Algorithmique Avancée et Complexité

Cours du Master "Ingénierie du Logiciel" 1ère année

Dr Amar ISLI
Département d'Informatique
Faculté d'Electronique et d'Informatique
Université des Sciences et de la Technologie Houari Boumediène
BP 32, El-Alia, Bab Ezzouar
DZ-16111 ALGER

http://www.usthb.dz/perso/info/aisli/#TA

CHAPITRE I
Introduction

Problème et instance d'un problème

- Définition d'un problème

Un problème est une description abstraite à laquelle est associée une question nécessitant une réponse.

CHAPITRE I
Introduction

Problème et instance d'un problème

- Exemple :
  - le problème du voyageur de commerce
  - ou TSP (Traveling Salesman Problem)

  Description : un graphe (non orienté) G=<V,E> dont les arêtes sont étiquetées par des coûts. Les nœuds correspondent aux villes à visiter, et le coût d'une arête peut, par exemple, correspondre à la distance séparant les deux villes qu'elle relie

CHAPITRE I
Introduction

Problème et instance d'un problème

Exemple :

- le problème du voyageur de commerce

  Question : calcul d'un cycle $(i_1,\ldots,i_n,i_1)$ de coût minimal, partant d'une ville i_1 et visitant chacune des autres villes exactement une fois avant de revenir à la ville de départ i_1

  - Cycle hamiltonien de coût minimal

CHAPITRE I
Introduction

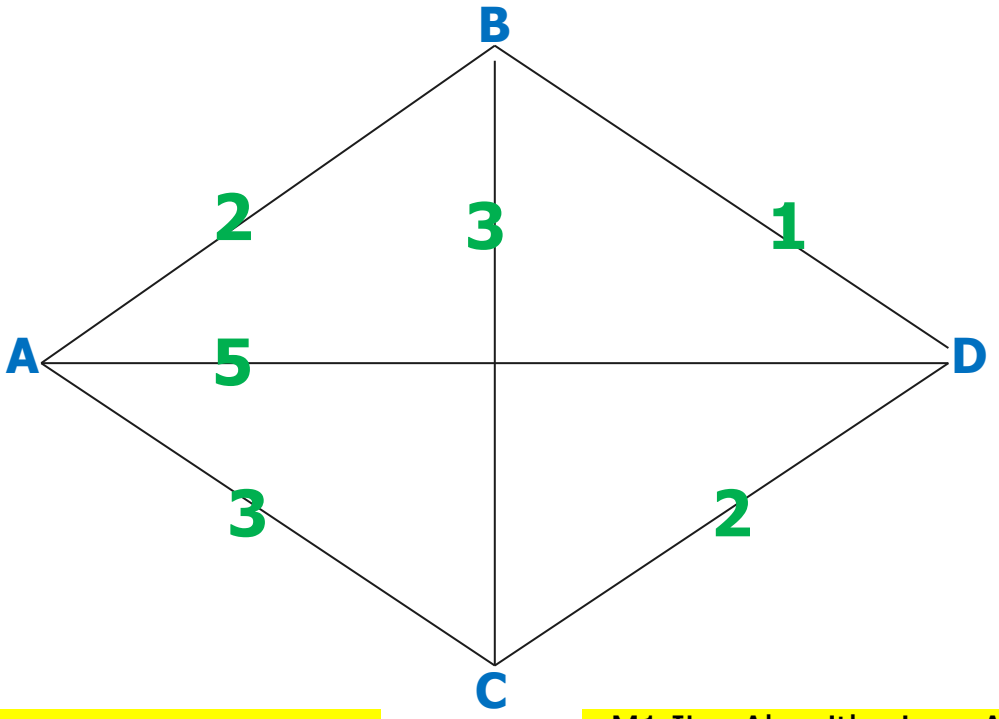
Problème et instance d'un problème

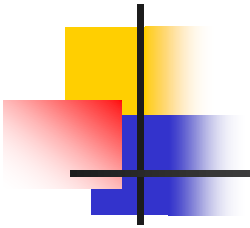
- Instance d'un problème
  - Une instance d'un problème est une spécialisation du problème obtenue en donnant une spécification exacte des données
- Exemple : une instance du TSP
  - G=<V,E> avec
    - V={A,B,C,D}
    - E={(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)}
    - c(A,B)=2; c(A,C)=3; c(A,D)=5; c(B,C)=3; c(B,D)=1; c(C,D)=1

CHAPITRE I
Introduction

Problème et instance d'un problème

- Exemple : une instance du TSP





CHAPITRE I
Introduction

Remarque

- Un programme, qui est une implémentation d'un algorithme, est conçu pour un problème donné
- Une exécution d'un programme nécessite comme entrée une instance du problème associé ; en sortie, l'exécution fournit une réponse à la question associée au problème adaptée à l'instance en question

CHAPITRE I
Introduction

Taille d'un problème

La taille d'un problème est la taille de l'espace mémoire nécessaire à sa représentation ; elle est mesurée en fonction des données du problème

- Exemple : dans le cas du TSP, les données sont
  - Le nombre de villes à visiter
  - Le nombre d'arêtes
  - Les coûts des différentes arêtes

CHAPITRE I
Introduction

Taille d'un problème

- Une instance du TSP peut, par exemple, être représentée par une matrice nxn, n étant le nombre de villes à visiter
- Les éléments d'une telle matrice seront les coûts des différentes arêtes
  - La matrice est symétrique : les éléments (i,j) et (j,i) sont égaux
  - La diagonale de la matrice est à 0 : les éléments (i,i) valent 0
  - Si deux villes i et j ne sont pas reliées par une arête, les éléments (i,j) et (j,i) seront mis à $+\infty$

CHAPITRE I
Introduction

Modèles de calcul

- Machine de Turing
- RAM (Random Access Machine)

- Un calcul = ensemble fini d'étapes élémentaires
- Au niveau de chaque étape, une action est choisie parmi un ensemble fini d'actions
- Déterministe : au niveau de chaque étape, au plus une action est possible
- Non déterministe : existence d'étapes avec plus d'une action possible

CHAPITRE I
Introduction

Autres modèles de calcul

- Les fonctions récursives
- Le lambda-calcul

CHAPITRE I
Introduction

Algorithme et complexité

- Algorithme

  Un algorithme est une suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème

CHAPITRE I
Introduction

Algorithme et complexité

- Complexité
  - La complexité d'un algorithme est la mesure du nombre d'opérations élémentaires qu'il effectue sur le problème pour lequel il a été conçu
  - La complexité est mesurée en fonction de la taille du problème ; la taille étant elle-même mesurée en fonction des données du problème
  - La complexité est par conséquent mesurée en fonction des données du problème

CHAPITRE I
Introduction

Complexité

- Exemple :

  dans le cas du TSP, la complexité va être mesurée en fonction des données du problème, qui sont
  - Le nombre de villes à visiter
  - Le nombre d'arêtes
  - L'espace nécessaire (en termes de bits par exemple) pour la représentation du coût d'une arête

CHAPITRE I
Introduction

Complexité

- Complexité au meilleur
  - Best-case complexity
- Complexité en moyenne
  - Average-case complexity
- Complexité au pire
  - Worst-case complexity

CHAPITRE I
Introduction

Notations de Landau

- On ne mesure généralement pas la complexité exacte d'un algorithme
- On mesure son ordre de grandeur qui reflète son comportement asymptotique, c'est-a-dire son comportement sur les instances de grande taille

CHAPITRE I
Introduction

Notations de Landau

Besoin des notations asymptotiques de Landau :

- $f=O(g)$ ssi il existe n_0, il existe $c\geq 0$,
  pour tout $n\geq n_0$, $f(n)\leq cxg(n)$
- $f=\Omega(g)$ ssi $g=O(f)$
- $f=o(g)$ ssi pour tout $c\geq 0$, il existe n_0,
  pour tout $n\geq n_0$, $f(n)\leq cxg(n)$
- $f=\Theta(g)$ ssi $f=O(g)$ et $g=O(f)$

CHAPITRE I
Introduction

Illustration

- tri par insertion
  - **Entrée :** Un tableau de n nombres
  - **Sortie :** Le tableau trié par ordre croissant

CHAPITRE I
Introduction

Illustration : tri par insertion

```
TRI-INSERTION{
1.    Pour j=2 à n{
2.              clé= A[j]
3.              i=j-1
4.              Tant que i>0 et A[i]>clé{
5.                                  A[i+1]=A[i]
6.                                  i=i-1
7.                                  }
8.              A[i+1]=clé
9.              }
10.   }
```

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :

Instruction	Nombre d'opérations élémentaires
1	n*(1 incrémentation+1 comparaison)
2	(n-1)*(1 affectation)
3	(n-1)*(1 décrémentation+1 affectation)
4	$\sum_{j=2,n}$(j-1)*(2 comparaisons)
5	$\sum_{j=2,n}$ (j-1)*(1 affectation)
6	$\sum_{j=2,n}$ (j-1)*(1 décrémentation+1 affectation)
8	(n-1)*(1 affectation)

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :

- les opérations élémentaires de l'algorithme sont l'incrémentation, la décrémentation, l'affectation, la comparaison
- On suppose que le temps d'exécution de l'opération élémentaire o_i est borné par une constante c_i
- Le maximum des c_i donne une constante bornant le temps d'exécution d'une instruction élémentaire quelconque
- Moralité : dans le calcul du nombre d'opérations dans le pire cas d'un algorithme, on peut ne pas différencier entre les différentes opérations élémentaires telles que l'incrémentation, la décrémentation, l'affectation, la comparaison

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :

Instruction	Nombre d'opérations élémentaires
1	n*(2 opérations)
2	(n-1)*(1 opération)
3	(n-1)*(2 opérations)
4	$\sum_{j=2,n}$(j-1)*(2 opérations)
5	$\sum_{j=2,n}$ (j-1)*(1 opération)
6	$\sum_{j=2,n}$ (j-1)*(2 opérations)
8	(n-1)*(1 opération)

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :

$f(n)=2*n+1*(n-1)+2*(n-1)+5*(\sum_{j=2,n}(j-1))+1*(n-1)$

$=2n+4(n-1)+5\sum_{j=1,n-1}j$

$=6n-4+5*(n-1)n/2$

$=(5n^2+7n-8)/2$

Instruction	Nombre d'opérations élémentaires
1	n*(2 opérations)
2	(n-1)*(1 opération)
3	(n-1)*(2 opérations)
4	$\sum_{j=2,n}$(j-1)*(2 opérations)
5	$\sum_{j=2,n}$ (j-1)*(1 opération)
6	$\sum_{j=2,n}$ (j-1)*(2 opérations)
8	(n-1)*(1 opération)

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme : $f(n)=(5n^2+7n-8)/2$

On montre que $f(n)=\Theta(n^2)$, chose qui se scinde en deux points :

- montrer que $f(n)=O(n^2)$
- montrer que $n^2=O(f(n))$

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :
$f(n)=(5n^2+7n-8)/2$

On montre que $f(n)=O(n^2)$: il faut trouver n_0 et $c\geq 0$ tels que $\forall n\geq n_0$ $f(n)\leq c*n^2$, i.e. $(5n^2+7n-8)/2 \leq c*n^2$

- $c\geq 5+(7n-8)/2*n^2$
- $(7n-8)/2*n^2$ tend vers 0^+ quand n tend vers l'infini : ceci suffit pour dire qu'il existe n_0 et $c\geq 0$ tels que $\forall n\geq n0$ $f(n)\leq c*n^2$
- La constante c, on peut la fixer à 6 : $c=6$
- Pour exhiber une valeur pour n_0, il faut trouver la valeur de x à partir de laquelle la fonction $h(x)=(7x-8)/2*x^2$ va commencer à décroitre : $h'(x)=(-7x+16)/2n^3$, donc h décroissante sur l'intervalle $]16/7,+\infty[$
- Il est clair maintenant qu'on peut prendre $n_0=3$ (l'entier immédiatement supérieur à 16/7), ou n_0 égal à n'importe quelle autre constante entière supérieure ou égale à 3 : à noter que $h(3)=13/18<1$

CHAPITRE I
Introduction

Illustration : tri par insertion

Nombre d'opérations élémentaires, dans le pire des cas, de l'algorithme :
$f(n)=(5n^2+7n-8)/2$

On montre que $n^2=O(f(n))$: il faut trouver n_0 et $c\geq 0$ tels que $\forall n\geq n_0$ $n^2\leq c*f(n)$, i.e. $n^2\leq c*(5n^2+7n-8)/2$

- $1/c\leq 5+(7n-8)/2*n^2$
- $(7n-8)/2*n^2$ tend vers 0^+ quand n tend vers l'infini : ceci suffit pour dire qu'il existe n_0 et $c\geq 0$ tels que $\forall n\geq n0$ $1/c\leq 5+(7n-8)/2*n^2$
- La constante c, on peut la fixer de telle sorte que $1/c=5$: $c=1/5$
- Pour exhiber une valeur pour n_0, il faut trouver la valeur de x à partir de laquelle la fonction $h(x)=(7x-8)/2*x^2$ va être décroissante : $h'(x)=(-7x+16)/2n^3$, donc h décroissante sur l'intervalle $]16/7,+\infty[$. Comme h(x) tend vers 0^+ quand n tend vers l'infini, on a bien $h(x)>0$ $\forall x\in]16/7,+\infty[$
- Il est clair maintenant qu'on peut prendre $n_0=3$ (l'entier immédiatement supérieur à 16/7)

CHAPITRE I
Introduction

Illustration : machine de Turing déterministe

$M=<S,Q,I,q_0,q_f>$

$S=\{0,1\}$

$Q=\{q_0,q_1,q_f\}$

$I=\{$(1) $q_0 1 D q_0$,

(2) $q_0 0 1 q_1$,

(3) $q_1 1 G q_1$,

(4) $q_1 0 D q_f\}$

CHAPITRE I
Introduction

Illustration : machine de Turing déterministe

- M calcule la fonction successeur s : s(n)=n+1
- Nombre de déplacements de la tête de lecture en fonction du nombre de barres initiales (taille de l'entrée) :
  - f(n)=2n+4

CHAPITRE I
Introduction

Illustration : machine de Turing non déterministe

$M=<S,Q,I,q_0,q_f>$

$S=\{0,1\}$

$Q=\{q_0,q_1,q_f\}$

$I=\{$(1) $q_0 1 D q_0$,

(2) $q_0 0 1 q_1$,

(3) $q_1 1 G q_1$,

(4) $q_1 0 D q_f$,

(5) $q_1 0 1 q_f\}$

CHAPITRE II
Structures de données élémentaires

A. Piles et files

Définition (pile) :

- Une pile est une structure de données mettant en œuvre le principe « dernier entré premier sorti »
- LIFO : Last In First Out

CHAPITRE II
Structures de données élémentaires

Une pile P peut être implémentée par un tableau, et elle est caractérisée par :

- Un sommet noté sommet(P) indiquant l'indice de l'élément le plus récemment inséré dans la pile
- Un caractère spécial, comme $, initialisant la pile
- Une procédure EMPILER(P,x)
- Une fonction DEPILER(P)
- Une fonction booléenne PILE-VIDE(P) retournant VRAI si et seulement si P est vide

```
PILE-VIDE(P){
    Si sommet(P)=$
    alors retourner VRAI
    sinon retourner FAUX
    }
```

CHAPITRE II
Structures de données élémentaires

```
EMPILER(P,x){
    Si sommet(P)=longueur(P)
                alors erreur (débordement positif)
                sinon{      sommet(P)=sommet(P)+1
                            P[sommet(P)]=x
                            }
    }
DEPILER(P){
    Si PILE-VIDE(P)
                alors erreur (débordement négatif)
                sinon{      sommet(P)=sommet(P)-1
                            retourner P[sommet(P)+1]
                            }
    }
```

CHAPITRE II
Structures de données élémentaires

A. Piles et files

Définition (file) :

- Une file est une structure de données mettant en œuvre le principe « premier entré premier sorti »
- FIFO : First In First Out

CHAPITRE II
Structures de données élémentaires

Une file F peut être implémentée par un tableau, et elle est caractérisée par :

- Un pointeur tête(F) qui pointe vers la tête de la file (l'élément le plus anciennement inséré)
- Un pointeur queue(F) qui pointe vers la première place libre, où se fera la prochaine insertion éventuelle d'un élément
- Initialement : tête(F)=NIL et queue(F)=1

CHAPITRE II
Structures de données élémentaires

```
FILE-VIDE(F){
    si tête(F)=NIL
      alors retourner VRAI
      sinon retourner FAUX
    }
INSERTION(F,x){
    si [tête(F)≠NIL et queue(F)=tête(F)]
      alors erreur (débordement positif)
      sinon{
                F[queue(F)]=x
                queue(F)=[queue(F)+1](modulo n)
                si[tête(F)=NIL] alors tête(F)=1
                }
    }
```

CHAPITRE II
Structures de données élémentaires

```
SUPPRESSION(F){
    si FILE-VIDE(F)
      alors erreur (débordement négatif)
      sinon{
                temp=F(tête(F));
                tête(F)=[tête(F)+1](modulo n);
                si[tête(F)=queue(F)]{
                          tête(F)=NIL ;
                          queue(F)=1;
                          }
                retourner temp;
                }
    }
```

CHAPITRE II
Structures de données élémentaires

A. Piles et files

- EXEMPLES
  - Pile d'exécution
  - Automate à pile : $a^n c b^n$
  - Simulation de la file d'attente d'un magasin

CHAPITRE II
Structures de données élémentaires

B. Listes chaînées

Définition (liste chaînée) :

- Une liste chaînée est une structure de données dont les éléments sont arrangés linéairement, l'ordre linéaire étant donné par des pointeurs sur les éléments
- Un élément d'une liste chaînée est un enregistrement contenant un champ *clé*, un champ *successeur* consistant en un pointeur sur l'élément suivant dans la liste
- Si le champ *successeur* d'un élément vaut NIL, l'élément est le dernier élément de la liste, appelé également *queue* de la liste
- Un pointeur TETE(L) est associé à une liste chaînée L : il pointe sur le premier élément de la liste
- Si une liste chaînée L est telle que TETE(L)=NIL alors la liste est vide

CHAPITRE II
Structures de données élémentaires

B. Listes chaînées

- **Exemple**
  - Liste chaînée contenant initialement 20, 7 et 10
    - INSERTION(30)
    - SUPPRESSION(7)

Liste doublement chaînée

Liste chaînée triée

Liste chaînée circulaire (anneau)

CHAPITRE II
Structures de données élémentaires

B. Listes chaînées

- **Algorithmes de manipulation de listes simplement chaînées**

```
RECHERCHE-LISTE(L,k){
    x=TETE(L);
    tant que(x!=NIL et clé(x)!=k)x=successeur(x);
    retourner x;
    }

INSERTION-LISTE(L,X){
    successeur(x)=TETE(L);
    TETE(L)=x;
    }
```

CHAPITRE II
Structures de données élémentaires

B. Listes chaînées

- **Algorithmes de manipulation de listes simplement chaînées**

```
SUPPRESSION-LISTE(L,X){
    Si x=TETE(L) alors TETE(L)=successeur(x)
    sinon{
      y=TETE(L);
      tant que successeur(y)<>x faire y=successeur(y);
      successeur(y)=successeur(x);
      }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

A. Graphes

- **Graphe orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arcs)
- **Graphe non orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arêtes)
  - Arcs : un arc est orienté (couple)
  - Arêtes : une arête n'est pas orientée (paire)
  - Possibilité d'avoir des boucles (une boucle est un arc reliant un sommet à lui-même)
  - Impossibilité d'avoir des boucles
  - Un arc (u,v) *part* du sommet u et *arrive* au sommet v
  - Une arête (u,v) est *incidente* aux sommets u et v
  - Degré sortant d'un sommet : nombre d'arcs en partant
  - Degré ®entrant d'un sommet : nombre d'arcs y arrivant
  - Degré=degré sortant+degré entrant
  - Degré d'un sommet : nombre d'arêtes qui lui sont incidentes

CHAPITRE III
Parcours des graphes et des arbres

A. Graphes

- **Graphe orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arcs)
- **Graphe non orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arêtes)
  - Chemin de degré k d'un sommet u à un sommet v : $(u_0,u_1,\ldots,u_k)$ $u=u_0$ et $v=u_k$
  - Chemin élémentaire
  - Chaîne et chaîne élémentaire
  - Circuit et circuit élémentaire
  - Cycle et cycle élémentaire
  - Graphe sans circuit
  - Graphe acyclique

CHAPITRE III
Parcours des graphes et des arbres

A. Graphes

- **Graphe orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arcs)
- **Graphe non orienté :** couple (S,A), S ensemble fini (sommets) et A relation binaire sur S (arêtes)
  - Graphe fortement connexe : tout sommet est accessible à partir de tout autre sommet (par un chemin)
  - Graphe connexe : chaque paire de sommets est reliée par une chaîne
  - Composantes fortement connexes d'un graphe : classes d'équivalence de la relation définie comme suit sur l'ensemble des sommets : $R(s_1,s_2)$ si et seulement si il existe un chemin de s_1 vers s_2 et un chemin de s_2 vers s_1
  - Composantes connexes d'un graphe : classes d'équivalence de la relation définie comme suit sur l'ensemble des sommets : $R(s_1,s_2)$ si et seulement si il existe une chaîne de s_1 vers s_2

CHAPITRE III
Parcours des graphes et des arbres

B. Arbres

- **Forêt :**
  - graphe non orienté acyclique
- **Arbre :**
  - graphe non orienté acyclique connexe (forêt connexe)

CHAPITRE III
Parcours des graphes et des arbres

B. Arbres

- G=(S,A) graphe non orienté :
  - G arbre
  - G connexe et |A|=|S|-1
  - G acyclique et |A|=|S|-1
  - Deux sommets quelconques sont reliés par une une unique chaîne élémentaire

CHAPITRE III
Parcours des graphes et des arbres

B. Arbres

- **Arbre enraciné** (rooted tree) :
  - un sommet se distingue des autres (la racine de l'arbre)
  - La racine d'un arbre enraciné impose un sens de parcours de l'arbre
  - Pour un arbre enraciné : sommets ou nœuds
  - Ancêtre, père, fils, descendant
  - L'unique nœud sans père : la racine
  - Nœuds sans fils : nœuds externes ou feuilles

CHAPITRE III
Parcours des graphes et des arbres

B. Arbres

- Nœuds avec fils : nœuds internes
- Sous-arbre de racine x : l'arbre composé des descendants de x, enraciné en x
- Degré d'un nœud : nombre de fils
- Profondeur d'un nœud x : longueur du chemin entre la racine et le nœud
- Hauteur d'un arbre
- Arbre ordonné

CHAPITRE III
Parcours des graphes et des arbres

B. Arbres

- Arbre binaire (description récursive) :
  - Ne contient aucun nœud (arbre vide)
  - Est formé de trois ensembles disjoints de nœuds : la racine ; le sous-arbre gauche (arbre binaire) ; le sous-arbre droit (arbre binaire)
- Un arbre binaire est plus qu'un arbre ordonné
- Arbre binaire complet
- Arbre n-aire : degré d'un nœud≤n

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre ordonné

- Parcours en profondeur d'abord
  - Descendre le plus profondément possible dans l'arbre
  - Une fois une feuille atteinte, remonter pour explorer les branches non encore explorées, en commençant par la branche la plus basse
  - Les fils d'un nœud sont parcourus suivant l'ordre défini sur l'arbre

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre ordonné

- Parcours en profondeur d'abord

```
PP(A){
   Si Arbre(A) n'est pas réduit à l'arbre vide{
     Pour tous les fils u de racine(A) faire dans l'ordre
                PP(u)
     }
   }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Structure de données récursive permettant la représentation d'un arbre binaire

```
Type arbre-b{
        valeur : entier;
        sa-gauche : pointeur sur type arbre-b;
        sa-droit : pointeur sur type arbre-b;
        }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Déclarer une variable pointeur sur type arbre-b, qui va être la racine de l'arbre :

Variable racine : pointeur sur type arbre-b;

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Parcours en profondeur d'abord

```
PP(A){
   Si A≠NIL{
     PP(A.sa-gauche);
     PP(A.sa-droit);
     }
   }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Parcours en profondeur d'abord

```
Préfixe(A){
    Si A≠NIL{
      Afficher(A.valeur);
      Préfixe(A.sa-gauche);
      Préfixe(A.sa-droit);
      }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Parcours en profondeur d'abord

```
Infixe(A){
    Si A≠NIL{
      Infixe(A.sa-gauche);
      Afficher(A.valeur);
      Infixe(A.sa-droit);
      }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Parcours en profondeur d'abord

```
Postfixe(A){
    Si A≠NIL{
      Postfixe(A.sa-gauche);
      Postfixe(A.sa-droit);
      Afficher(A.valeur);
      }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre binaire

- Parcours en profondeur d'abord
  - Exemple
    - Utilisation d'un arbre binaire pour le tri d'un tableau par ordre croissant : 50; 40; 45; 60; 30; 55; 44; 100
      - A gauche si ≤; à droite sinon
      - Parcours infixe de l'arbre pour l'affichage du résultat du tri

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre ordonné

- Parcours en largeur d'abord
  - Visiter tous les nœuds de profondeur i avant de passer à la visite des nœuds de profondeur i+1
  - Utilisation d'une file

CHAPITRE III
Parcours des graphes et des arbres

C. Parcours d'un arbre ordonné

- Parcours en largeur d'abord

```
PL(A){
    Si A≠NIL{
      INSERTION(F,A);
      Tant que (non FILE-VIDE(F)){
                u=SUPPRESSION(F);          //visiter le noeud
                Pour tous les fils v de u (dans l'ordre) faire
                        INSERTION(F,v);    //faire attendre le noeud
                }
      }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Initialement, tous les sommets sont blancs
- Lorsqu'un sommet est rencontré pour la première fois, il est colorié en gris
- Lorsque tous les successeurs d'un sommet, dans l'ordre de parcours, ont été visités, le sommet est colorié en noir

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Parcours en profondeur d'abord

```
PP(G){
    Pour chaque sommet u de G{
            couleur[u]=BLANC;
            }
    Tant que G a des sommets blancs{
            prendre un sommet blanc u;
            VISITER-PP(G,u,couleur);
            }
    }
VISITER-PP(G,s,couleur){
            couleur[s]=GRIS;
            Pour chaque voisin v de s{si couleur[v]=BLANC{VISITER-PP(G,v,couleur);}}
            couleur[s]=NOIR;
            }
```

CHAPITRE III
Parcours des graphes et des arbres

- Forêt de recouvrement (spanning forest)
  - Forêt à un seul arbre : arbre de recouvrement (exemple à dérouler)
- Graphe orienté : complexité de la recherche des composantes fortement connexes
- Graphe non orienté : Complexité de la recherche des composantes connexes
  - Transformer en graphe orienté
  - Recherche des composantes fortement connexes

CHAPITRE III
Parcours des graphes et des arbres

Forêt de recouvrement (spanning forest) :

```
Foret_de_recouvrement(G){
    Pour chaque sommet u de G{
        couleur[u]=BLANC;
        fils(u)=∅
        }
    racines=∅
    Tant que G a des sommets blancs{
        prendre un sommet blanc u; racines=racines∪{u} ; VISITER-PP(G,u,couleur);
        }
    }
VISITER-PP(G,s,couleur){
        couleur[s]=GRIS;
        Pour chaque voisin v de s
            si couleur[v]=BLANC{fils(s)=fils(s)∪{v} ; VISITER-PP(G,v,couleur);}
        couleur[s]=NOIR;
        }
```

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Parcours en largeur d'abord d'un graphe

```
PL(G,s){
    couleur[s]=GRIS;
    Pour chaque sommet u≠s de G{couleur[u]=BLANC;}
    F={s};
    Tant que F≠Φ{
        u=SUPPRESSION(F);
        Tant que u a des voisins blancs{
            prendre un voisin blanc v de u;
            couleur[v]=GRIS;
            INSERTION(F,v);
            }
        couleur[u]=NOIR;
        }
    }
```

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Parcours en largeur d'abord d'un graphe : dérouler l'algorithme sur un exemple

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

Recherche des composantes fortement connexes

- Graphe orienté vu comme relation sur l'ensemble des sommets
- Fermeture réflexive d'une relation R
  - La plus petite relation réflexive contenant R (notée R^1)
- Fermeture transitive d'une relation R
  - La plus petite relation transitive contenant R (notée R^+)
- Fermeture réflexive-transitive d'une relation R
  - La plus petite relation réflexive et transitive contenant R (notée R^*)

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Recherche des composantes fortement connexes :

Soit G=(S,A) un graphe :

- Voir A comme relation binaire sur S
- Calculer la fermeture réflexive-transitive A^* de A comme suit :
  - $A^0=\{(s,s) : s\in S\}$
  - $A^1=A\cup A^0$
  - $A^2=A^1\cup(A^1)^2= A^1\cup\{(x,y)\in S^2 : \exists z\in S/(x,z)\in A^1 \text{ et } (z,y)\in A^1\}$
  - Pour tout i=2,n :
    - $A^i=A^{i-1}\cup(A^{i-1})^2=A_{i-1}\cup\{(x,y)\in S^2 : \exists z\in S/(x,z)\in A^{i-1} \text{ et } (z,y)\in A^{i-1}\}$
  - $A^*=\lim_{n->\infty} A^n=\bigcup_{i\in IN} A^i$

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Calcul de la fermeture réflexive-transitive d'un graphe orienté
- Utilisation de la matrice d'adjacence du graphe
  - M_G matrice carrée booléenne nxn, n nombre de sommets de G

$$M_G[i,j] = \begin{cases} 1 \text{ si } (X_i, X_j) \text{ sommet de G} \\ 0 \text{ sinon} \end{cases}$$

- Algorithme de Roy-Warshall de complexité $O(n^3)$

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Algorithme de Roy-Warshall :

```
Procedure ft_rw(G)
Début
Pour i=1 à n faire M_G[i,i]=1 fait
Pour k=1 à n faire
    Pour i=1 à n faire
        Pour j=1 à n faire
            M_G[i,j]=M_G[i,j]∨M_G[i,k]∧M_G[k,j]
        Fait
    Fait
Fait
fin
```

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Algorithme de Roy-Warshall :

```
Procedure ft_rw(G)
Début
Pour i=1 à n faire M_G[i,i]=1 fait
Pour k=1 à n faire
    Pour i=1 à n faire
        Pour j=1 à n faire
            si M_G[i,k]=1 et M_G[k,j]=1 alors M_G[i,j]=1 finsi
        Fait
    Fait
Fait
fin
```

CHAPITRE III
Parcours des graphes et des arbres

D. Parcours d'un graphe

- Algorithme de Roy-Warshall : amélioration

```
Procedure ft_rw(G)
Début
Pour i=1 à n faire M_G[i,i]=1 fait
Pour k=1 à n faire
    Pour i=1 à n faire
        si M_G[i,k]=1 alors
            Pour j=1 à n faire M_G[i,j]=M_G[i,j]∨M_G[k,j] Fait
        finsi
    Fait
Fait
fin
```

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x * x^{n-1} & sinon \end{cases}$$

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x * x^{n-1} & sinon \end{cases}$$

CHAPITRE IV
Récursivité et technique « diviser pour régner »

D. Itérativité versus récursivité

- Récursivité simple
  - La fonction puissance :

$$X^n = \begin{cases} 1 & \text{si } n=0 \\ x*x^{n-1} & \text{sinon} \end{cases}$$

- Récursivité multiple
  - Relation de Pascal donnant les combinaisons C(n,p) :

$$C(n,p) = \begin{cases} 1 & \text{si } p=0 \text{ ou } p=n \\ C(n-1,p)+C(n-1,p-1) & \text{sinon} \end{cases}$$

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x * x^{n-1} & sinon \end{cases}$$

CHAPITRE IV
Récursivité et technique « diviser pour régner »

D. Itérativité versus récursivité

- Récursivité mutuelle

Pair(n)= { VRAI si n=0 ; impair(n-1) sinon

impair(n)= { FAUX si n=0 ; pair(n-1) sinon

- Récursivité imbriquée (la fonction d'Ackermann)

A(m,n)= { n+1 si m=0 ; A(m-1,1) si m>0 et n=0 ; A(m-1,A(m,n-1)) sinon

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Exemple : les tours de Hanoï
  - Trois tiges A, B et C sur lesquelles sont enfilés des disques de diamètres tous différents
  - On peut déplacer un seul disque à la fois
  - Il est interdit de poser un disque sur un autre disque plus petit
  - Initialement, tous les disques sont sur la tige A
  - A la fin, tous sont sur la tige C

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

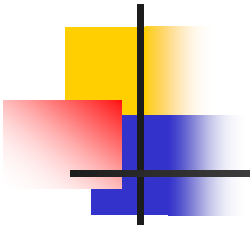
- Exemple : les tours de Hanoï

```
HANOI(n,x,y,z){//x, y et z : départ, intermédiaire, destination
        si n=1 alors déplacer topx vers z
        sinon{  HANOI(n-1,x,z,y);
                déplacer topx vers z;
                HANOI(n-1,y,x,z);
                }
        fin-si
        }
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Exemple : les tours de Hanoï
  - Complexité de l'algorithme en nombre de déplacements de disques :
    - $h(n) = \begin{cases} 1 & \text{si } n=1 \\ h(n-1)+1+h(n-1) & \text{sinon} \end{cases}$
    - $h(n) = \begin{cases} 1 & \text{si } n=1 \\ 1+2 \times h(n-1) & \text{sinon} \end{cases}$
    - $h(n)=2^n-1$
    - $h(n)=O(2^n)=\Theta(2^n)$
  - Complexité exponentielle



CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Il est toujours possible de dérécursiver un algorithme (récursif), c'est-à-dire de transformer un algorithme récursif en un algorithme itératif équivalent

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Dérécursivation du parcours en profondeur d'abord d'un graphe

```
PP(G){
    Pour chaque sommet u de G{
        couleur[u]=BLANC;
        fils(u)=∅
        }
    Tant que G a des sommets blancs{
        prendre un sommet blanc u;
        VISITER-PP(G,u,couleur);
        }
    }
VISITER-PP(G,s,couleur){
        couleur[s]=GRIS;
        Pour chaque voisin v de s{si couleur[v]=BLANC{VISITER-PP(G,v,couleur);}}
        couleur[s]=NOIR;
        }
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Dérécursivation du parcours en profondeur d'abord d'un graphe

```
PP_it( v )
Pour chaque sommet u de G{
        couleur[u]=BLANC;
        }
Initialiser une pile P à vide
Tant que G a des sommets blancs{
        prendre un sommet blanc u
        EMPILER(P,u)
        VISITER-PP-it(G,couleur)
        }
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

F. Itérativité versus récursivité

- Dérécursivation du parcours en profondeur d'abord d'un graphe

```
VISITER-PP-it(G,couleur){
Tant que non PILE_VIDE(P) faire
        s=DEPILER(P)
        Si couleur(s)=BLANC alors
                couleur(s) :=GRIS    //VISITE DU NŒUD s//
                 EMPILER(P,s)
                Pour chaque voisin v de s faire
                        Si couleur(v) =BLANC alors EMPILER(P,v) Fsi
                                //ON FAIT ATTENDRE LES NŒUDS//
                Fait
        sinon couleur(s)=NOIR
        Fsi
Fait
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

G. Technique 'diviser pour régner'

- Diviser le problème à résoudre en un certain nombre de sous-problèmes (plus petits)
- Régner sur les sous-problèmes en les résolvant récursivement :
  - Si un sous-problème est élémentaire (indécomposable), le résoudre directement
  - Sinon, le diviser à son tour en sous-problèmes
- Combiner les solutions des sous-problèmes pour construire une solution du problème initial

CHAPITRE IV
Récursivité et technique « diviser pour régner »

G. Technique 'diviser pour régner'

- Tri par fusion d'un tableau

```
TRI-FUSION(A,p,r){
    si p<r{
        q=⌊(p+r)/2⌋;
        TRI-FUSION(A,p,q);
        TRI-FUSION(A,q+1,r);
        FUSIONNER(A,p,q,r);
        }
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

G. Technique 'diviser pour régner'

- Tri par fusion d'un tableau

```
FUSIONNER(A,p,q,r){
        i=p;j=q+1;k=1;
        tant que i≤q et j≤r{
                si A[i]<A[j]        alors{C[k]=A[i];i=i+1;}
                                    sinon{C[k]=A[j];j=j+1;}fsi
                k=k+1;
                }
        tant que i≤q{C[k]=A[i];i=i+1;k=k+1;}
        tant que j≤r{C[k]=A[j];j=j+1;k=k+1;}
        Pour k=1 à r-p+1{A[p+k-1]=C[k];}
        }
```

CHAPITRE IV
Récursivité et technique « diviser pour régner »

G. Résolution des récurrences 'diviser pour régner' : deux cas

Soient $\boldsymbol{a} \geq 1$ et $\boldsymbol{b} > 1$ deux constantes et f(n) une fonction définie sur IN par $f(n)=\boldsymbol{a}f(n/\boldsymbol{b})+g(n)$, où $n/\boldsymbol{b}$ est interprété comme $\lfloor n/\boldsymbol{b} \rfloor$ ou $\lceil n/\boldsymbol{b} \rceil$. La fonction f(n) peut alors être bornée asymptotiquement comme suit :

- Si $g(n)=O(n^{(\log_{\boldsymbol{b}}\boldsymbol{a})-\varepsilon})$ alors $f(n)= \Theta(n^{(\log_{\boldsymbol{b}}\boldsymbol{a})})$
- Si $g(n)=\Theta(n^{(\log_{\boldsymbol{b}}\boldsymbol{a})})$ alors $f(n)= \Theta(n^{(\log_{\boldsymbol{b}}\boldsymbol{a})}\log n)$

CHAPITRE IV
Récursivité et technique « diviser pour régner »

G. Technique 'diviser pour régner'

- Tri par fusion d'un tableau : complexité

$$T(n)=\begin{cases}\Theta(1) \text{ si } n=1\\ 2T(n/2)+\Theta(n) \text{ sinon}\end{cases}$$

$$T(n)=\Theta(n \log n)$$

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Définition : Un arbre binaire de recherche est un arbre binaire tel que pour tout nœud x :

- Tous les nœuds y du sous-arbre gauche de x vérifient clef(y)<clef(x)
- Tous les nœuds y du sous-arbre droit de x vérifient clef(y)≥clef(x)

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Propriété : le parcours (en profondeur d'abord) infixe d'un arbre binaire de recherche permet l'affichage des clés des nœuds par ordre croissant

```
Infixe(A){
    Si A≠NIL{
      Infixe(A.sa-gauche);
      Afficher(A.clef);
      Infixe(A.sa-droit);
      }
    }
```

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Recherche d'un élément :

- la fonction rechercher(A,k) ci-dessous retourne le pointeur sur un nœud dont la clé est k, si un tel nœud existe
- elle retourne le pointeur NIL sinon

```
rechercher(A,k){
    si (A=NIL ou A.clef=k) retourner A
    sinon
            si (k<A.clef) rechercher(A.sa-gauche,k)
            sinon rechercher(A.sa-droit,k)
    }
```

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Minimum :

- la fonction minimum(A) retourne le pointeur NIL si l'arbre dont la racine est pointée par A est vide
- sinon, elle retourne le pointeur sur un nœud contenant la clé minimale de l'arbre

```
minimum(A){
    si (A=NIL) retourner A
    sinon{
       x=A;
       tant que (x.sa-gauche≠NIL) x=x.sa-gauche;
       retourner x;
       }
    }
```

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Maximum :

- la fonction maximum(A) retourne le pointeur NIL si l'arbre dont la racine est pointée par A est vide
- sinon, elle retourne le pointeur sur un nœud contenant la clé maximale de l'arbre

```
maximum(A){
    si (A=NIL) retourner A
    sinon{
       x=A;
       tant que (x.sa-droit≠NIL) x=x.sa-droit;
       retourner x;
       }
    }
```

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

Insertion d'un élément :

- la fonction insertion(A,k) insère un nœud de clé k dans l'arbre dont la racine est pointée par A

```
Insertion(A,k){
    créer nœud z;z.clef=k;z.sa-gauche=NIL;z.sa-droit=NIL;
    x=A;père_de_x=NIL;
    tant que (x≠NIL){
       père_de_x=x;
       si (k<x.clef) x=x.sa-gauche
       sinon x=x.sa-droit
       }
    si (père_de_x=NIL) A=z
    sinon          si (k<père_de_x.clef) père_de_x.sa-gauche=z
                   sinon père_de_x.sa-droit=z
    }
```

CHAPITRE V
Structures de données avancées

A. Arbres binaires de recherche

- Complexité des différentes fonctions :
  - rechercher(A,k), minimum(A), maximum(A), Insertion(A,k)
    - Chacune des fonctions nécessite, dans le pire des cas, une et une seule descente complète (sans retour arrière) dans l'arbre de recherche
    - En d'autres termes, les nœuds visités par chacune des fonctions sont tous sur une unique branche allant de la racine vers une feuille
    - La complexité de chacune des fonctions est donc en O(h), h étant la hauteur de l'arbre de recherche
    - Comme la hauteur est bornée par n-1, la complexité est en O(n)

CHAPITRE VI
NP-complétude

- Problème
  - Description
  - Question
- Instance d'un problème
  - Une instance d'un problème est une spécialisation du problème obtenue en donnant une spécification exacte des données

CHAPITRE VI
NP-complétude

Exemple 1 :

Le problème SAT

- Description : une conjonction de m clauses construites à partir de n propositions atomiques
- Question : la conjonction est-elle satisfiable ?

Instance du problème SAT

- La conjonction $p \vee q \wedge p \vee \neg r \wedge \neg p \vee \neg q \vee \neg r$

CHAPITRE VI
NP-complétude

Exemple 1 :

Le problème SAT

- La réponse à une instance du problème SAT est unique :
  - La réponse est OUI si l'instance est satisfiable
  - La réponse est NON sinon

CHAPITRE VI
NP-complétude

Exemple 2 :

Le problème PLUS-COURT-CHEMIN

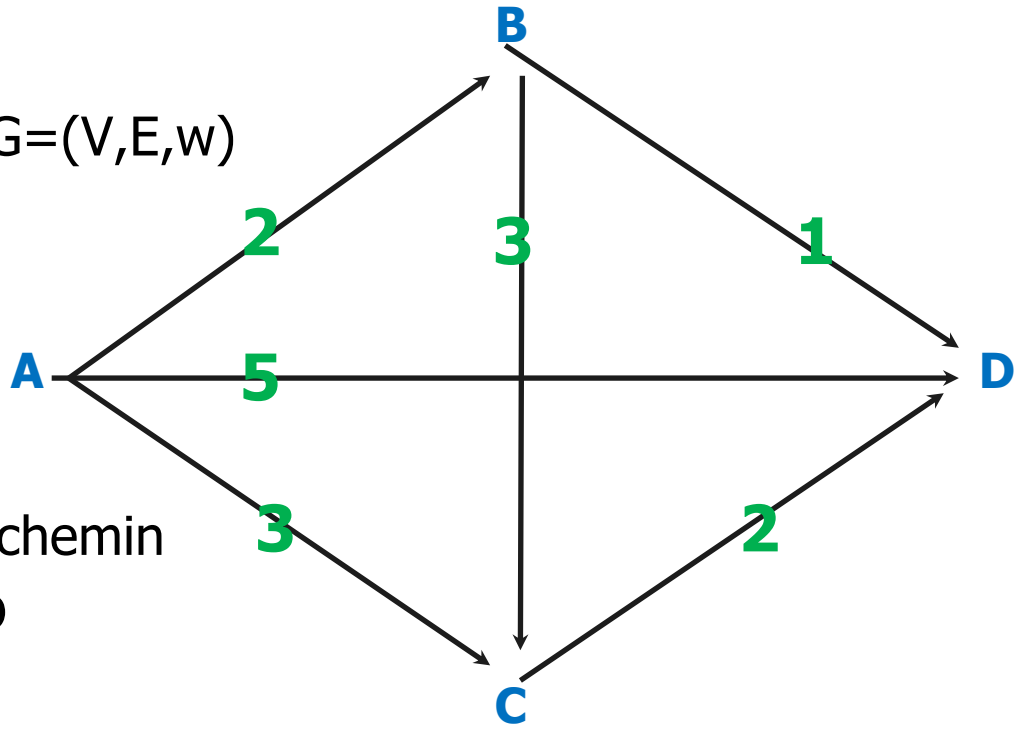
- Description : un graphe orienté pondéré G=(V,E,w) et deux sommets u et v du graphe
- Question : trouver un plus court chemin entre les sommets u et v

CHAPITRE VI
NP-complétude

Exemple 2 :

Instance du problème PLUS-COURT-CHEMIN

- Description
  - Le graphe étiqueté G=(V,E,w)
  - Les sommets A et D



- Question
  - Trouver un plus court chemin

entre les sommets A et D

CHAPITRE VI
NP-complétude

Exemple 2 :

Le problème PLUS-COURT-CHEMIN

- La réponse à une instance du problème PLUS-COURT-CHEMIN est une séquence de sommets du graphe :
  - Si la séquence est vide, il n'existe pas de chemin entre les deux sommets
- La réponse n'est donc pas forcément unique :
  - Étant donnés deux sommets A et B d'un graphe orienté pondéré, il peut exister plusieurs plus courts chemins pour aller de A vers B

CHAPITRE VI
NP-complétude

Problème de décision

- La théorie de la NP-complétude se restreint aux problèmes dits de décision
- Un problème de décision est un problème dont chaque instance admet une unique réponse, OUI ou NON

CHAPITRE VI
NP-complétude

- Problèmes d'optimisation
  - Un problème d'optimisation n'est généralement pas un problème de décision
    - Un problème d'optimisation admet généralement plus d'une solution
  - Mais on peut toujours associer un problème de décision à un problème d'optimisation
- Exemple :
  - Le problème d'optimisation PLUS-COURT-CHEMIN
  - Le problème de décision associé est le problème CHEMIN

CHAPITRE VI
NP-complétude

Le problème de décision CHEMIN

- Description : un graphe orienté pondéré G=(V,E,w), deux sommets u et v du graphe et un entier positif k
- Question : existe-t-il un chemin entre les sommets u et v de longueur inférieure ou égale à k ?

CHAPITRE VI
NP-complétude

Deux façons de définir un problème :

- Définition abstraite (mathématique)
- Définition concrète (informatique→implémentation)

Si un problème est donné par sa définition abstraite, on parle de problème abstrait. S'il est donné par sa définition concrète, on parle de problème concret

CHAPITRE VI
NP-complétude

Définition abstraite :

Soit P un problème, I(P) l'ensemble de ses instances et S(P) l'ensemble des solutions de ses instances

- La définition abstraite de P voit P comme une relation binaire associant à tout élément de I(P) un sous-ensemble de S(P) :

  P={(i,s) : (i instance de P)
  (s⊆S(P))
  (tout élément de s est solution de i)}

- La définition abstraite est donc une définition en compréhension

CHAPITRE VI
NP-complétude

Définition concrète :

Utilisation d'un codage

- Un codage permet une représentation interne d'une instance directement exploitable par un algorithme

CHAPITRE VI
NP-complétude

Codage

- Codage d'un ensemble S d'objets abstraits
  - Application e de S dans l'ensemble des chaînes binaires
  - Ou, de façon générale, application de S dans l'ensemble des chaînes d'un alphabet fini (alphabet {0,…,9,A,…,F} des chiffres du système de numération hexadécimal -de base 16-)
- Exemple :
  - Codage des entiers sous forme binaire (base 2)
  - Codage des entiers en hexadécimal
- Un algorithme conçu pour résoudre un problème de décision P prend en entrée un codage d'une instance de P

CHAPITRE VI
NP-complétude

Codage (exemple) : Codage d'une instance I du problème SAT donnée par une conjonction $C=c_1 \wedge \ldots \wedge c_i \wedge \ldots \wedge c_m$ (m clauses ; n propositions atomiques) :

- Codage évolué (langage évolué : haut niveau, proche du langage naturel)
  - Deux entiers m et n
  - Une m*n-matrice M_I d'entiers pour coder la conjonction C de clauses : pour tous i,j, $M_I[i,j]=0$ si la jème proposition atomique occurre négativement dans la ième clause ; 1 si elle y occurre positivement ; 2 si elle n'y occurre pas
  - Le codage peut ainsi être vu comme un mot (chaîne) sur un alphabet fini {0,…,L} d'entiers, de longueur (2+m*n) : les 2 premiers entiers codent les données m et n ; les n suivants codent la première clause ; …; les tout derniers n codent la toute dernière clause

CHAPITRE VI
NP-complétude

Codage (exemple)

Codage évolué de l'instance I du problème SAT donnée par la conjonction $C=p\vee q \wedge p\vee\neg r \wedge \neg p\vee\neg q\vee\neg r$ de SAT (m=n=3) :

- m=3 ; n=3
- M_I=

	p	q	r
c1	1	1	2
C2	1	2	0
c3	0	0	0

- Codage=

3	3	1	1	2	1	2	0	0	0	0

CHAPITRE VI
NP-complétude

Codage (exemple) : Codage d'une instance I du problème SAT donnée par une conjonction $C=c_1 \wedge \ldots \wedge c_i \wedge \ldots \wedge c_m$ (m clauses ; n propositions atomiques) :

- Codage binaire (bas niveau : langage machine)
  - On suppose qu'un entier est représenté sur un octet (8 bits)
  - Deux octets pour coder m et n
  - Une m*n-matrice M_I d'octets pour coder la conjonction C de clauses : pour tous i,j, $M_I[i,j]$=00000000 si la jème proposition atomique occurre négativement dans la ième clause ; 00000001 si elle y occurre positivement ; 00000010 si elle n'y occurre pas
  - Le codage peut ainsi être vu comme un mot (chaîne) sur l'alphabet binaire {0,1}, de longueur (2+m*n)*8 : les 2*8 premiers bits codent les données m et n ; les n*8 suivants codent la première clause ; …; les tout derniers n*8 codent la toute dernière clause

CHAPITRE VI
NP-complétude

Codage (exemple)

Codage binaire de l'instance I du problème SAT donnée par la conjonction C= $p \vee q \wedge p \vee \neg r \wedge \neg p \vee \neg q \vee \neg r$ (m=n=3) :

- m=00000011 ; n=00000011
- M_I=

	p	q	r
c1	00000001	00000001	00000010
C2	00000001	00000010	00000000
c3	00000000	00000000	00000000

- Codage=00000011000000110000000100000001000000100000000100000010000000000000000000000000

CHAPITRE VI
NP-complétude

Problème concret

- Problème dont les instances sont des chaînes d'un alphabet binaire
- Un algorithme résout un problème concret P en O(g(n)) si le nombre f(n) d'opérations élémentaires du pire cas de l'algorithme vérifie f(n)=O(g(n)) :

$\exists n_0, \exists c \geq 0$: pour toute instance I de P de taille $n \geq n_0$, $f(n) \leq c*g(n)$

CHAPITRE VI
NP-complétude

- La classe de complexité P
  - La classe de complexité P est l'ensemble des problèmes concrets de décision qui sont résolubles en un temps polynomial
  - Un problème concret est résoluble en un temps polynomial s'il existe un algorithme permettant de le résoudre en $O(n^k)$, pour une certaine constante k

CHAPITRE VI
NP-complétude

La classe de complexité NP

- Certificat :
  - Considérons l'instance suivante de SAT :

    $p \vee q \wedge p \vee \neg r \wedge \neg p \vee \neg q \vee \neg r$
  - Si on considère une instanciation (b_1,b_2,b_3) du triplet (p,q,r) de propositions atomiques, on peut facilement vérifier si oui ou non elle satisfait l'instance. Et si elle la satisfait, l'instanciation peut être vue comme un certificat (preuve) que l'instance du problème SAT est satisfiable
  - Un certificat peut donc être vu comme une instanciation des variables de l'instance
  - Les certificats de l'instance ci-dessus de SAT sont tous les triplets du produit cartésien $\{VRAI,FAUX\}^3$

CHAPITRE VI
NP-complétude

La classe de complexité NP

- Algorithme de validation :
  - Soit Q un problème concret de décision. Un algorithme de validation pour Q est un algorithme de décision A à deux arguments, une instance du problème Q, et un certificat y. L'algorithme A valide l'instance x si et seulement si il existe un certificat y tel que A(x,y)=vrai :

    A valide une instance x de Q si et seulement si la solution de x est OUI

CHAPITRE VI
NP-complétude

La classe de complexité NP

- La classe de complexité NP est l'ensemble des problèmes concrets de décision pour lesquels il existe un algorithme polynomial de validation
- NP : Non-déterministe Polynomial

CHAPITRE VI
NP-complétude

La classe de complexité NP (exemple 1 : le problème SAT)

- Description : une conjonction de m clauses construites à partir de n propositions atomiques
- Question : la conjonction est-elle satisfiable ?

Soit I une instance de SAT donnée par la conjonction $C=c_1 \wedge ... \wedge c_i \wedge ... \wedge c_m$

Certificat :

- un tableau t de taille n de booléens
- t[i] est la valeur booléenne que le certificat affecte à la proposition atomique p_i

CHAPITRE VI
NP-complétude

La classe de complexité NP (exemple 1 : le problème SAT)

```
Boolean validation_sat(I,t){
i=1                                  // i parcourt les clauses (lignes de la matrice M_I)
Tant que (i≤m) faire
        clause_satisfaite=FAUX ; j=1   // j parcourt les propositions atomiques
        tant que (non clause_satisfaite) et (j≤n) faire
            si (t[j]=VRAI et M_I[i,j]=1) ou (t[j]=FAUX et M_I[i,j]=0) alors
                        clause_satisfaite=VRAI sinon j=j+1 finsi
        fait
        si clause_satisfaite alors i=i+1 sinon retourner FAUX finsi
Fait
Retourner VRAI
}
```

CHAPITRE VI
NP-complétude

La classe de complexité NP (exemple 2 : le problème CLIQUE)

- Description : un graphe non orienté G=(V,E) et k∈{1,…,|V|}
- Question : G admet-il une clique de taille k ?

Soit I une instance de CLIQUE donnée par le graphe G=(V,E) avec |V|=n et un entier k∈{1,…,n}

Codage : deux entiers n et k, et une n*n-matrice booléenne M_I (matrice d'adjacence de G) : $M_I[i,j]=1$ ssi (Xi,Xj) arête de G

Certificat :

- un tableau t de taille k d'éléments de {1,…,n} tous différents (indices de sommets de V)

CHAPITRE VI
NP-complétude

La classe de complexité NP (exemple 2 : le problème CLIQUE)

```
Boolean validation_clique(I,t){
i=1
Tant que (i≤k) faire
        j=i+1
        tant que (j≤k) faire
            si (M_I[t[i],t[j]]=0) alors retourner FAUX sinon j=j+1 finsi
        fait
        i=i+1
Fait
Retourner VRAI
}
```

CHAPITRE VI
NP-complétude

Fonction calculable en temps polynomial

- Une fonction $f:\{0,1\}^* \rightarrow \{0,1\}^*$ est calculable en temps polynomial s'il existe un algorithme polynomial la calculant, c'est-à-dire produisant f(x), pour tout $x \epsilon \{0,1\}^*$.

CHAPITRE VI
NP-complétude

Réductibilité

Soient Q_1 et Q_2 deux problèmes concrets de décision. Q_1 est réductible en temps polynomial à Q_2 ($Q_1 \leq_P Q_2$) s'il existe une fonction calculable en temps polynomial $f:\{0,1\}^* \rightarrow \{0,1\}^*$ telle que pour toute instance $x \in \{0,1\}^*$ de Q_1, f(x) est instance de Q_2 ; et :

La solution de l'instance x de Q_1 est OUI

si et seulement si

la solution de l'instance f(x) de Q_2 est OUI

CHAPITRE VI
NP-complétude

Problème NP-complet

Un problème concret de décision Q est NP-complet si :

1. $Q \in NP$
2. $\forall Q' \in NP,\ Q' \leq_P Q$

- *On note NPC la classe des problèmes NP-complets*
- *Un problème concret de décision qui vérifie la propriété 2 mais pas nécessairement la propriété 1 est dit* ***NP-difficile***

CHAPITRE VI
NP-complétude

Exemples de problèmes NP-complets

- SAT
- le père des problèmes NP-complets (le tout premier à avoir été montré NP-complet par Stephen COOK en 1971)
- 3-SAT : Satisfiabilité d'une conjonction de clauses dont chaque clause est composée d'exactement trois littéraux
- CYCLE HAMILTONIEN : Existence dans un graphe d'un cycle hamiltonien
- VOYAGEUR DE COMMERCE : Existence dans un graphe pondéré d'un cycle hamiltonien de coût minimal
- CLIQUE : Existence dans un graphe d'une clique (sous-graphe complet) de taille k
- 3-COLORIAGE D'UN GRAPHE : peut-on colorier les sommets d'un graphe avec trois couleurs de telle sorte que deux sommets adjacents n'aient pas la même couleur ?
- PARTITION : Peut-on partitionner un ensemble d'entiers en deux sous-ensembles de même somme ?

CHAPITRE VI
NP-complétude

Preuve de NP-complétude

- Comment démontrer qu'un problème est NP-complet ?

Théorème

- Si Q_1 est un problème tel que $Q_2 \leq_P Q_1$ pour un certain problème $Q_2 \in NPC$, alors Q_1 est NP-difficile
- Si, de plus, $Q_1 \in NP$, alors $Q_1 \in NPC$.

CHAPITRE VI
NP-complétude

Méthode pour montrer la NP-complétude d'un problème Q_1

- Prouver que $Q_1 \in NP$
- Choisir un problème NP-complet Q_2
- Décrire un algorithme polynomial capable de calculer une fonction f faisant correspondre toute instance de Q_2 à une instance de Q_1
- Démontrer que la fonction f satisfait la propriété :

  la réponse $Q_2(x)$ à une instance x de Q_2 est oui

  si et seulement si

  la réponse $Q_1(f(x))$ à l'instance $f(x)$ de Q_1 est oui
- Démontrer que l'algorithme calculant f s'exécute en temps polynomial

CHAPITRE VI
NP-complétude

- Montrer que le problème de 3-coloriage d'un graphe est NP-complet ?