2020-2021, semestre automne
L3, Licence Sciences et Technologies

LIFAP6: Algorithmique, Programmation et Complexité

Chaine Raphaëlle (responsable semestre automne)
E-mail : raphaelle.chaine@liris.cnrs.fr
http://liris.cnrs.fr/membres?idn=rchaine

1

1

Graphe

- Idée générale
  - Notion plus générale que la notion d'arbre
  - Arbre = cas particulier d'un graphe
- Graphe
  - Modélisation de relations binaires entre des éléments

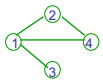
Remarque : En théorie des graphes un arbre n'est pas forcément orienté!

2

2

Définitions

- Un **graphe non orienté** est un couple <S, A> où
  - S est un ensemble fini de **sommets**
  - A est un ensemble fini de paires de sommets, appelées **arêtes**



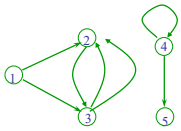
$S = \{ 1, 2, 3, 4 \}$
$A = \{ \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\} \}$

3

3

Définitions

- Un **graphe orienté** est un couple <S, A> où
  - S est un ensemble fini de **nœuds**
  - A un ensemble fini de paires ordonnées de nœuds, appelées **arcs**



$S = \{ 1, 2, 3, 4, 5 \}$
$A = \{ (1, 2), (1, 3), (2, 3), (3, 2), (3, 2), (4, 4), (4, 5) \}$

4

4

Définitions

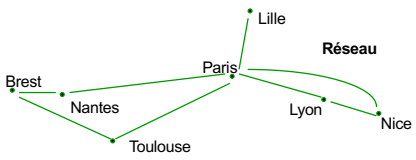
- Il arrive qu'une information de coût soit associée aux arcs (ou arêtes), voire aux nœuds (resp. sommets)
- Un **graphe valué** est un triplet <S, A, C> où
  - S est un ensemble fini de nœuds (ou sommets),
  - A un ensemble fini d'arcs (ou d'arêtes)
  - C une fonction de A dans R appelée **fonction de coût**

5

5

Exemples

- GPS : Localités reliées par des voies de communication (routes, voies ferrées, lignes aériennes, …), la fonction de coût pouvant correspondre à une distance, ou un temps de parcours, le prix du trajet...
  → Recherche de plus courts chemins



6

6

Exemples

- Représentation de localités reliées par des canalisations (eau, gaz, électricité) caractérisées par leur débit et leur capacité. Certains nœuds pouvant correspondre à des stations de distribution ou de pompage
  → Problème de flux maximal

Pipelines

7

7

Exemples

- Représentation
  - des configurations possibles d'un jeu tactique par des nœuds,
  - des coups légaux par des arcs permettant de passer d'une configuration à une autre

→Recherche position gagnante, séquences de coups forcés, etc.

8

8

Algorithmes

- **Explorations**
  - Parcours en profondeur, en largeur
  - Tri topologique
  - Composantes fortement connexes, ...
- **Recherche de chemins**
  - Clôture transitive
  - Chemin de coût minimal
  - Circuits Eulériens (passer 1 et 1 seule fois sur chaque arête) et Hamiltoniens (passer 1 et 1 seule fois sur chaque sommet), ...
- **Arbres couvrants**
  - Algorithmes de Kruskal et Prim

9

9

Algorithmes

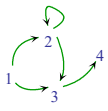
- **Réseaux de transport**
  - Flot maximal
- **Classification**
  - Coupures de graphes
- **Divers**
  - Coloration d'un graphe
  - Test de planéarité, ...

10

10

Terminologie

- Etant donné un arc (x, y)
  - x est l'**extrémité initiale** de l'arc,
  - y l'**extrémité terminale**
- On dit que
  - x et y sont **adjacents**
  - y est un **successeur** de x
  - x est un **prédécesseur** de y
- Les arcs qui partent d'un nœud lui sont **incidents extérieurement**, ceux qui y arrivent lui sont **incidents intérieurement**

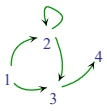


11

11

Terminologie

- Dans un graphe orienté (resp. non orienté) G, on appelle **degré** d'un nœud (resp. sommet) x et on note d°(x) le nombre d'arcs (resp. d'arêtes) dont x est une extrémité
- **Demi-degré extérieur** d'un nœud : nombre d'arcs incidents extérieurement
- **Demi-degré intérieur** d'un nœud : nombre d'arcs incidents intérieurement



12

12

Terminologie

- Dans un graphe orienté (resp. non orienté) G, on appelle **chemin** (resp. **chaîne**) de longueur l une suite de l+1 nœuds (resp. sommets) (s_0, …, s_l) tels que

$$\forall i, 0 \le i \le l-1, s_i \rightarrow s_{i+1} \text{ est un arc (resp. arête) de G}$$

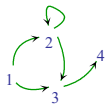
- Un chemin (resp. une chaîne) est dit **élémentaire** s'il ne contient pas plusieurs fois le même nœud (resp. sommet)
- Un **circuit** (resp. un **cycle**) est un chemin (resp. une chaîne) tel (resp. telle) que les 2 sommets aux extrémités coïncident

13

13

Terminologie

- Un graphe orienté est dit **fortement connexe** si pour toute paire de noeuds distincts (u,v) il existe un chemin de u vers v et un chemin de v vers u.
- Un graphe non orienté est dit **connexe** si pour toute paire de sommets distincts (u,v) il existe une chaîne entre u et v



14

14

Terminologie

- Un **arbre** est un graphe non orienté, connexe et sans cycle
- Etant donné un graphe orienté G = <S,A>, on appelle **racine** de G un nœud r de S tel que tout autre nœud puisse être atteint par un chemin d'origine r
- On appelle **arborescence** un graphe orienté admettant une racine et tel que le graphe non orienté associé soit un arbre

15

15

Type Abstrait Graphe

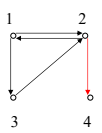
- Prévoir des procédures
  - d'initialisation, de testament, (d'affectation)
  - d'ajout / de retrait d'un nœud ou d'un arc
- Prévoir des fonctions
  - testant l'existence d'un nœud ou d'un arc
- Prévoir des routines
  - Permettant d'itérer
    - sur l'ensemble des arcs issus d'un nœud (par exemple une fonction ième arc)
    - Sur l'ensemble des successeurs d'un nœud (par exemple une fonction ième successeur)
  - De connaître l'origine (resp. l'extrémité) d'un arc

16

16

Représentations possibles

- Matrices d'adjacence
  - Chaque sommet est représenté par un indice dans 1..n
  - Ensemble des arcs représenté par un tableau de booléens de dimension n*n, où n est le nombre de nœuds



	1	2	3	4
1	F	V	V	F
2	V	F	F	V
3	F	V	F	F
4	F	F	F	F

17

17

  - Si le graphe est valué on remplace les booléen par le coût associé à l'arc (en réservant une valeur spécifique pour signifier l'absence d'arc)
  - Types
    - Nœud = 1..n
    - Graphe = tableau [Nœud, Noeud] de valeurs
  - Utilisable si pas plus d'un arc entre 2 nœuds

18

18

– Quelle propriété est vérifiée par la matrice d'adjacence d'un graphe non orienté?
– Quel est la complexité du parcours des successeurs d'un nœud?

19

19

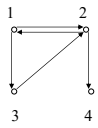
- Matrice d'adjacence
  - Représentation commode pour tester l'existence d'un arc, pour ajouter ou supprimer un arc (en $\Theta(1)$)
  - Parcours de tous les successeurs ou de tous les prédécesseurs d'un sommet en $\Theta(n)$
- Une consultation de l'ensemble des arcs du graphe requiert un temps en $\Theta(n^2)$ et cette représentation exige un espace mémoire de $\Theta(n^2)$

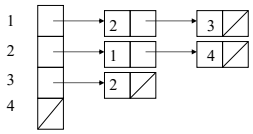
20

20

Représentations possibles

- Listes d'adjacence
  - Etant donnée une représentation des nœuds, on associe à chaque nœud la **liste de ses successeurs** rangés dans un certain ordre
  - Cette représentation utilise un espace mémoire en $\Theta(n+p)$ pour un graphe avec n nœuds et p arcs





21

21

- Parcours des successeurs d'un nœud n_i en $\Theta(deg^+(n_i))$
- Mais parcours des prédécesseurs d'un nœud n_i en $\Theta(p)$

22

22

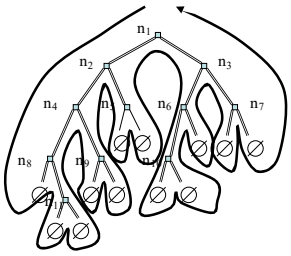
Parcours de graphe

- But : parcourir l'ensemble des nœuds du graphe, pour effectuer une certaine action en chacun des nœuds (ex. affichage d'information)
- Exploration d'un graphe plus compliquée que celle d'un arbre, mais elle s'en inspire!
- Il existe 2 grandes stratégies de parcours :
  - le parcours en profondeur (*depth-first search*)
  - le parcours en largeur (*breadth-first search*)

23

23

Rappel : Parcours d'un arbre (binaire)



Parcours en profondeur « à main gauche »
Se généralise à un arbre n-aire :
on rencontre les nœuds n+1 fois

24

24

Parcours de graphe

- Lors du parcours, on matérialisera l'état d'un nœud par une couleur
  - Blanc : nœud non découvert
  - Gris : nœud découvert
  - Noir : nœud dont tous les successeurs ont été parcourus

25

25

Parcours en profondeur

- On considère un graphe orienté dont tous les nœuds sont initialement non découverts (blanc)
- Le parcours en profondeur consiste à
  - choisir un nœud de départ s et le marquer comme découvert (gris)
  - suivre un chemin issu de s **aussi loin que possible** en marquant les nœuds en gris au fur et à mesure qu'on les découvre
  - en fin de chemin (marquage en noir), **revenir au dernier choix fait** et prendre une autre direction.

26

26

Parcours en profondeur

- Utilisation de propriétés associées aux nœuds
  - Couleur (père, ordre de découverte, fin de découverte …)
- Pour y stocker des informations sur l'exploration
- Certaines de ces informations ne sont pas indispensables à l'exploration et dépendront des besoins de l'application
- Seule l'info de couleur est indispensable
- Possibilité de stocker ces infos de manière non invasive
  - dans des Tableaux ou des Tables

27

27

Parcours en profondeur (DFS)

```
procédure DFS(donnée-résultat G : graphe)
variable
   u : sommet
début
   pour tout noeud u de G faire
      colorie(u,blanc); set_père(u, nil)
   finpour
   temps ← 0

   pour tout noeud u de G faire
      si couleur(u) = blanc alors
         set_tps_découverte(u, temps);  temps ++
         colorie(u,gris)
         DFS_visite(G,u)
      finsi
   finpour
fin
```

Procédure interne récursive

28

28

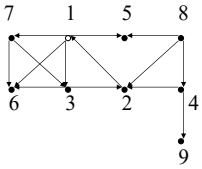
Parcours en profondeur (DFS)

```
Procédure DFS_visite(donnée-résultat G : graphe, u : sommet)
variable
   v : Nœud
début
   pour tout  nœud v successeur de u faire
      si couleur(v) = blanc alors
         set_tps_découverte(v, temps);  temps ++
         colorie(v,gris)
         set_père(v,u)
         DFS_visite(G, v)
      finsi
   finpour
   colorie(u,noir) (facultatif, 2 couleurs suffisent)
   set_tps_fin(u,temps); temps++
fin
```

29

29

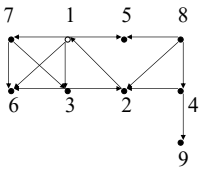
- Effectuer le parcours en profondeur sur le graphe suivant



30

30

- On obtient l'ordre de parcours suivant des sommets en partant de 1 : 1, 3, 2, 6, 5, 7, 4, 9, 8



31

31

Parcours en profondeur (DFS)

- *Complexité en temps* : O(N+M) où N est le nombre de sommets et M le nombre d'arcs
  - en effet, un sommet n'est empilé qu'une seule fois (passage de blanc à gris) et chaque arc n'est traité qu'une seule fois (à partir de son extrémité initiale)
- *Complexité en espace* : O(M)
  - un nœud est empilé au plus 1 fois,
  - la pile d'appel a une profondeur max O(M) (longueur max d'un chemin entre deux nœuds)

32

32

Parcours en largeur (BFS)

- Pour un sommet de départ s on commence par visiter tous les successeurs de s **avant** de visiter les autres descendants de s
- Le parcours en largeur consiste à visiter d'abord
  - tous les sommets à distance 1 de s,
  - puis ceux à distance 2 qui n'ont pas été visités,
  - et ainsi de suite …
- Le parcours en largeur permet donc de résoudre les problèmes de plus court chemin dans un graphe non valué

33

33

Parcours en largeur (BFS)

- Pour programmer l'algorithme, on utilise une structure de **file**:
  - lorsque à partir de s, on s'apprête à visiter ses successeurs non marqués, il est nécessaire de les ranger successivement dans une file
  - la recherche repartira ainsi de chacun des successeurs de s, à partir du premier.

34

34

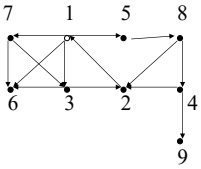
Parcours en largeur (BFS)

```
procédure BFS(donnée-résultat G: graphe, s : noeud)
variables
    u,v : nœud ; F : file
début
  pour tout noeud u ≠ s faire
    colorie(u,blanc); set_père(u, nil); set_dist(u,∞)
  finpour
  colorie(s,gris); set_père(s,nil); set_dist(s,0)
  initialise_file_vide(F); enfile(F,s)
  tant que non est-vide(F) faire
    u ← tête(F);
    pour tout noeud v successeur de u faire
      si couleur(v)=blanc alors
        colorie(v,gris); set_père(v,u); set_dist(v, u.dist +1)
        enfile(F,v)
      finsi
    finpour
    défile(F); colorie(u,noir) (facultatif : 2 couleurs suffisent))
  fintantque
fin
```

35

35

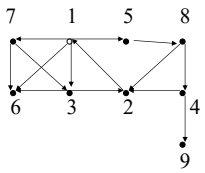
- Effectuer le parcours en largeur sur le graphe suivant



36

36

- Les sommets sont visités dans l'ordre 1, 3, 5, 6, 7, 2, 8, 4, 9



37

37

- *Complexité en temps* : O(N+M)
  où N est le nombre de nœuds et M le nombre d'arcs
  - en effet, un sommet n'est mis dans la file qu'une seule fois (passage de blanc à gris) et les arêtes sont toutes parcourues 1 fois (découverte des voisins)
- *Complexité en espace* : O(N)
  la file a une longueur au plus en O(N) (si s est connecté à tous les autres sommets)

38

38

- En fait, parcours en largeur et en profondeur s'inscrivent dans une même stratégie générale d'exploration des nœuds du graphe
- Diffèrent suivant que les successeurs d'un nœud seront rangés dans une pile ou une file

39

39

```
procédure ExplorerGraphe (donnée-résultat G: Graphe, x : noeud)
variable
  E : Salle d'attente de Noeud
début
  pour tout nœud n de G faire
    colorie(n,blanc) …
  finpour
  initialiser(E); colorie(x,gris); ajouter(E,x)
  répéter
    y← sommet(E); retirer_sommet(E)
    pour tout nœud z successeur de y faire
      si couleur(z) = blanc alors
        colorie(z, gris)
        ajouter(z,E)
      finsi
    finpour
    colorie(y, noir) (facultatif))
  jusque estvide(E)
fin
```

Version différente

Si E correspond à une Pile : parcours en profondeur (légèrement différent de celui de la version récursive)
Si E correspond à une File : parcours en largeur

40

40

```
procédure ExplorerGraphe (donnée-résultat G: Graphe, x : noeud)
variable
  E : Salle d'attente de Noeud
début
  pour tout nœud n de G faire
    colorie(n,blanc) …
  finpour
  initialiser(E); ajouter(E,x)
  répéter
    y← sommet(E); retirer_sommet(E),
    si couleur(y)=blanc alors
      colorie(y,gris);
      pour tout nœud z successeur de y faire
        si couleur(z) = blanc alors
          ajouter(z,E)
        finsi
      finpour
      colorie(y, noir) (facultatif))
    finsi
  jusque estvide(E)
fin
```

Version identique

Si E correspond à une Pile : parcours en profondeur (identique au parcours de la de la version récursive)
Si E correspond à une File : parcours en largeur

41

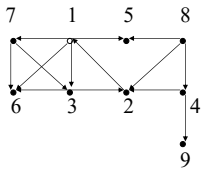
41

Tri topologique

- Problème :
  - Etant donné un graphe orienté **acyclique** modélisant une relation d'ordre partiel, trouver une relation d'ordre total entre les nœuds qui respecte l'ordre partiel
  - ie. trouver un ordre des nœud tel qu'un nœud soit toujours visité avant ses successeurs
- Utile pour les problèmes de séquencement

42

42



- Si chaque Nœud représente une tâche, la tâche 1 devra être réalisée avant les tâches 5 et 7, etc.

43

43

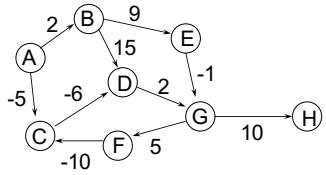
- Le parcours en profondeur permet de résoudre le problème du tri topologique
- Il suffit pour cela de classer les nœuds dans l'ordre inverse où ils sont coloriés en noir (**ordre postfixe inverse**)
- En effet un nœud est colorié en noir APRES les nœuds qu'il a permis de découvrir!

44

44

Recherche de plus court chemin

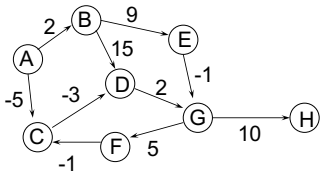
- Dans un graphe valué, il existe un plus court chemin entre 2 nœuds si il n'existe pas de circuit de valeur négative dans un chemin menant de l'un à l'autre
- Un tel circuit est dit absorbant



45

45

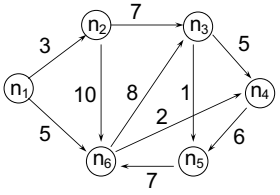
- Soit G un graphe orienté valué avec des valeurs ≥ 0
  - en fait on peut également avoir des valeurs négatives si cela ne provoque pas l'apparition de circuits absorbants



46

46

- Exemple d'un graphe orienté valué représenté par une représentation par matrice d'adjacence
  - M[i,j]=v_{ij} si les nœuds i et j sont reliés par un arc de valeur v_{ij}
  - M[i,i]=0 (entre un nœud et lui-même)
  - Distance infinie entre 2 nœuds non connectés



	n_1	n_2	n_3	n_4	n_5	n_6
n_1	0	3	∞	∞	∞	5
n_2	∞	0	7	∞	∞	10
n_3	∞	∞	0	5	1	∞
n_4	∞	∞	∞	0	6	∞
n_5	∞	∞	∞	∞	0	7
n_6	∞	∞	8	2	∞	0

- Cas d'une représentation par liste d'adjacence
  - Plus de problème de matérialisation de la distance infinie

47

47

Algorithme de Dijkstra

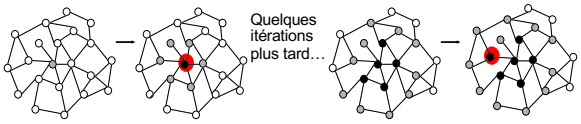
- But : Connaître les plus courts chemins entre un nœud source donné S et TOUS les nœuds du graphe accessibles depuis S
- Valable uniquement pour les graphes valués sans circuit absorbant
- Construction incrémentale et gloutonne d'un ensemble E_{noir} de nœuds accessibles depuis S
- Initialisation :
  - $E_{noir\ 0}$ vide $E_{gris\ 0}$ ={S} Ensemble des nœuds gris
- Passage à l'étape suivante en coloriant en noir un nœud gris
  - $E_{noir\ i+1}$=$E_{noir\ i}$ U {**nœud de E_{gris}** le plus proche de S en empruntant un chemin qui ne traverse que des nœuds de $E_{noir\ i}$}

48

48

Algorithme de Dijkstra

- Passage à l'étape suivante
  - $E_{noir\ i+1}=E_{noir\ i}$ U {nœud **gris** le plus proche de S en empruntant un chemin qui ne passe que par des nœuds noirs}
- Affirmation : Les sommets entrent dans E_{noir} par ordre croissant de distance à S ☺



49

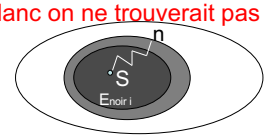
49

- Algorithme glouton :
  - à chaque étape, les choix sont faits sur la base d'une stratégie locale, qui ne sera pas remise en cause plus tard, lorsqu'on aura une meilleure vision globale
  - les algorithmes gloutons ont souvent un bon comportement asymptotique en termes de complexité mais ne sont pas toujours exacts …

50

50

- L'algorithme de Dijkstra est un algorithme glouton mais exact ☺
  - En effet, à chaque étape, **le nœud n Gris le plus proche de S « en empruntant un chemin qui ne traverse que des nœuds de $E_{noir\ i}$ »** se révèle être **le nœud de $E_{gris\ i}$ le plus proche de S.**
  - Même en passant par des sommets gris ou blanc on ne trouverait pas plus court!



51

51

- Répartition des nœuds en 3 ensembles :
  - Ensemble blanc : nœuds non atteints par un chemin
  - Ensemble gris : nœuds atteints mais à partir desquels on n'a encore mené aucune exploration et pour lesquels il existe peut-être un meilleur chemin depuis S
  - Ensemble noir : nœuds dont on a fini d'explorer le voisinage et pour lesquels on connaît un plus court chemin depuis S
- Initialisation :
  - Nœuds tous blancs, sauf le sommet S de départ en gris

52

52

- Ensemble des nœuds noirs : **$E_{noir\ i}$**
- Ensemble G des nœuds gris $E_{gris\ i}$: frontière extérieure de **$E_{noir\ i}$**
- Ensemble des nœuds blancs : nœuds non voisins d'un nœud de **$E_{noir\ i}$**

53

53

Algorithme de Dijkstra

- Mise en œuvre :
  A chaque étape, pour connaître rapidement le nœud Gris qui est le plus proche de S
  - Utilisation d'un tableau PCD (Plus Courte Distance), indicé par les numéros des sommets, et contenant 2 infos dist et pred.
    - PCD[Y].dist correspond à la plus courte distance entre S et Y si Y appartient à $E_{noir\ i}$
    - Sinon PCD[Y].dist correspond à la distance du plus court chemin entre S et Y ne traversant que des nœuds de $E_{noir\ i}$
  - Pour connaître le plus court chemin entre le sommet de départ et chacun des nœuds, on stockera également le prédécesseur, dans le chemin, de chaque nœud (dans PCD[Y].pred)

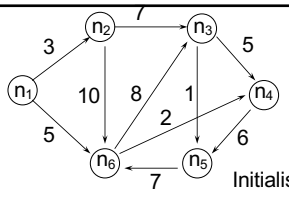
54

54

- Après chaque sélection d'un nouveau nœud Gris n pour entrer dans l'ensemble E_{noir}, on effectue un relâchement des arcs issus de n, c'est-à-dire une mise à jour des distances aux sommets directement accessibles depuis n
- Relâchement d'un arc (n,x)
  - Si cet arc permet d'améliorer la longueur du chemin menant de la source S à x :
    - Si x était blanc alors coloriage en gris
    - Mise à jour de PCD[x].dist et de PCD[x].pred

55

55



Recherche des plus courts chemins entre n1 et les autres nœuds

Initialisation :

	n1	n2	n3	n4	n5	n6
dist	0	∞	∞	∞	∞	∞
pred	0	0	0	0	0	0

Etape 1 :

Sélection de n1

	n1	n2	n3	n4	n5	n6
dist	0	3	∞	∞	∞	5
pred	0	n1	0	0	0	n1

E_{noir}={n1}

Etape 2

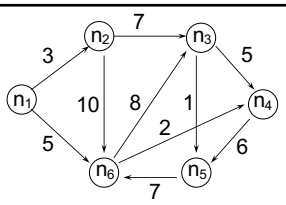
Sélection de n2

	n1	n2	n3	n4	n5	n6
dist	0	3	10	∞	∞	5
pred	0	n1	n2	0	0	n1

E_{noir}={n1,n2}
Relâchement de (n2,n3) et de (n2,n6)

56

56



Recherche des plus courts chemins entre n1 et les autres nœuds

Etape 3

Sélection de n6

	n1	n2	n3	n4	n5	n6
dist	0	3	10	7	∞	5
pred	0	n1	n2	n6	0	n1

E={n1,n2,n6}
Relâchement de (n6,n3) et de (n6,n4)

Etape 4

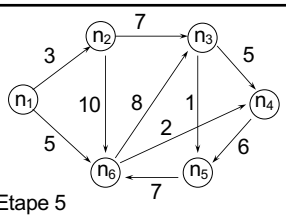
Sélection de n4

	n1	n2	n3	n4	n5	n6
dist	0	3	10	7	13	5
pred	0	n1	n2	n6	n4	n1

E={n1,n2,n6, n4}
Relâchement de (n4,n5)

57

57



Recherche des plus courts chemins entre n1 et les autres nœuds

Etape 5

Sélection de n3

	n1	n2	n3	n4	n5	n6
dist	0	3	10	7	11	5
pred	0	n1	n2	n6	n3	n1

E={n1,n2,n6, n4, n3}
Relâchement de (n3,n5)

Etape 6

Sélection de n5

	n1	n2	n3	n4	n5	n6
dist	0	3	10	7	11	5
pred	0	n1	n2	n6	n3	n1

E={n1,n2,n6, n4, n3, n5}

Plus court chemin entre n1 et n5?

58

58

Algorithme de Dijkstra

procédure Dijkstra(**données** G : Graphe, S : indice de Nœud,
résultat PCD : tableau [indice de Noeud]
de paire(distance, indice de Nœud))
variables
n_i,n_{min} : indice de Nœud
min : distance
début
//Initialisation
pour chaque nœud n_i de G **faire**
Initialisation avec couleur blanche
finpour
couleur(s)←gris
…

Le type paire (distance, indice de Nœud) est un type composé d'un champ dist et d'un champ pred

59

59

….
tant que il reste des nœuds gris **faire**
Recherche du prochain nœud gris à colorier en noir :
min←∞
pour tout nœud n_i gris **faire**
Si n_i est plus proche de S que les nœuds gris précédemment observés (avec dist(n_i) et n_i), mise à jour de min et n_{min}
finpour
pour tout arc $n_{min}n_i$ avec n_i non noir **faire**
Relâchement des arêtes issues de n_{min} (pour colorier de nouveaux sommets en gris et mettre à jour les chemins aux voisins déjà gris)
finpour
couleur(n_{min})←noir
fintantque
fin

60

60

Algorithme de Dijkstra

(dans le cas d'une représentation par matrice d'adjacence)

procédure Dijkstra(**données** G : Graphe, S : indice de Nœud,
résultat PCD : tableau [indice de Noeud]
de paire(distance, indice de Nœud))
variables
n_i,n_{min} : indice de Nœud
min : distance
début
//Initialisation
pour chaque nœud n_i de G **faire**
PCD[n_i].dist←G[S,n_i]
si G[S,n_i] = ∞ **alors** PCD[n_i].pred ← 0, couleur(n_i) ←blanc
sinon PCD[n_i].pred ← S, couleur (n_i) ←gris,
finsi
finpour
S.couleur←noir
…

Coût de l'arête entre S et n_i

Pas d'arête entre S et n_i

Le type paire (distance, Nœud) est un type composé d'un champ dist et d'un champ pred

61

61

….
tant que il reste des nœuds gris **faire**
min←∞
pour tout nœud n_i gris **faire**
si PCD[n_i].dist < min **alors**
min←PCD[n_i].dist, n_{min}←n_i
finsi
finpour
pour tout arc $n_{min}n_i$ avec n_i non noir **faire**
si n_i.couleur=blanc **alors**
n_i.couleur←gris
finsi
si PCD[n_{min}].dist + G[n_{min},n_i] < PCD[n_i].dist **alors**
PCD[n_i].dist ←PCD[n_{min}].dist + G[n_{min},n_i]
PCD[n_i].pred ← n_{min}
finsi
finpour
couleur(n_{min})←noir
fintantque
fin

Complexité améliorable en uilisant une file de priorité!

Sélection du Nœud à colorier en noir

Relâchement des arcs issus du nœud colorié en noir

62

62

Algorithme de Dijkstra

Si n est le nombre de nœuds dans le graphe et m le nombre d'arcs

- Initialisation : Parcours de n nœuds en θ(n)
- (au plus) n étapes successives
  - A chaque étape on cherche le nœud gris à colorier en noir parmi les n_g nœuds gris (n_g<=n-n_n)
    - En θ(n_g) si cet ensemble de nœuds est représenté par une liste chainée
    - En θ(n) si cet ensemble est représenté par un tableau de booléen
    - En θ(lg(n_g)) si cet ensemble est représenté par un tas binaire
  - A chaque étape, des opérations de relâchement (mise à jour de distance), suite au recrutement d'un sommet x dans E_{noir}
    - En θ(degré$^+$(x)) si représentation de G par des listes d'adjacence
    - En θ(n) si représentation de G par matrice d'adjacence

    auxquelles il faut ajouter des opérations de réajustement de tas binaire si jamais on décide d'en utiliser un!
    - En θ(lg(n_g))*(degré$^+$(x))

63

63

Algorithme de Dijkstra

- Algorithme
  - en θ(n^2) pour une représentation par matrice d'adjacence,
  - ou en θ(n(lg(n)+m+lg(n)*m)) si on représente G, E_{noir} et l'ensemble E_{gris} des nœuds gris par des structures judicieuses!

64

64

Réseau de transport

- Définition
  - Graphe orienté et valué G=(N,A) ayant un unique nœud **s** sans prédécesseur (**entrée** ou **source** du réseau) et un unique sommet **t** sans successeur (**sortie** ou **puit** du réseau)
  - Notion de capacité maximale d'un arc u de A
    - C_u : saturation, valeur maximale pour l'arc
    - Φ_u : flot réel

    On a $0 \le \Phi_u \le C_u$

65

65

- Lois de conservation aux nœuds
  - Pour tout nœud **x** différent de **s** ou **t**, le flux entrant doit correspondre au flux sortant

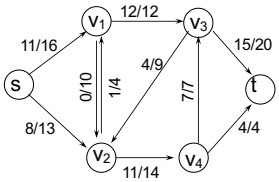
$$\sum_{y \in succ(x)} \phi_{(x,y)} = \sum_{z \in pred(x)} \phi_{(z,x)}$$

  - En ce qui concerne **s** et **t**, Φ_0 est la valeur du flot circulant entre **s** et **t** dans G

$$\phi_0 = \sum_{y \in succ(s)} \phi_{(s,y)} = \sum_{z \in pred(t)} \phi_{(z,t)}$$

66

66



- La loi de conservation aux nœuds est-elle vérifiée dans ce réseau de transport?
- Quelle est la valeur totale du flot circulant entre **s** et **t**?

67

67

Ford-Fulkerson

- L'algorithme de Ford-Fulkerson permet de connaître le flot maximal supportable par un réseau de transport
- En partant d'une valeur nulle, l'algorithme consiste à augmenter la valeur du flot de manière itérative tout en respectant la capacité de l'ensemble des arcs

68

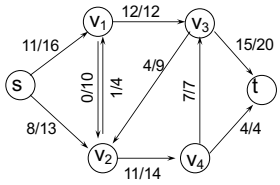
68

Ford Fulkerson

- A chaque itération :
  - On cherche un chemin non saturé de G reliant **s** à **t** le long duquel on pourrait augmenter le flot (chemin dit « améliorant »)
  - Pour cela :
    - On considère le graphe résiduel G' à partir de l'état actuel des flots dans G
    - Un chemin non saturé dans G correspond à un chemin dans G'
  - Si un tel chemin existe, on augmente le flot de la quantité correspondante
- Attention :
  - Pour augmenter la valeur du flot circulant entre **s** et **t** on pourra être amené à diminuer le flot dans certains arcs ☺

69

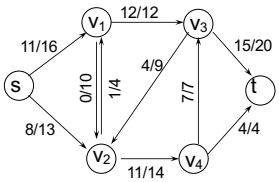
69



- Testez si le flot est optimal en recherchant un chemin améliorant (s'il existe)

70

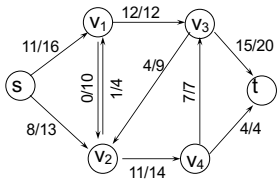
70



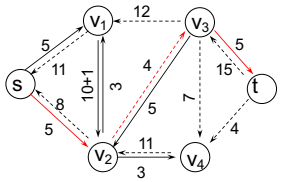
Etant donné un réseau de transport G, les arcs du graphe résiduel G' permettent d'identifier:
• les arcs de G dans lesquels il est encore possible d'augmenter le flux (arcs orientés dans le même sens dans G et G')
• les arcs de G dans lesquels il est possible de décrémenter le flux (arcs orientés en sens inverse dans G et G')

71

71



- Réseau résiduel G'



On a trouvé un chemin améliorant s, v2, v3, t.

72

72

- Recherche d'un chemin améliorant sans calculer l'ensemble du graphe résiduel
  - Initialisation :
    - au début, tous les sommets de **G** sont non marqués (blancs)
    - Les nœuds de **G** sont les nœuds de **G'**
  - On marque (en gris) l'entrée **s** du réseau de **G**
  - Tant que la sortie **t** est non marquée (blanche) et qu'il reste des sommets marqués non examinés (ie des sommets gris)
    - On examine (avant passage au noir) un sommet **x** marqué mais non examiné
      - On marque (en gris) **tous les successeurs** **y** de **x** correspondant à des arcs non saturés
        L'arc **(x,y)** est un arc de **G'** et on lui associe la valeur $C_{(x,y)} - \Phi_{(x,y)}$
      - On marque **tous les prédécesseurs** **z** de **x** correspondant à des arcs portant un flux strictement positif
        L'arc **(x,z)** est un arc de **G'** et on lui associe la valeur $\Phi_{(z,x)}$

73

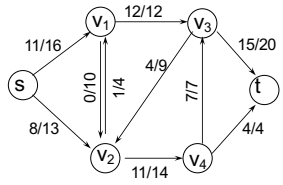
73

- Si on réussit à marquer **t**, alors on a trouvé un chemin améliorant :
  - Ce chemin améliorant, ainsi que sa capacité, apparaissent dans le graphe résiduel **G'** en cours de construction

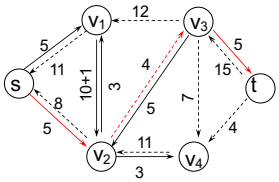
    Capacité **ε**= min(valeurs des arcs du chemin améliorant)
- Attention :
  - Certains arcs du chemin améliorant correspondent à des arcs de **G** (appelons $\mathbf{L^+}$ leur ensemble)
  - Certains arcs du chemin améliorant correspondent à des arcs de **G** pris dans le sens inverse (appelons $\mathbf{L^-}$ leur ensemble)
- Il est alors possible d'augmenter de **ε** la valeur du flot dans le réseau en augmentant de **ε** la valeur des arcs de $\mathbf{L^+}$ et en diminuant de **ε** la valeur des arcs de $\mathbf{L^-}$

74

74



- Réseau résiduel G'



On a trouvé un chemin améliorant s, v2, v3, t.
La capacité de ce chemin est 4.
Dans le graphe G, on améliore donc le flot en affectant :
- 8+4=12 à l'arc (s,v2)
- 4-4 =0 à l'arc (v3,v2)
- 15+4=19 à l'arc (v3,t)

75

75

- Méthode de Ford-Fulkerson
  - Initialiser le flot Φ de tous les arcs à 0
  - Tant qu'il existe un chemin améliorant p de capacité ε, modifier les arcs correspondant dans G
  - Retourner Φ_0

76

76