, et d'après l'hypothèse (2) vraie en entrée, donc pour y :

$$\text{coût}([1,x]) \geq D[y]$$

On peut de plus affirmer l'inégalité D[y] $\geq$ D[x] puisque c'est x qui a été choisi dans S-C, pour minimiser D. On arrive finalement à :

$$\text{coût}([1,...y,...x]) \geq D[x]$$

Donc le chemin interne à C est un chemin optimal, puisque de coût inférieur à celui de tout chemin optimal.

propriété (2) :

Considérons $y \notin C$. On doit montrer que D[y] est le coût du plus court chemin spécial de 1 à y. Soit un chemin de la forme [1,y], quand x est ajouté à C, on a trois possibilités :

- [1,y] reste inchangé, ce chemin spécial ne passe pas par x. La propriété (2) est donc toujours vérifiée puisque D[y] n'est mis à jour que si on a $D[y] > D[x] + v(x,y)$, ce qui ne peut se présenter que si le plus court chemin spécial passe par x.
- [1,y] passe maintenant par x, le chemin est de la forme [1,...x,y], ie x est l'avant dernier sommet du chemin où seul y n'est pas dans C. Ce cas est traité par l'algorithme lors de l'exécution de l'instruction "$D[y] := \min(D[y],D[x]+v(x,y))$".

- [1,y] passe maintenant par x, le chemin est de la forme [1,...x,...,z,y] ou $z \neq x$ et seul y n'est pas dans C. Ce cas peut etre éliminé puisque z a déjà été inséré dans C avant que x ne le soit, et donc que D[z] a pour valeur la distance de 1 à z, et que nécessairement, le cout du chemin [1,...,x,...,z,y] ne peut etre inférieur strictement à D[z]+v(z,y) qui a déjà été considéré lors de l'insertion de z dans l'ensemble C.

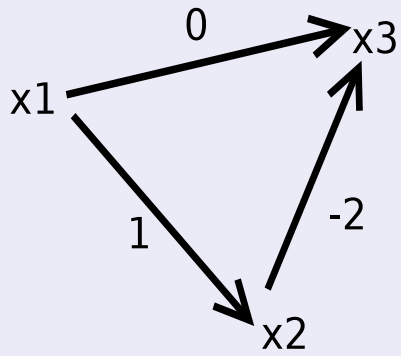
La propriété (2) est donc vérifiée après chaque nouvel ajout dans C.

En fin de traitement, tous les sommets, sauf un, sont dans C. D'après la propriété (1), D mémorise pour tous les ommets de C le cout des chemins optimaux. Pour x_n, le seul sommet hors de C, on sait que D[x_n] a pour valeur le cout du plus court chemin spécial de 1 à x_n, or, le chemin optimal de 1 à x_n est nécessairement un chemin spécial, donc D[x_n] est bien égal à son cout.

CQFD

Remarque

Les conditions d'applciation de cet algorithme imposent une valuation des arcs positive.
En effet, ceci est nécessaire, et d'ailleurs la preuve y fait référence.
L'exemple ci-dessous précise les raisons de cette condition d'application :



Dans ce cas de figure, c'est x_3 qui sera choisi plutôt que x_2, or :

$$\text{coût}([x_1,x_3])=0 \geq \text{coût}([x_1,x_2,x_3])=-1$$

Complexité de l'algorithme

- Représentation matricielle
  1. ensemble avec représentation tabulaire : $\theta(1)$ pour l'insertion mais $\theta(n)$ pour l'initialisation
  2. $\theta(n)$
  3. n-2 passages exactement
  4. si S-C est explicitement représenté, $\theta(|S\text{-}C|)$
  5. temps constant, soit $\theta(1)$
  6. si S-C est explicitement représenté, $\theta(|S\text{-}C|)$
  7. temps constant, soit $\theta(1)$

  Globalement, avec (4) et (6) cumulés avec (3), on a :

  $(\sum_{2 \leq i \leq n-1} i)$ examens d'éléments de S-C $= \frac{n(n-1)}{2} - 1 = \frac{n^2-n-2}{2}$

  Le cout globale est $\frac{n^2-n-2}{2} + n + n$ soit $\theta(n^2)$

- Représentation par listes d'adjacence

  Si on reprend l'analyse précédente, seule l'étape (6) peut être améliorée par le changement de représentation, puisqu'il suffit de traiter les sommets y succeseurs du sommet courant x, d'où un coût cumulé de $\theta(n+m)$ pour (3) et (6). Le problème ici est que rien ne change pour le coût cumulé de (3) et (4), puisque l'on a toujours $\theta(n^2)$. Il faut donc optimiser l'étape (4), ceci en changeant l'implémentation. C'est possible en utilisant la structure de tas ordonnés (vu en cours), mémorisant les sommets ordonnés par la valeur D[ ]. Incidence sur l'algorithme :

- Représentation par listes d'adjacence (suite)
  - L'étape (4) est réalisable en temps constant pour le choix de x, et log(n) pour la mise à jour de la structure, soit globalement, un coût cumulé de n.log(n) pour (4) et (3).
  - L'étape (7) est réalisable en log(n), car il faut éventuellement mettre à jour la structure, soit globalement, un coût cumulé de m.log(n) pour (3), (6) et (7).

    Finalement, on obtient $\theta((n+m).\log(n))$

    Conclusion : discussion sur :

    $$\theta(n^2 \text{ VS } \theta((n+m).\log(n))$$
  - Si le graphe est très dense, ie, si $m=n^2$, ou plus formellement $m \in \theta(n^2)$, on préfèrera bien sûr la représentation matricielle puisque l'on a $\theta(n^2)$ en comparaison de $\theta(n^2.\log(n))$.
  - Si le graphe est peu dense, ie si $m << n^2$, il convient précisement de comparer les valeurs de n^2 et de $(n+m).\log(n)$. Par exemple, si $m=n$, on a alors $(n+m).\log(n) = n.\log(n)$, il est clair que la seconde approche est meilleure.
  - Si $n^2 = (n+m).\log(n)$, il faut tenir compte des "constantes cachées" de l'implémentation, c'est à dire raisonner plus en termes d'efficacités comparées des deux approches pour choisir.

Calcul effectif des plus courts chemins

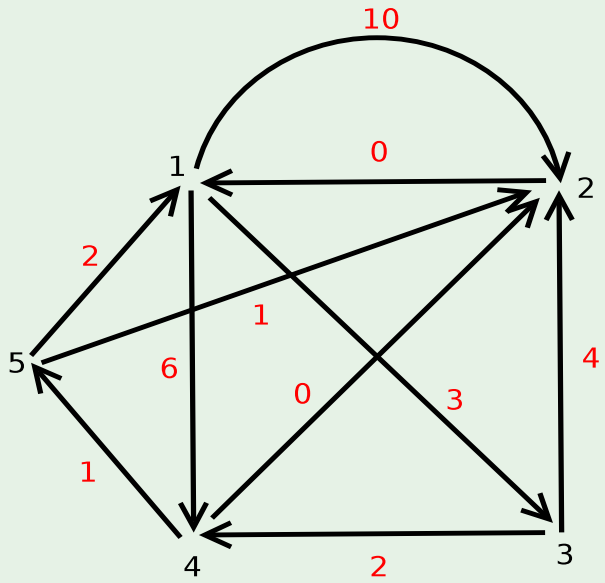
Il suffit d'ajouter au précédent algorithme, une structure de type tableau et modifier l'étape (7)

```
P : tableau [sommet] de sommet
```

Cette structure est initialisée à 1 pour tous les sommets du graphe. Le sens attribué à P est le suivant :
L'idée est que, disposant de y, on connaît son prédécesseur P[y], dont le prédécesseur est P[P[y]], et ainsi de suite... La modification dans le code ne modifie pas la complexité, elle est relative à l'étape notée (7) qui est remplacée par :

```
si D[y] > D[x]+v(x,y) alors
    D[y] := D[x]+v(x,y)
    P[y] := x
finsi
```

Exemple



Appliquons l'algorithme de Dijkstra (complet)

- D=[0,10,$\underline{\underline{3}}$,6,$+\infty$] et P=[1,1,1,1,1]
  => choix de 3 d'où C={1,3}
  et donc D=[0,3+4,3,3+2,∞] et P=[1,3,1,3,1]
- D=[0,7,3,5,$+\infty$] et P=[1,3,1,3,1]
  => choix de 4 d'où C={1,3,4}
  et donc D=[0,5+0,3,5,5+1] et P=[1,4,1,3,4]
- D=[0,5,3,5,6] et P=[1,4,1,3,4]
  =>choix de 2 d'où C={1,3,4,2}
  et donc D=[0,5,3,5,6] et P=[1,4,1,3,4]

Les chemins optimaux correspondants sont :

$$
\begin{aligned}
[1,...2] &= [1,...P[2],2] = [1,...4] + (4,2) \\
&= [1,...P[4],4] + (4,2) = [1,...3] + (3,4) + (4,2) \\
&= [1,...P[3],3] + (3,4) + (4,2) = [1,...1] + (1,3) + (3,4) + (4,2) \\
&= (1,3) + (3,4) + (4,2)
\end{aligned}
$$

$$
[1,...3] = [1,...P[3],3] = [1,...1] + (1,3) = (1,3)
$$

$$
\begin{aligned}
[1,...4] &= [1,...P[4],4] = [1,...3] + (3,4) \\
&= [1,...P[3],3] + (3,4) = [1,...1] + (1,3) + (3,4) \\
&= (1,3) + (3,4)
\end{aligned}
$$

$$
\begin{aligned}
[1,...5] &= [1,...P[5],5] = [1,...4] + (4,5) \\
&= [1,...P[4],4] + (4,5) = [1,...3] + (3,4) + (4,5) \\
&= [1,...P[3],3] + (3,4) + (4,2) = [1,...1] + (1,3) + (3,4) + (4,5) \\
&= (1,3) + (3,4) + (4,5)
\end{aligned}
$$

Preuve :

Pour prouver la validité de l'algorithme de recherche des plus courts chemins, on ajoute deux assertions supplémentaires :

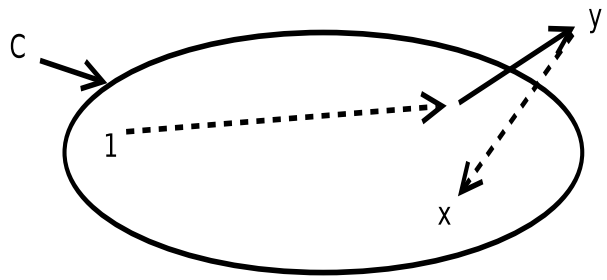
3. si x $\in$ C, P[x] est le prédécesseur de x dans le plus court chemin menant de 1 à x
4. si x $\notin$ C, P[x] est le prédécesseur de x dans le plus court chemin spécial menant de 1 à x

Quand C=1, ces propriétés sont trivialement vérifiées :
(3) puisque le seul sommet dans C est le sommet 1
(4) puisque tous les chemins spéciaux optimaux sont réduits aux arcs de la forme (1,y).
Etudions le cas général, ie, quand on ajoute un sommet x à C. On reprend le même schéma de preuve que pour vérifier la véracité des assertions (1) et (2).

propriété (3) :

Cette propriété est trivialement vérifiée pour les sommets de C avant ajout de x car la valeur de P n'est pas modifiée pour ces sommets.
Considérons x ; on doit vérifier que P[x] est le prédécesseur de x sur un chemin optimal. Par hypothèses, on sait que P[x] est le prédécesseur de x sur un plus court chemin spécial de 1 à x. Il suffit de montrer que le plus court chemin de 1 à x est dans C (c'est un chemin spécial). Ceci revient à montrer que le plus court chemin de 1 à x ne passe par aucun sommet situé hors de C ; on le montre par l'absurde.

Soit y, le premier sommet hors de C par lequel passe le plus court chemin de 1 à x ;



Il est clair que [1,y] est un chemin spécial ; par ailleurs, c'est un plus court chemin spécial car le chemin [1,...y,...x] considéré est un plus court chemin de 1 à x. On a donc :

$$\text{coût}([1,...y,...x]) \geq \text{coût}([1,y])$$

car par hypothèse pour l'algorithme de Dijkstra, $\forall i, j$, v(i,j) ≥ 0, et d'après l'hypothèse (2) vraie en entrée, donc pour y :

$$\text{coût}([1,y]) \geq D[y]$$

On peut de plus affirmer l'inégalité D[y] $\geq$ D[x] puisque c'est x qui a été choisi dans S-C, pour minimiser D. On arrive finalement à :

$$\text{coût}([1,...y,...x]) \geq D[x]$$

Donc le chemin interne à C est un chemin optimal, puisque de coût inférieur à celui de tout chemin optimal.

propriété (4) :

On a trois possibilités concernant la forme du plus court chemin spécial de 1 à y :

- [1,y] reste inchangé, ce chemin spécial ne passe pas par x. L'assertion est donc toujours vérifiée puisque P[y] n'est mis à jour que si on a D[y] $>$ D[x]+v(x,y), ce qui ne peut se présenter que si le plus court chemin spécial passe par x.
- [1,y] passe maintenant par x, le chemin est de la forme [1,...x,y], i.e x est l'avant dernier sommet du chemin où seul y n'est pas dans C. Ce cas est traité par l'algorithme lors de l'exécution de l'instruction "si D[y] $>$ D[x]+v(x,y) etc..." ; En effet, P[y] est mis à jour si nécessaire.

- [1,y] passe maintenant par x, le chemin est de la forme [1,...,x,z,y] où z≠x et seul y n'est pas dans C. Ce cas peut etre éliminé puisque z a déjà été inséré dans C avant que x ne le soit, et donc que D[z] a pour valeur la distance de 1 à z, et que nécéssairement, le cout du chemin [1,....x,...z,y] ne peut etre inférieur strictement à D[z]+v(z,y), qui a déjà été considéré lors de l'insertion de z dans l'ensemble C.

En fin de traitement, tous les sommets, sauf un, sont dans C. Donc, P mémorise pour tous les sommets x de C, leur prédécesseur sur les plus courts chemins et pour x_n, le seul sommet hors de C, on sait que P[x_n] a pour valeur le prédécesseur de x sur un plus court chemin spécial de 1 à x_n. Or le chemin optimal de 1 à x_n est nécessairement un chemin spécial, donc D[x_n] est bien égal à son cout.

CQFD

Problème du plus court chemin

Cas des valuations quelconques et graphes sans circuits : algorithme de Bellman

Cet algorithme repose sur l'hypothèse suivante : la valuation de chaque arc est quelconque, mais le graphe est sans circuit. Le problème traité est le problème 2 : on recherche les chemins optimaux d'un sommet x_1 vers tous les autres sommets du graphe ; puisque le graphe est sans circuit, on supposera de plus que le sommet point de départ est racine du graphe (unique source).

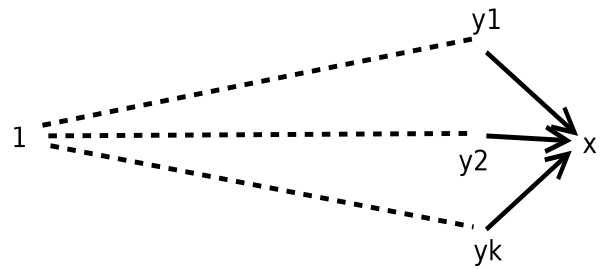
Principe de l'algorithme :

Le principe est le suivant ; Un plus court chemin de 1 à x doit passer par l'un des prédécesseurs y de x, celui pour lequel on a :

$$\text{distance}(1,y) + v(y,x) = \text{distance}(1,x)$$

Comme pour l'algorithme de Dijkstra, on utilisera donc un tableau D qui aura pour objet de mémoriser distance(1,x) pendant la recherche, i.e D[x]=distance(1,x). Un tableau P sera également utilisé pour mémoriser les chemins : P[x]=y si y est le prédécesseur de x sur un plus court chemin menant de 1 à x.

Le principe repose sur une approche du même type que celle utilisée pour le calcul d'une S-séquence pour ce qui concerne l'exploration du graphe : si les prédécesseurs y_1, y_2,... y_k de x ont déjà été traités quand on considère x :



On connait les coûts pour les plus courts chemins [1,y_i], le graphe étant sans circuit, on va pouvoir déterminer le plus court chemin [1,x], puisqu'il s'agit du chemin passant par le y_i qui minimise :

$$\text{coût}([1,y_i])+v(y_i,x)$$

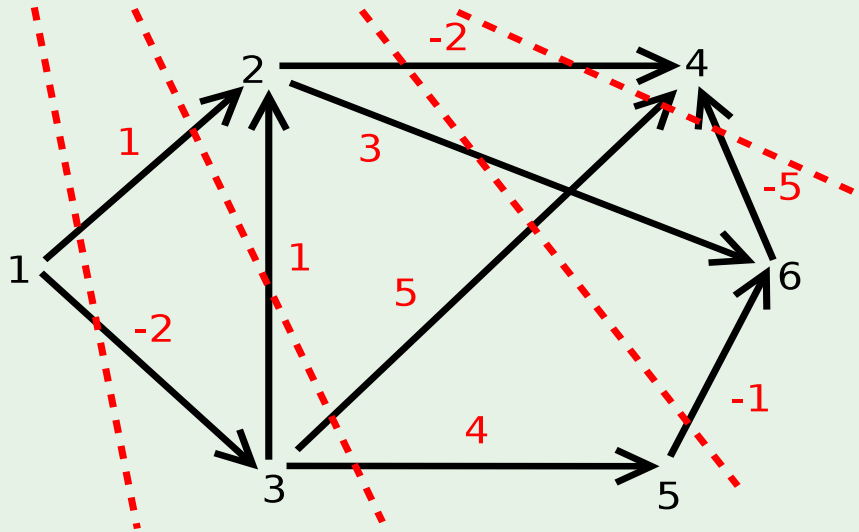
Après avoir déterminé le "bon" y_i, il suffira d'affecter la valeur de la somme $D[1,y_i]+v(y_i,x)$ à D[x], puis y_i à P[x].

L'algorithme peut alors s'écrire comme suit :

algorithme

```
var x,y:sommet;
    D:tableau[sommet] de r;
    P:tableau[sommet] de sommet;
    M:ens de sommet;

debut
   1) M := S-{1}; D[1] := 0; P[1] := 1;
   2) tantque M <> ensvide faire
   3)    choix de x dans M / tous les predecesseurs de x sont
         dans S-M;
   4)    y := <sommet predecesseur de x qui minimise
         D[y]+v(y,x)>;
   5)    M := M-{x};
   6)    D[x] := D[y]+v(y,x);
   7)    P[x] := y;
      fintq
fin
```

Exemple



D=[0, ?, ?, ?, ?, ?] et P=[1, ?, ?, ?, ?, ?]

choix de 3 D=[0, ?,-2, ?, ?, ?] et P=[1, ?,1, ?, ?, ?]

choix de 2(ou 5) D=[0,-1,-2, ?, ?, ?] et P=[1,3,1, ?, ?, ?]

choix de 5 D=[0,-1,-2, ?,2, ?] et P=[1,3,1, ?,3, ?]

choix de 6 D=[0,-1,-2, ?,2,1] et P=[1,3,1, ?,3,5]

choix de 4 D=[0,-1,-2,-4,2,1] et P=[1,3,1,6,3,5]