Cours: Algo. sur les Graphes

Chapitre 1: Introduction à la théorie des graphes

Réalisé par:
Dr. Sakka Rouis Taoufik
https://github.com/srtaoufik/CoursAlgoGraphe

1

Chapitre 1: Introduction à la théorie des graphes

I. Définition d'un Graphe

Un graphe est un ensemble de nœuds (sommets) reliés par des liens (arêtes).

Notation : G = (V, E) où V est l'ensemble des sommets et E l'ensemble des arêtes.

Les graphes modélisent des relations entre objets.

Ils sont utilisés en informatique, réseaux, logistique, biologie, etc.

Exemple : réseaux sociaux, cartes routières.

2

Définition : Un graphe non orienté (GNO) G est formellement défini comme suivant: G=(S,A), où:

S=$\{s_1, \dots, s_n\}$ représente un ensemble fini et non vide de sommets de G.

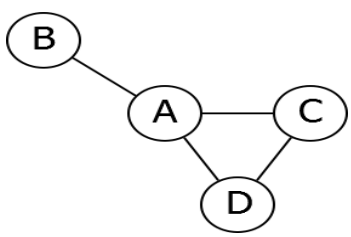
A ∈ $\mathrm{P}(\mathrm{S} \times \mathrm{S})$ est un ensemble d'arêtes de G.

Relation entre sommets est symétrique: on ne tient pas compte de la direction des arêtes.

$$\forall (s_i, s_j) \in A \Rightarrow (s_i, s_j) = (s_j, s_i)$$

Une arête (s1,s2) d'un graphe GNO est représentée comme suit:

Exemple 1: Considérons le graphe suivant d'ordre 4 (c'est le nombre de sommets).

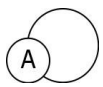


- **Sommets:** A, B, C, D
- **Arêtes:** (A,B), (A,D), (A,C), (C,D).

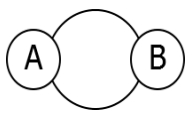
II. Types de Graphes

2.1 Graphe non orienté

Boucle: une arête reliant un sommet à lui-même.



Multi-arêtes: deux sommets peuvent être reliés par plusieurs arêtes.



- Tout graphe comportant des multi-arêtes est appelé **multigraphe**.
- Tout graphe ne comportant pas de boucles et de multi-arêtes est appelé **graphe simple**.

II. Types de Graphes

2.2 Graphe orienté

Définition : Un graphe orienté (GO) G est formellement défini comme suivant: G=(S,A), où:

- S=$\{s_1, \dots, s_n\}$ représente un ensemble fini et non vide de sommets de G.
- A ∈ $\mathrm{P}(S \times S)$ est un ensemble d'arcs de G.

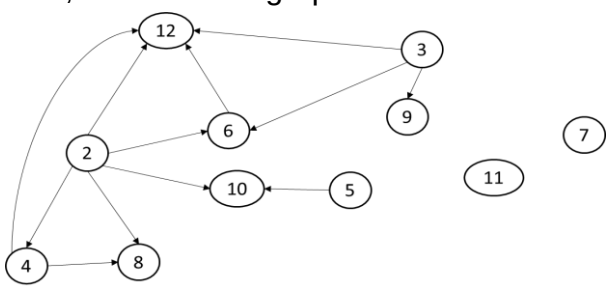
- Relation de sommets est asymétrique: on tient compte de la direction des arêtes.

$$\forall (s_i, s_j) \in A \Rightarrow (s_i, s_j) \neq (s_j, s_i)$$

- Une arête (s1,s2) est appelée **arc** et représentée comme suit:

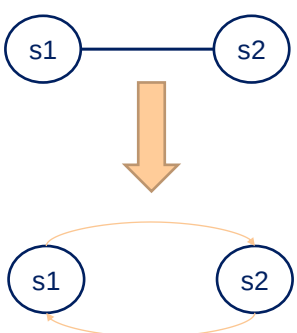
Exemple 2: le graphe de diviseurs d'entiers

- Les sommets sont les nombres {2,3,4,5,…, N}, où N est un entier naturel.
- Une arête joint p à q si l'entier p divise q.
- Pour N=12, on obtient le graphe suivant :



7

- Un graphe GNO peut être transformé en un graphe équivalent orienté en se servant de la règle suivante:



8

Définition : Un graphe pondéré (ou valué) G est formellement défini comme suivant: G=(S,A,v), où:

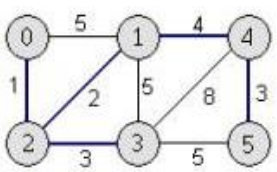
S=$\{s_1, \dots, s_n\}$ représente un ensemble fini et non vide de sommets de G.

A ∈ $\mathrm{P}(\mathrm{S} \times \mathrm{S})$ désigne un ensemble d'arcs ou arêtes de G.

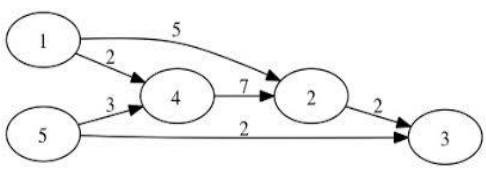
v∈ $\mathrm{A} \to \mathbb{R}$ est la fonction de pondération des arcs ou arêtes de G.

Un graphe valué est un graphe GO ou GNO ayant des arêtes (ou arcs) pondérés.

Exemple 3: un graphe non orienté valué



Exemple 4: un graphe orienté valué



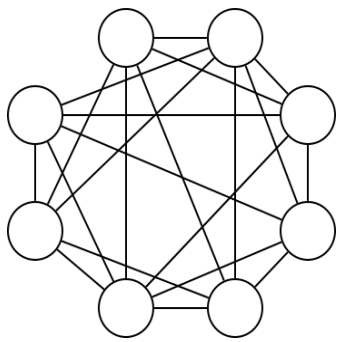
Chapitre 1: Introduction à la théorie des graphes

II. Types de Graphes

2.4 Graphe dense/clairsemé

Graphe dense G=(S,A):

$|A| \approx |S|^2$



Une grande partie des paires de sommets sont reliées par des arêtes.

11

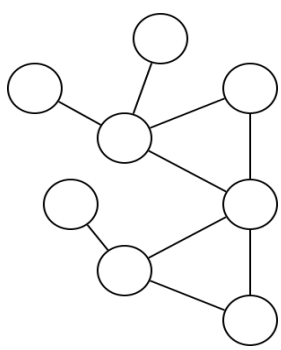
Chapitre 1: Introduction à la théorie des graphes

II. Types de Graphes

2.4 Graphe dense/clairsemé

Graphe clairsemé G=(S,A):

$|A| \approx |S|$



Chaque sommet n'a que quelques arêtes.

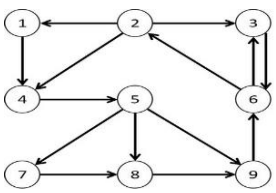
12

II. Types de Graphes

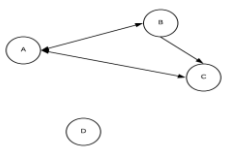
2.5 Graphe connexe

Définition : un graphe connexe est un graphe dont ses sommets sont deux à deux reliés par au moins une chaine.

- Exemple 1: un graphe connexe



- Exemple 2: un graphe non connexe



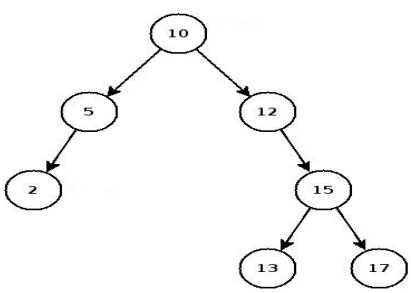
13

II. Types de Graphes

2.5 Graphe connexe

Définition: un Arbre est un graphe connexe sans cycle (ne comportant pas de cycles) et ayant un nombre d'arcs égal au nombre de sommets moins 1.

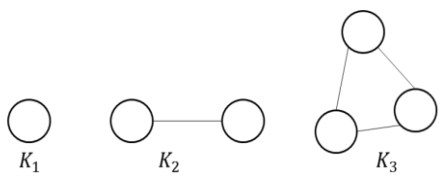
- Exemple : un arbre binaire de recherche



14

Définition: Un graphe **complet** est un graphe connexe dans lequel chaque paire de sommets est reliée par une arête.

Exemple :



Exercice:

Construire un graphe orienté dont les sommets sont les entiers compris entre 2 et 12 et dont les arcs représentent la relation « être diviseur de ».

Réponse:

Chapitre 1: Introduction à la théorie des graphes

III. Représentations des Graphes

Principalement, on distingue deux représentations physiques possibles d'un graphe:

- Représentation par matrice d'adjacence
- Représentation par liste d'adjacence

➔ La première représentation concerne les graphes n'ayant pas des arêtes multiples.

- La représentation par matrice d'adjacence est utilisable pour des graphes denses.
- La représentation par liste d'adjacence est utilisable pour des graphes non denses.

17

Chapitre 1: Introduction à la théorie des graphes

III. Représentations des Graphes

3.1 Rep. par matrice d'adjacence

- Un graphe peut être matérialisé par une matrice carré (n*n), avec n est le nombre de sommets d'un graphe.
- Un graphe non pondéré, n_{ij}=1 si et seulement si il existe un arc relient le sommet i par le sommet j, sinon n_{ij}=0.
- La complexité en espace est O (n^2).
- La traduction en C est comme suit :

```
#define nb_sommet 100
unsigned mat_adj [nb_sommets][nb_sommets];
```

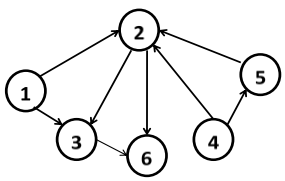
18

Chapitre 1: Introduction à la théorie des graphes

III. Représentations des Graphes

3.1 Rep. par matrice d'adjacence

Illustration:



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

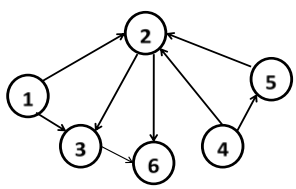
19

Chapitre 1: Introduction à la théorie des graphes

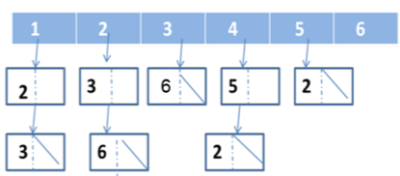
III. Représentations des Graphes

3.2 Rep. par liste d'adjacence

- **Illustration:**



Les successeurs **directes** de sommets



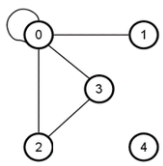
20

Chapitre 1: Introduction à la théorie des graphes

IV. Propriétés de base

4.1 Degré d'un sommet

- Dans un graphe GNO simple, le degré d'un sommet est le nombre des voisins de ce sommet.
- Un sommet avec un degré 0 est appelé isolé.
- La somme des degrés de tous les sommets d'un graphe est égale au double du nombre total d'arêtes.
- Considérons le graphe ci-dessous:



d(0)=5
d(1)=1
d(2)=2
d(3)=2
d(4)=0

- On remarque que la somme des degrés est 10 qui est le double du nombre d'arêtes.

21

Chapitre 1: Introduction à la théorie des graphes

IV. Propriétés de base

4.1 Degré d'un sommet

Dans un graphe GO, on parle de deux type de degré:

Degré intérieur (noté d^-)

Degré extérieur (noté d^+)

Définition : Soit s un sommet d'un graphe GO.

Le **degré intérieur** de s (d^-(s)) est le nombre de ses arcs entrants:

$$d^-(s) = |\{x|(x,s) \in A\}|$$

Le **degré extérieur** de s (d^+(s)) est le nombre de ses arcs sortants:

$$d^+(s) = |\{x|(s,x) \in A\}|$$

22

Chapitre 1: Introduction à la théorie des graphes

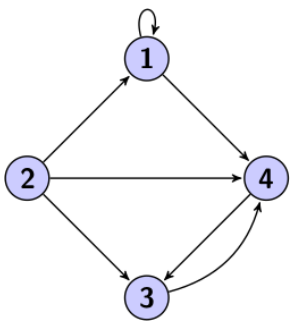
IV. Propriétés de base

4.1 Degré d'un sommet

- Le degré d'un sommet s (d(s)) est la somme de son degré intérieur et son degré extérieur.

$$d(s) = d^-(s) + d^+(s)$$

- Considérons le graphe GO suivant:



$d^-(1)=2$
$d^+(1)=2$
$d^-(2)=0$
$d^+(2)=3$
$d^-(3)=2$
$d^+(3)=1$
$d^-(4)=3$
$d^+(4)=1$

Le total des degrés est 14 qui est le double du nombre des arcs (7).

23

Chapitre 1: Introduction à la théorie des

IV. Propriétés de base

4.2 Chemin/chaine dans un graphe

Définition : Un chemin (resp. une chaine) d'un graphe GO (resp. GNO) est une séquence <$s_0, \dots, s_k$> de sommets tels que :

$$\forall i \in \{1, \dots, k\} \Rightarrow (s_{i-1}, s_i) \in A$$

Pour un chemin, si les sommets d'extrémités u et v sont identiques, on parle alors d'un **circuit**. Pour une chaine, on parle d'un **cycle**.

Le graphe (**dirigé ou non orienté)** ne comportant pas de cycles est appelé **acyclique**.

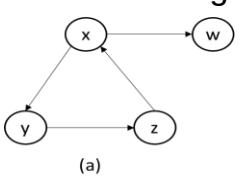
Un chemin (ou une chaine) est **élémentaire** si et seulement si ses sommets sont tous distincts.

24

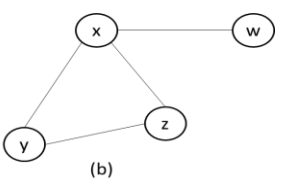
Définition :

- La longueur d'un chemin (resp. chaine) d'un graphe GO (resp. GNO) est le nombre d'arcs (resp. arêtes) parcourus.
- La distance entre deux sommets d'un graphe GO (resp. GNO) est la longueur du **plus court chemin** (resp. chaine) reliant ces deux sommets.
- On appelle diamètre (ou profondeur) d'un graphe G, la distance maximale entre deux sommets de celui-ci.

Considérons les deux graphes ci-dessous.



(a)



(b)

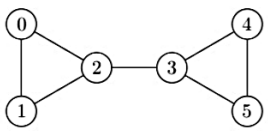
- La profondeur du graphe (a) est 3.
- La séquence « <x,y,z,x> » est un chemin pour le graphe (a) et une chaine pour celui de (b). Cette séquence forme une circuit. La longueur de cette séquence est 3.
- Le chemin « <x,y,z> » du graphe (a) est élémentaire tandis que « <x,y,z,x> » ne l'est pas.

Chapitre 1: Introduction à la théorie des graphes

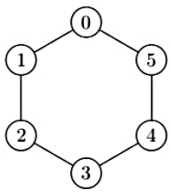
IV. Propriétés de base

4.2 Chemin/chaine dans un graphe

: Déterminez la diamètre de chacun des graphes suivants:



graphe G_1



graphe G_2

27

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.1 Graphe du flot du contrôle

- Examine la structure du programme en testant des portions de programme.
- Un graphe de flot est utilisé pour extraire les chemins d'exécution possibles.
- l'analyse du graphe permet d'identifier les données de tests en fonction d'un chemin dans le graphe.

28

V. Applications

5.1 Graphe du flot du contrôle

On dit que des données de test sensibilisent un tel chemin si le flot d'exécution avec ces données en entrée suit ce chemin.

• Critère de couverture: « tous-les-noeuds » (statement coverage)
Objectif : sensibiliser suffisamment de chemins de contrôle pour visiter tous les nœuds du graphe.

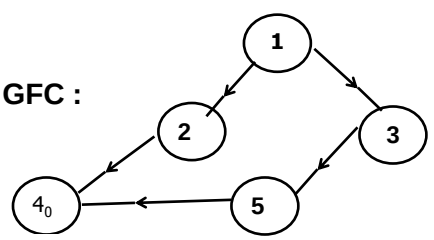
• Critère de couverture : « tous-les-arcs » (decision coverage)
Objectif : sensibiliser suffisamment de chemins de contrôle pour visiter tous les arcs du graphe.

V. Applications

5.1 Graphe du flot du contrôle

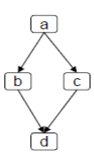
•Tout chemin qui mène d'un sommet de départ au sommet d'arrivé est une exécution possible

Règles de construction du GFC :

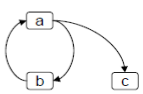


séquentielle

alternative



itérative



Choix des données de test

- Chemin à suivre?: impérativement on part du sommet de départ et on fini par le point d'arrivé.

- Comment tester? : il faut sensibiliser tout les chemins:
- Pour l'exemple précédant: (1-2-4_0 , 1-3-5-4_0)

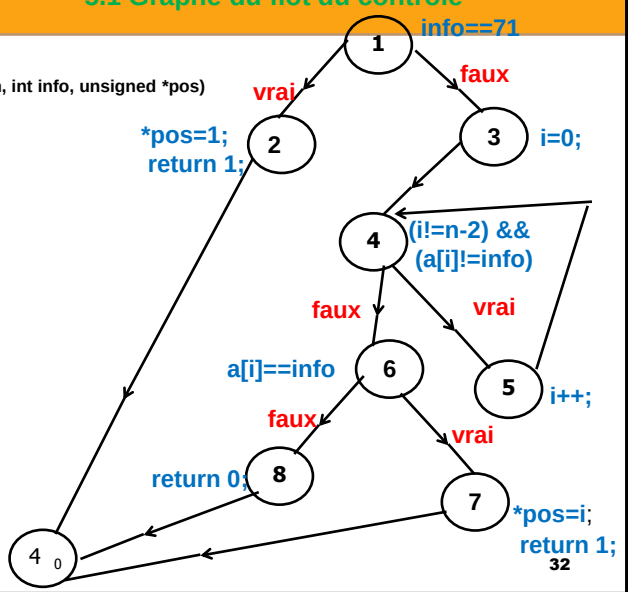
Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.1 Graphe du flot du contrôle

Exemple:

```
unsigned recherche(int a[], unsigned n, int info, unsigned *pos)
{  unsigned i;
     if (info==71)
    { *pos=1;
      return 1;
    }
    i=0;
    while((i!=n-2) && (a[i]!=info))
     i++;
     if (a[i]==info)  { *pos=i;
                        return 1;
    }
   return 0;
  }
```



Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.1 Graphe du flot du contrôle

•Une exécution possible (une DT) est un chemin partant du sommet de début (ici 1) et arrivant au sommet de destination (ici 4_0)

•La faute peut être détecter facilement en sensibilisant le chemin: 1-2-4_0

33

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.1 Graphe du flot du contrôle

Exercice 1:
1/ Réaliser le graphe de contrôle associe à la fonction suivante ;
2/ Calculer le nombre de chemins de contrôle ;

```
int pgcd (int m, int n) {
 int a = m;
 int b = n ;
 if(a <= 0 || b <= 0)
  return 1 ;
 while(a >= b) {  /*la version correcte est a!=b */
  if(a > b)
    a = a - b ;
  else
    b = b - a ;
  }
  return a ;
}
```

34

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.1 Graphe du flot du contrôle

Exercice 2

1/ Donner le **GFC (Graphe de Flot de Contrôle)** de l'implémentation suivante.

2/ Proposer des **DT (Données de Test)** permettant de sensibiliser tous les chemins d'exécution du GFC établi.

```
int recherche_dichotomique(int info, int tab[], unsigned n) {
  int g, d ; /*sous tableau de recherche*/
  int m ; /*indice de l'élément au milieu*/
  int intervalle ; /*nombre d'élément de l'ensemble de recherche*/
/*initialisation*/
  g=0 ;  d=n-1 ;
  do  {      intervalle=d-g+1;
             m=(g+d)/2 ;/*division entière*/
             if(tab[m]==info) /*issue positive*/
                                return (1) ;
     /*soit déplacer g soit déplacer d jamais déplacer les deux à la fois*/
              if(info>tab[m])
                                g=m+1;
              else
                                d=m-1;
    } while(g<=d) ;
    return 0; /*issue négative*/
}
```

35

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

- **Algorithme glouton** : construit une solution de manière incrémentale, en optimisant un critère de manière locale.

- **Diviser pour régner** : divise un problème en sous-problèmes indépendants (qui ne se chevauchent pas), résout chaque sous-problème, et combine les solutions des sous-problèmes pour former une solution du problème initial.

- **Programmation dynamique** : divise un problème en sous-problèmes qui sont non indépendants (qui se chevauchent), et cherche (et stocke) des solutions de sous-problèmes de plus en plus grands

36

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

En règle générale, la programmation dynamique permet de résoudre tous les problèmes nécessitant de maximiser ou de minimiser certaines quantités ou les problèmes de dénombrement nécessitant de compter les arrangements dans certaines conditions ou avec certains problèmes de probabilité.

Tous les problèmes de programmation dynamique satisfont la propriété de sous-problèmes qui **se chevauchent** et la plupart des problèmes dynamiques classiques satisfont également à la propriété de sous-structure optimale. Une fois que nous observons ces propriétés dans un problème donné, assurez-vous qu'il peut être résolu en utilisant la programmation dynamique.

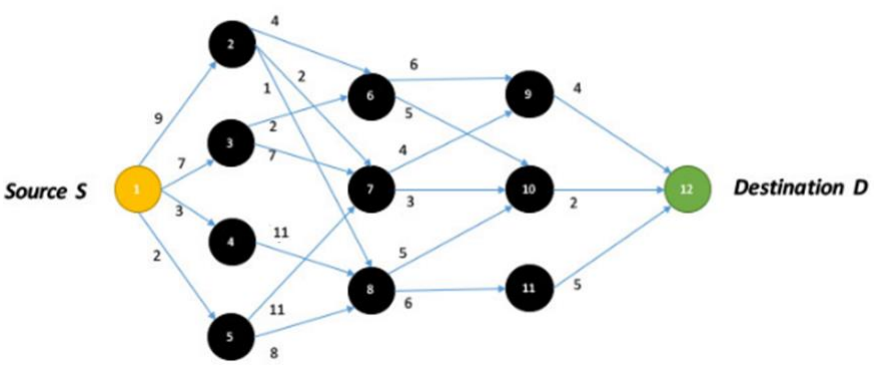
37

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Le plus cours chemin entre la source **S** et la destination **D ?**



38

Dans notre exemple, le problème consiste à calculer le plus cours chemin entre la source **S** et la destination **D**. Ce problème a les deux propriétés : Chevauchement de sous-problèmes et Sous-structure optimale donc c'est un problème de programmation dynamique.

Chevauchement de sous-problèmes : deux appels au plus court chemin à partir du noeud 9, une à partir du noeud 6 et l'autre du noeud 7

Sous-structure optimale : le nœud 7 se trouve dans le plus court chemin du nœud 1 à un nœud de destination 12,
le plus court chemin de 1 à 12 est la combinaison du plus court chemin de 1 à 7 et du plus court chemin de 7 à 12

39

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Etape 1: Identifiez les variables du problème
Dans notre exemple, les deux paramètres susceptibles de changer pour chaque sous-problème sont :

- On veux calculer le cout minimum du nœud x au nœud D

➔Cout minimum entre deux noeuds?

- On veux déterminer le prochaine nœud dans le chemin le plus court entre le nœud x et le nœud D

➔Prochaine noeud dans le chemin le plus court?

40

Etape 2: Exprimer clairement la relation de récurrence

C'est une étape important. Exprimer la relation de récurrence aussi clairement que possible renforcera la compréhension de votre problème et facilitera considérablement le reste.

Une fois que vous avez déterminé que la relation de récurrence existe et que vous avez spécifié les problèmes en terme de paramètres, cela devrait être une étape naturelle. Comment les problèmes se rapportent-ils ? En d'autres termes, supposons que vous avez calculé les sous-problèmes. Comment calculeriez-vous le problème principal ?

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Etape 2: Exprimer clairement la relation de récurrence

Voici comment nous y pensons dans notre exemple :

Pour calculer le plus court chemin entre le nœud **6** et le nœud **12 (D)** il faut récupérer le minimum du cout entre les nœuds voisins de **6** dans le chemin vers le nœud **12 (deux choix passant par le nœud 9 ou par le nœud 10)**

→ On peut représenter cette relation comme suit :

Cout(6)=Min{poids(6-9) + Cout(9), poids(6-10) + Cout(10)}

en générale :

Cout(i)=Min $_{\text{v=nœuds voisins}}$ **{ poids(i, v) + cout(v) }**.

Etape 3: Faire la tabulation (de bas en haut)

C'est l'étape la plus simple, consiste à commencer le calcule à partir du destination **(D)** et remonté dans le graph jusqu'à le nœud source **(S)**.

Etape 1 : trivial le plus court chemin entre 12 et 12 c'est zéro

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	12

Etape 2 : calculer le plus court chemin entre la destination et les noeud menant directement au nœud 12

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	4	2	5	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	nulle	nulle	nulle	12	12	12	12

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Etape 3 :

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	nulle	nulle	nulle	nulle	7	5	7	4	2	5	0
Nœud suivant	nulle	nulle	nulle	nulle	nulle	10	10	10	12	12	12	12

Cout(8)=min{ poids(8-11)+cout(11), poids(8-10)+cout(10)}
=min{6+5, 5+2}=7 (nœud suivant est 10)

Cout(7)=min{ poids(7-9)+cout(9), poids(7-10)+cout(10)}
=min{4+4, 3+2}=5 (nœud suivant est 10)

Cout(6)=min{ poids(6-9)+cout(9), poids(6-10)+cout(10)}
=min{6+4, 5+2}=7 (nœud suivant est 10)

45

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Etape 4 :

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	nulle	7	9	18	15	7	5	7	4	2	5	0
Nœud suivant	nulle	7	6	8	8	10	10	10	12	12	12	12

Cout(5)=min{ poids(5-7)+cout(7), poids(5-8)+cout(8)}
=min{11+5, 8+7}=15 (nœud suivant est 8)

Cout(4)=min{ poids(4-8)+cout(8)}
=min{11+7}=18 (nœud suivant est 8)

Cout(3)=min{ poids(3-6)+cout(6), poids(3-7)+cout(7)}
=min{2+7, 7+5}=9 (nœud suivant est 6)

Cout(2)=min{ poids(2-6)+cout(6), poids(2-7)+cout(7) , poids(2-8)+cout(8)}
=min{4+7, 2+5, 1+8}=7 (nœud suivant est 7)

46

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.2 Graphe de décision & Prog dynamique

Etape 5:

Nœud	1	2	3	4	5	6	7	8	9	10	11	12
Cout	16	7	9	18	15	7	5	7	4	2	5	0
Nœud suivant	{2,3}	7	6	8	8	10	10	10	12	12	12	12

Cout(1)=min{ poids(1-2)+cout(2), poids(1-3)+cout(3) , poids(1-4)+cout(4) , poids(1-5)+cout(5)}
=min{9+7, 7+9, 3+18, 2+15}=16(nœud suivant est 2 ou 3)

Donc le cout du plus court chemin allant du nœud **1 (S)** au noeud **12(D)** est **16** en partant soit de **2 ou de 3:**
1->2->7->10->12 ou bien 1->3->6->10->12
La complexité de cet algorithme est **$O(n^2)$** dans un graph complet ou **n** est le nombre de nœuds
Comme vous voyez au cours de résolution du problème on a 5 étapes et dans chaque étape il faut prendre une décision pour le chemin optimal à suivre d'ou le nom de **graphe en plusieurs étapes** et **c'est l'idée générale de la programmation dynamique,**
Implémentation ==> https://sourceforge.net/projects/algocompmr1/files/ **(GraphDesision.py)**

47

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.3 Graphe de décision & Algo. glouton

Idée générale:

- Un algorithme de Glouton effectue à chaque instant, le meilleur choix possible sur le moment, sans retour en arrière ni anticipation des étapes suivantes, dans l'objectif d'atteindre un résultat optimal.

Cas d'usages:

- Les algorithmes gloutons permettent de résoudre les problèmes d'optimisation
  Exemples:
  le problème du rendu de monnaie,
  le problème du sac à dos,
  Les problèmes d'organisation,
  le problème du voyageur de commerce

48

Avantages:

- Les algorithmes gloutons sont simples dans leur logique et donc faciles à implémenter
- les algorithmes gloutons sont généralement moins coûteux qu'une exploration systématique des solutions
- Ils sont capables de produire rapidement des résultats qui s'avèrent, dans de nombreux cas , suffisamment optimisées pour être acceptables

Limites:

- Dans certains cas, ils donnent la meilleure solution: on parlera d'algorithmes gloutons exacts.
- Dans d'autres, non, on parlera d'heuristiques gloutonnes.

Exemple 1: Le problème du voyageur de commerce

- L'énoncé du problème du voyageur de commerce est le suivant : étant donné n villes et les distances entre toutes les paires de villes, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque ville et revienne à la ville de départ.

- Le problème du voyageur de commerce peut être modélisé par un **graphe non orienté pondéré**. Les villes sont les sommets du graphe. Le coût d'une arête entre deux sommets est la distance entre les deux ville correspondante. Le problème est de trouver le plus court cycle hamiltonien dans le graphe G.

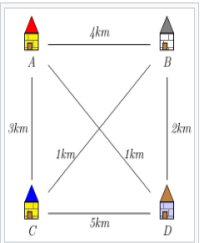
Chapitre 1: Introduction à la théorie des graphes

V. Applications

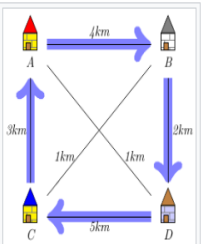
5.3 Graphe de décision & Algo. glouton

➔ Approche gloutonne: le plus proche voisin

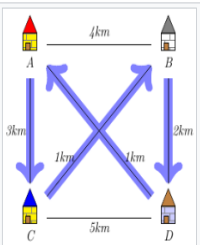
Dans la méthode du plus proche voisin, on part d'un sommet quelconque et à chacune des(n-1) itérations on relie le dernier sommet atteint au sommet le plus proche au sens coût, puis on relie finalement le dernier sommet au premier sommet choisi.



Instance du problème du voyageur de commerce.



Le chemin ABDCA de longueur 4+2+5+3 = 14 km.



Le chemin ACBDA est le chemin le plus court qui part de A, revient à A et passe par toutes les villes. Il est de longueur 3+1+2+1 = 7 km.

51

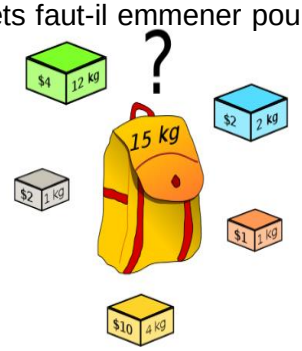
Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.3 Graphe de décision & Algo. glouton

Exemple 2: Le problème du sac à dos

Supposons que vous souhaitiez emporter des objets ayant chacun une certaine valeur mais que vous n'ayez à votre disposition qu'un sac ne pouvant supporter que 15 kg maximum. Quels objets faut-il emmener pour que la valeur du sac soit maximale ?



52

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.3 Graphe de décision & Algo. glouton

Si l'on ne veut pas passer trop de temps pour remplir son sac à dos, il va falloir trouver une autre méthode !
Méthode gloutonne: en fonction des valeurs, certains critères de choix fonctionneront mieux que d'autres ; ainsi le choix d'un ratio valeur/poids maximal semble le plus pertinent.
Voici cinq objets, leur poids et leur valeur. Vous disposez d'un sac de capacité maximale de 32 kg.

Objet	Poids (kg)	Valeur (euros)	Ratio Valeur/poids
R	16	512	32
B	15	495	33
C	19	950	50
D	14	280	20
E	14	420	30

S={R,B}

53

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.4 Graphe de décision & Algo. Dijkstra

L'algorithme de Dijkstra est un algorithme de recherche de chemin utilisé en théorie des graphes. Il a été développé par l'informaticien néerlandais Edsger W. Dijkstra en 1956 et il est utilisé pour trouver le chemin le plus court entre deux nœuds dans un graphe.

L'algorithme fonctionne en attribuant des distances provisoires à tous les nœuds du graphique, puis en sélectionnant de manière itérative le nœud avec la plus petite distance provisoire, en mettant à jour les distances provisoires de ses voisins et en le marquant comme visité.

Le problème de trouver les chemins les plus courts d'un sommet source à tous les autres sommets dans le graphe.

54

```
dist[s] ←0                                  (la distance au sommet source est nulle)
For all v ∈ V–{s}                           (régler toutes les autres distances sur l'infini)
        do dist[v] ←∞
S←∅                                         (S, l'ensemble des sommets visités est initialement vide)
Q←V                                         (Q, la file d'attente contient initialement tous les sommets)
while Q ≠∅                                  (tant que la file d'attente n'est pas vide)
do u ← mindistance(Q,dist)                  (sélectionnez l'élément de Q avec la distance min.)
        S←S∪{u}                             (ajoutez-vous à la liste des sommets visités)
        for all v ∈ neighbors[u]
                do if dist[v] > dist[u] + w(u, v)   (si nouveau chemin le plus court trouvé)
                        then d[v] ←d[u] + w(u, v)   (définir la nouvelle valeur du chemin le plus court)
return dist
```

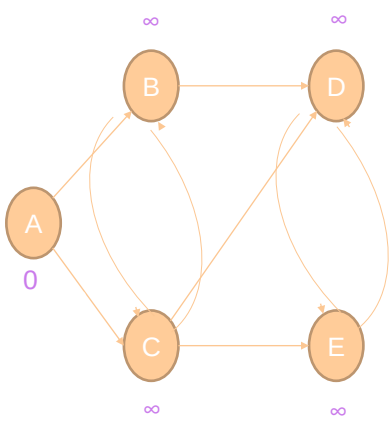
55

Exemple Animé

Q :

A	B	C	D	E

S={}



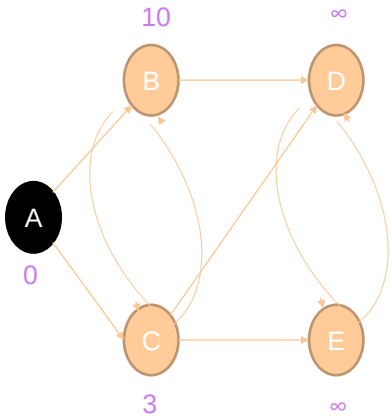
56

Exemple Animé

Q :

A	B	C	D	E
0	∞	∞	∞	∞

S={}

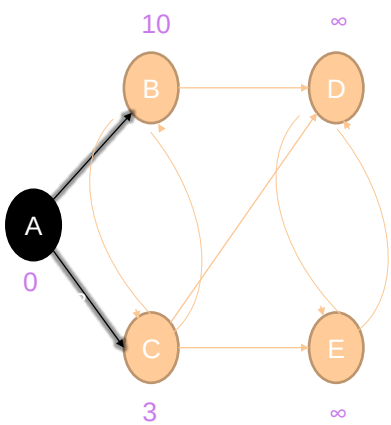


Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0	∞	∞	∞	∞

S={A}

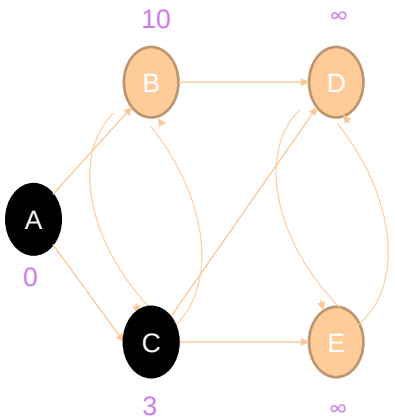


Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		

S={A,C}

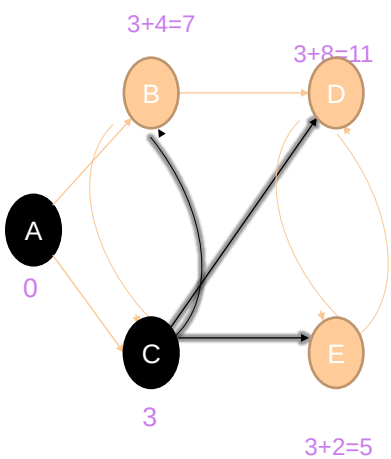


Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		

S={A,C}



Chapitre 1: Introduction à la théorie des graphes

V. Applications

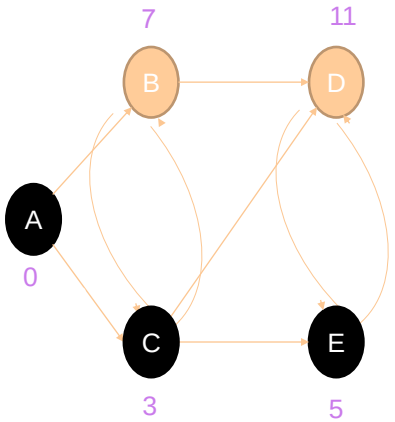
5.4 Graphe de décision & Algo. Dijkstra

Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		
				5

S={A,C,E}



61

Chapitre 1: Introduction à la théorie des graphes

V. Applications

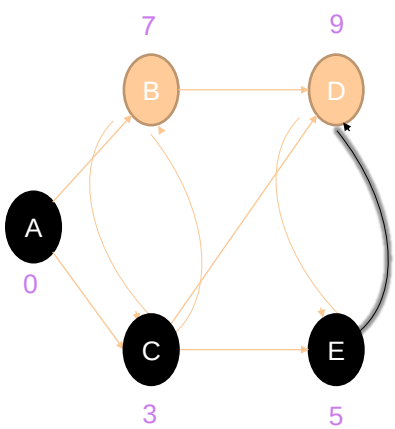
5.4 Graphe de décision & Algo. Dijkstra

Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		
				5
	7			

S={A,C,E}

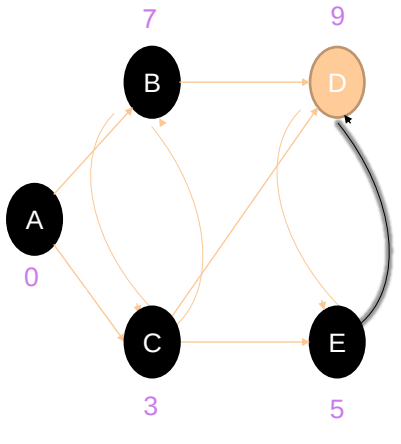


62

Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		
				5
	7			

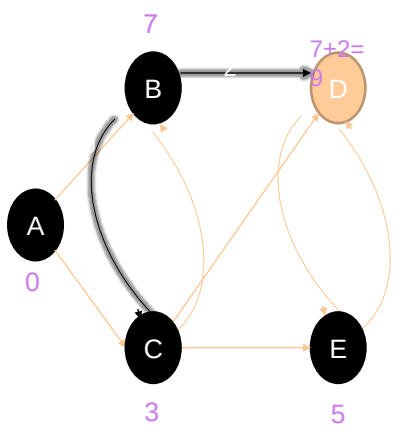


S={A,C,E,B}

Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		
				5
	7			
			9	



S={A,C,E,B}

Chapitre 1: Introduction à la théorie des graphes

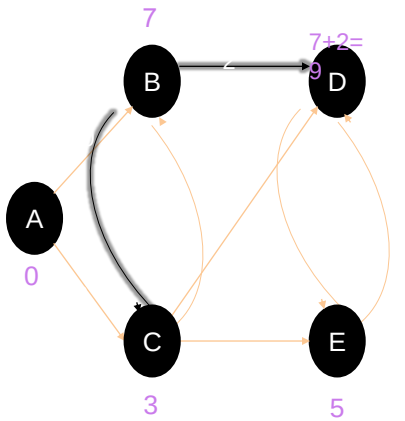
V. Applications

5.4 Graphe de décision & Algo. Dijkstra

Exemple Animé

Q : get shortest path from to :

A	B	C	D	E
0				
		3		
				5
	7			
			9	



S={A,C,E,B,D}

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.4 Graphe de décision & Algo. Dijkstra

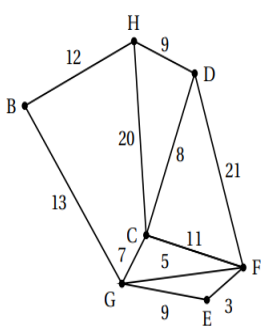
Exercice d'application:

Des touristes sont logés dans un hôtel H. Un guide souhaite faire visiter la région à ces touristes en empruntant les routes signalées comme d'intérêt touristique par l'office du tourisme. Les tronçons de route qu'il souhaite emprunter sont représentés sur le graphe ci-contre. Le long de chaque arête figure la distance en kilomètres des différents tronçons.

1.a. Le guide peut-il emprunter tous les tronçons de route en passant une et une seule fois sur chacun d'eux, en partant de l'hôtel et en y revenant ? Justifier la réponse.

1. b. Le guide peut-il emprunter tous les tronçons de route en passant une et une seule fois sur chacun d'eux, en partant de l'hôtel mais sans forcément y revenir ? Justifier la réponse.

2. Un musée est situé en E. Déterminer le plus court chemin menant de l'hôtel H au musée E. Justifier la réponse



Chapitre 1: Introduction à la théorie des graphes

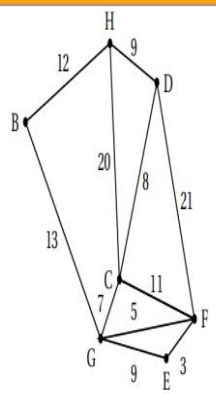
V. Applications

5.4 Graphe de décision & Algo. Dijkstra

Rép question 1.a) D'après le théorème d'EULER, un graphe connexe possède un cycle eulérien si et seulement si tous les sommets sont de degrés pairs.

• Le graphe est connexe car la chaîne BGEFCDH par exemple contient tous les sommets du graphe (non orienté).

• On cherche les degrés des sommets :



Sommet	H	B	C	D	E	F	G
Degrés	3	2	4	3	2	4	4

➔Il y a deux sommets de degrés impairs, donc il n'y a pas de cycle eulérien dans ce graphe.

• Conclusion : le guide ne peut pas emprunter tous les tronçons de route en passant une et une seule fois sur chacun d'eux, en partant de l'hôtel et en y revenant.

67

Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.4 Graphe de décision & Algo. Dijkstra

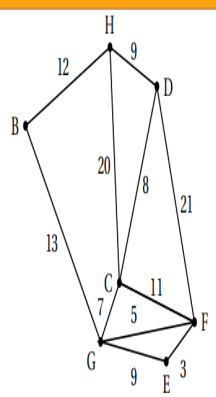
Rép question 1.b) Le guide souhaite partir de l'hôtel et parcourir tous les tronçons de route sans forcément revenir à l'hôtel ; il s'agit alors de trouver une chaîne eulérienne dans ce graphe.

D'après le théorème d'EULER, un graphe connexe contient une chaîne eulérienne si et seulement si exactement zéro ou deux de ses sommets sont de degrés impairs.

C'est le cas ici car pour ce graphe connexe, seuls les sommets H et D sont de degrés impairs ;

➔on peut donc trouver un parcours partant de H et arrivant à D passant une et une seule fois par chaque tronçon de route. Voici un tel parcours :

H −B −G −C − H −D −C −F −G −E −F −D



68

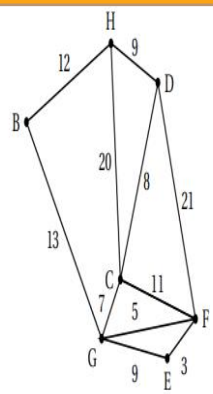
Chapitre 1: Introduction à la théorie des graphes

V. Applications

5.4 Graphe de décision & Algo. Dijkstra

Rép question 2)

Départ	B	C	D	E	F	G	On garde
H	12H	20H	9H	∞	∞	∞	D(H)
D(9)	12H	~~20H~~ 17D		∞	30D	∞	B(H)
B(12)		17D		∞	30D	25B	C(D)
C(17)				∞	~~30D~~ 28C	~~25B~~ 24C	G(C)
G(24)				33G	28C ~~39G~~		F(C)
F(28)				~~33G~~ 31F			E(F)



Le chemin de longueur minimale 31 km entre H et E est :
H −9→ D −8→ C −11→ F −3→ E

69