Introduction à la Théorie de la Complexité

Introduction générale

Qu'est ce que l'optimisation combinatoire ?

- Problèmes d'existence
- Problèmes de dénombrement (analyse combinatoire)
- Problèmes combinatoires

Introduction générale

- Les problèmes traités au début de l'informatique étaient de taille relativement réduite, mais on s'est vite aperçu qu'il y avait des limites à la puissance des ordinateurs qu'il fallait cerner.
- La théorie de la complexité étudie deux aspects:
  - la complexité des algorithmes : évaluer en fonction de la taille des données, le nombre des opérations des algorithmes.
  - la complexité des problèmes et leur classification.

Notions de base

- Un problème est une « question » qui cherche une réponse et qui comporte un ensemble de paramètres.
  - description des paramètres;
  - propriétés que la réponse ou la *solution,* doit satisfaire;
  - *Une instance du problème est obtenue quand on fixe les paramètres à des valeurs données.*
- Algorithme : un ensemble d'instructions décrivant comment réaliser une tâche ou résoudre un problème.
- Programme : la réalisation informatique d'un algorithme

Complexité des algorithmes

- Il peut exister plusieurs algorithmes pour le même problème… comment choisir ?
- Le temps d'exécution du programme dépend :
  - des données du problème, de leur taille,
  - de la qualité du code…,
  - de la puissance de l'ordinateur,
  - de l'efficacité de l'algorithme,
  - etc.
- On procède alors par analyse de l'algorithme.
  - On cherche une grandeur *n* pour quantifier les données d'entrée.
  - On calcule les performances seulement en fonction de *n*.
  - On évalue le nombre d'opérations élémentaires, *(on appel opération élémentaire toute instruction d'un langage informatique).*

Évaluation des algorithmes (I)

- Chaque algorithme est composé d'une phase de lecture de données et d'une phase de calcul.
  - Le paramètre de complexité d'un algorithme est la taille d'une donnée (nous utilisons le codage binaire)

  <u>Définition de mesure de calcul de complexité :</u>

  Soit un entier $n>0$ et $T(n)$ le temps d'exécution d'un algorithme en fonction de la taille n des données, on dira que $T(n)$ est en $O(f(n))$ si et seulement si $\exists n_0$ et une constante c tels que :

  $\forall n \geq n_0$, on a que $T(n) \leq c\, f(n)$

  Si f(n) est un polynôme alors l'algorithme est de complexité polynomiale.

- Étudier la complexité au pire cas.
- Étudier la complexité en moyenne : $tm(n)$ est l'espérance de l'ensemble des temps d'exécution sur l'ensemble des données muni d'une distribution des probabilités des temps d'exécution.

Évaluation des algorithmes (II) exemple

Algorithme dichotomique (rechercher un élément dans un tableau trié de taille n) : $O(\log(n))$.

Calcul de l'élément minimum dans un ensemble de taille n : $O(n)$.

Tri de n éléments : $O(n\log(n))$

Calcul du plus court chemin : $O(m)$, $O(n^2)$, $O(nm)$

Enumérer les sous-ensembles d'un ensemble de n éléments (2^n);

Importance des algorithmes polynomiaux (I)

f(n)	n=10	n=20	n=30	n=40	n=50
n	0.00001 sec	0.00002 sec	0.00003 sec	0.0000 4 sec	0.0000 5 sec
n^2	0.0001 sec	0.0004 sec	0.0009 sec	0.0016 sec	0.0025 sec
n^3	0.001 sec	0.008 sec	0.027 sec	0.064 sec	0.125 sec
2^n	0.001 sec	1 sec	17.9 min	12.7 jours	35.7 années
3^n	0.059 sec	58 min	6.5 années	3.855 siècles	$2*10^8$ siècles

Tableau rapportant les valeurs de f(n) pour des fonctions de complexité usuelles, si une opération élémentaire coûte une microseconde.

Importance des algorithmes polynomiaux (II)

f(n)	Avec les ordinateurs d'aujourd'hui	100 fois plus rapide	1000 fois plus rapide
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N3	10N3
2^n	N4	N4 +6.64	N4 + 9.97
3^n	N5	N5 + 4.19	N5 + 6.29

Tableau rapportant la taille des problèmes résolus avec une heure de calcul.

Théorie de la complexité complexité des problèmes

- Au cœur de la théorie de la complexité et de son formalisme se trouve les problèmes de décision.
- Un problème de décision est une question mathématiquement définie portant sur des paramètres donnés sous forme manipulable informatiquement, et demandant une réponse par oui ou par non.

Le problème SAT

- Le problème SAT est un problème de décision d'une grande importance en théorie de la complexité.
- Définition. Etant donné une fonction booléenne sous une forme conjonctive, existe-t-il une affectation des variables rendant vraie cette fonction ?

De façon plus formelle :

- Le problème SAT s'énonce de la manière suivante :
  - Données : un ensemble V de variables booléennes et une famille C de clauses sur V, (une clause est un ensemble de variables booléennes sous la forme directe ou complémentée; elle est vraie des qu'une de ses variables est vraie, *donc la somme*).
  - Question : Existe-t-il une façon d'affecter les valeurs « vrai » ou « faux » aux variables afin de rendre vraie toutes les clauses ?

- Problème 3SAT : chaque clause ne contient que trois variables.

Problème du voyageur de commerce (TSP)

- Étant donné *n* points (des « villes ») et les distances séparant chaque point, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point (et revienne au point de départ).
- On associe le problème de décision suivant.
  - Donnée : un graphe complet $G = (V,E,\omega)$ avec V un ensemble de sommets, E un ensemble d'arcs et ω une fonction de coût sur les arcs, et un entier B.
  - Question : Existe-t-il un circuit hamiltonien (c'est-à-dire passant une et une seule fois par chaque sommet) dont le coût (la somme du coût des arcs qui le composent) est inférieur ou égal à B ?

Problèmes de partition

- Le problème de la partition
  - Donnée : un ensemble $A=\{a_i \mid i \in I\}$ de n nombres entiers.
  - Question : Existe-t-il une partition de A en deux sous-ensembles A1 et A2 de même poids ?

Exemples de problèmes équivalents

- Problème de l'ensemble indépendant (independent set).
  - Donnée : un graphe $G=(V,E)$ avec V un ensemble de sommets, E un ensemble d'arcs et un entier (positif) $B \leq |V|$.
  - Question : Existe-t-il un ensemble de sommets $V' \subseteq V$ de taille supérieure ou égale à B tel que pour tout arc $(i,j) \in E$, $i \notin V'$ ou $j \notin V'$?

- Problème de clique maximum.
  - Donnée : un graphe $G = (V,E)$ avec V un ensemble de sommets, E un ensemble d'arcs, et un entier B.
  - Question : Existe-t-il un sous-ensemble $V' \subseteq V$ tel que le sous-graphe correspondant est complet et de cardinal supérieur ou égal à B ?

Théorie de la complexité
complexité des problèmes : notions de base

- Les problèmes de décision; pourquoi ce choix ?
- Pour introduire un formalisme et faciliter la comparaison des problèmes selon leur difficulté.
- *La théorie de la complexité est basée sur la notion de la machine de Turing…*

Complexité des problèmes

- On distingue des classes de complexité :
  - Classe P : Un problème de décision est dans P si il peut être résolu par un algorithme *déterministe* en un temps *polynomial* par rapport à la taille de l'instance. On qualifie alors le problème de « polynomial ».
  - Classe NP : C'est la classe des problèmes de décision pour lesquels la réponse *oui* peut être décidée par un algorithme *non-déterministe* en un temps polynomial par rapport à la taille de l'instance.
    - **La vérification est polynomiale.**

Théorie de la complexité
les problèmes NP-complet

Article de Stephen Cook, «The complexity of Theorem Proving Procedures», 1971.

- définit la réduction polynomiale
- définit les problèmes de décision et la classe NP
- démontre qu'il y a un problème (SAT) qui est au moins aussi difficile que tous les autres → NP-complet.

Théorie de la complexité
complexité des problèmes

- La réduction polynomiale permet de comparer selon leur difficulté les problèmes NP entre eux.
- *Définition : On dit que P_1 se réduit polynomialement à P_2 ($P_1 \propto P_2$) si P_1 est polynomial ou s'il existe un algorithme polynomial A construisant à partir d'une donnée d_1 de P_1 une donnée d_2 de P_2 tel que d_1 à la réponse oui si et seulement si $d_2=A(d_1)$ a la réponse oui.*
- Problème NP-Complet : un problème est NP-Complet si il est dans NP, et si n'importe quel problème NP peut se réduire polynomialement à ce problème.
- La réduction polynomiale est une relation de pré-ordre sur NP.
- Théorème de Cook : SAT est NP-Complet.

Complexité des problèmes, méthodes de démonstration

- Comment démontrer qu'un problème Π est NP-complet
  - 1) démontrer que $\Pi \in$ NP
  - 2) démontrer qu'il existe P'$\in$NP-complet tel que P'$\propto \Pi$.
- Exemples de problèmes NP-complets :
  - le problème du voyageur de commerce,
  - les problèmes de partition,
  - le problème de la clique

Théorie de la complexité exercices (III)

Classer les problèmes énoncés ci-dessous :

<u>Problème de plus court chemin entre deux sommets</u> Donnée : Graphe G(V,E), l(e)$\in$Z et a,b dans V, un entier positif B. Question : Existe-t-il un chemin élémentaire de a à b de longueur inférieure ou égale à B?	<u>Problème du plus long chemin entre deux sommets</u> Donnée : Graphe G(V,E), l(e)$\in$Z et a,b dans V, un entier positif B. Question : Existe-t-il un chemin élémentaire de a à b de longueur supérieure ou égale à B?
<u>Problème de couverture des arcs</u> Donnée : Graphe G(V,E), un entier positif B. Question : Existe-t-il un sous-ensemble E' $\subseteq$E tel que $\lvert E'\rvert \leq$ B, et pour tout sommet v $\in$ V, il existe e$\in$E' tel que v$\in$e?	<u>Problème de couverture des sommets</u> Donnée : Graphe G(V,E), un entier positif B. Question : Existe-t-il un sous-ensemble V'$\subseteq$V tel que $\lvert V'\rvert \leq$ B, et pour tout arc (i,j)$\in$E, alors i$\in$V' ou j$\in$V'?

Problème NP-complet : conjecture

- Conjecture fondamentale de la théorie de la complexité : P≠NP.

Le prix de 1000000 USD sera attribué à celui qui démontre que soit P=NP soit P≠NP.

Comment résoudre les problèmes NP-complets

Méthodes exactes:

- programmation dynamique (algorithme pseudopolynomial)
  - Applicable à certains problèmes NP-complets dit « faibles »
- Méthodes arborescentes
  - Enumération implicite de l'ensemble des solutions.

Méthodes heuristiques

- Glouton, recuit simulé, liste tabou, algorithmes génétique…

Algorithmes pseudo-polynomiaux (I)

- Une méthode de programmation dynamique
  - Idée : décomposer la résolution du problème initial en une suite de problèmes plus simples, la résolution du n-ème se déduisant de celle du (n-1)-ème par une équation de récurrence.
  - Problème PDSE (poids d'un sous-ensemble)
    - Donnée : Un ensemble fini A composé de n éléments $a_i \in Z+$ et un entier positif K.
    - Question : Existe-t-il un sous-ensemble de poids K ?

Algorithmes pseudo-polynomiaux (II)

Algorithme PDSE
début
pour k=0 à $\sum_{ai \in A} a_i$
pour j=0 à n faire :
POIDS(k,j):=faux;
fin pour;
fin pour;
poser POIDS(0,0) := vrai;
pour j=1 à n
pour k=0 à $\sum_{ai \in A} a_i$ faire :
Si ($k \geq a_j$ et POIDS(k-a_j, j-1)=vrai) ou POIDS(k, j-1)=vrai
alors POIDS(k, j)=vrai;
fin pour;
fin pour;
fin.

Proposition : L'algorithme PDSE a une complexité $O(n\sum_{ai \in A} a_j)$ et affecte à POIDS(K, n) la valeur vrai s'il existe un sous-ensemble de n nombres $a_1, a_2, .., a_n$ de poids K.

Algorithmes pseudo-polynomiaux (III)

- Codage d'un mot : codage binaire, a → $\log_2(a)$
  - exemple : la taille du mot 2^{31} sera $\log_2(2^{31}) + 1 = 32$, donc il faut 32 bits pour le coder;
- Par définition, un algorithme est de complexité pseudo-polynomial s'il est polynomial pour un codage dans lequel les nombres apparaissant dans les données sont codés en base 1.
  - exemple : la taille du mot 2^{31} sera 2^{31}, donc il faut 2^{31} symboles pour le coder.

Problème NP-complet au sens forts

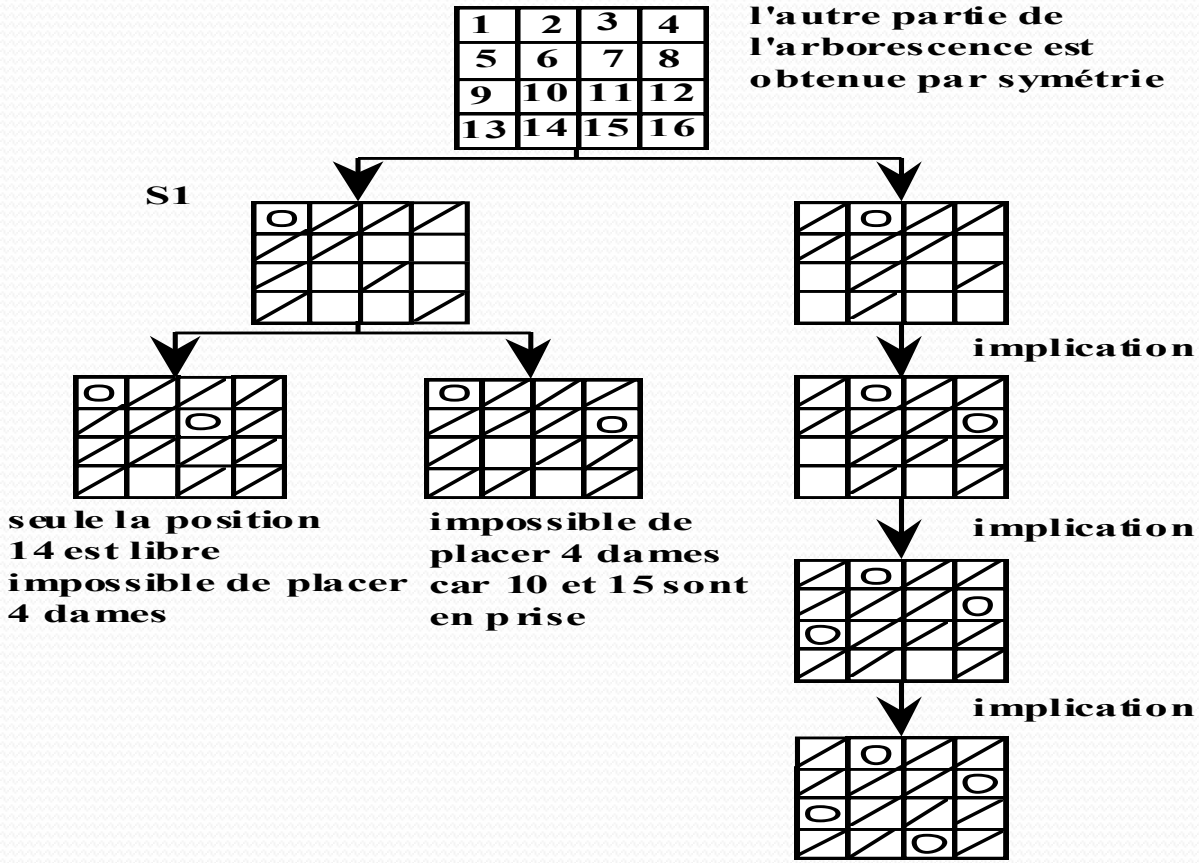
- Définition : Un problème est dit NP-complet au sens fort, s'il est NP-complet et si l'existence d'un algorithme pseudo-polynomial pour le résoudre entraîne celle d'un algorithme polynomial.
  - Exemples : voyageur de commerce, clique, sat.
  - Que peut-on dire de la partition?

Méthodes arborescentes

Méthodes arborescentes:

famille d'algorithmes qui énumèrent de manière implicite l'ensemble des solutions d'un problème.

L’arborescence des dames de Gauss



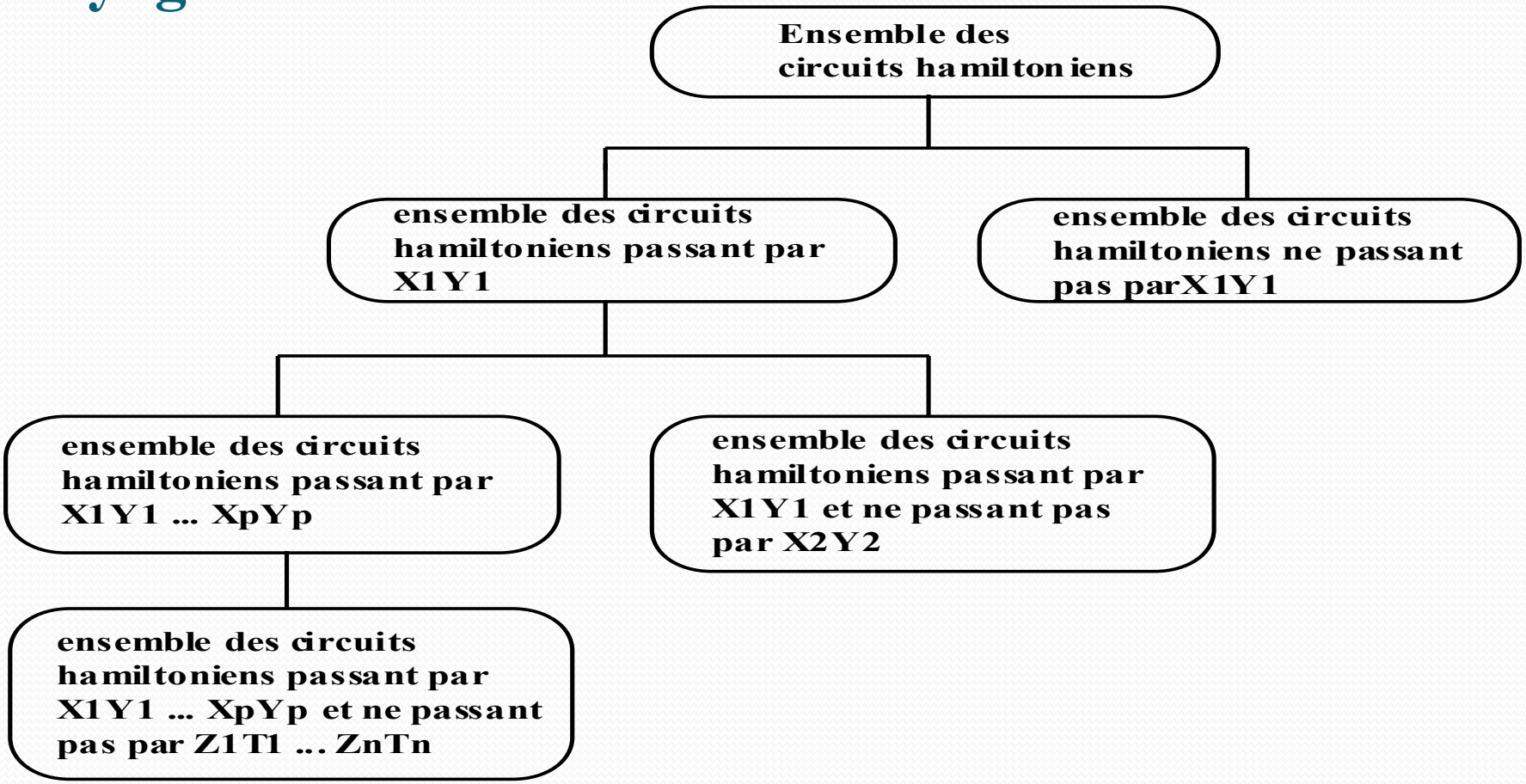
Méthodes exactes de résolution

- Énumération complète des solutions : *Branch and Bound method (B&B)*, méthode de séparation et évaluation progressive.

  - *Principe : On construit un arbre de recherche dont le problème initial (p.ex. problème de minimisation) est la racine.*
  - *On divise le problème en sous-problèmes : l'optimum peut appartenir à l'un quelconque de ces sous-problèmes.*
    - *tout sous-problème infaisable sera éliminé;*
    - *si possible on calcul la solution du sous-problème.*
    - *sinon on calcule une borne inférieure, si elle est supérieure de la meilleure solution déjà obtenue, on éliminé le sous-problème.*
    - *Dans le cas restant, on sub-divise à nouveau le domaine.*

Noter que la meilleure solution obtenue constitue aussi une borne supérieure.

Méthode de LITTLE pour le problème du voyageur de commerce



Arborescence dichotomique

Algorithme de LITTLE :

(1) Initialisation :

Faire apparaître des **zéros** (au moins un par ligne et un par colonne)

{L'évaluation par défaut de la racine de l'arborescence est la somme des chiffres soustraits}.

Poser **f0 =** ∞ {f0 sera la valeur de la meilleure solution connue}.

(2) Boucle principale :

Tant que le sommet S de l'arborescence de plus petite évaluation par défaut f(S) est tel que

f(S) < f0 faire {largeur d'abord}

- évaluer le regret de chacun des zéros de la matrice associée à **S** et retenir **(i, j)** correspondant à un zéro de regret maximal.
- introduire les sommets **(S + (i, j))** et **(S +** ¬**(i, j))** en interdisant le circuit parasite dans **(S + (i, j)),** en faisant apparaître des zéros dans les deux matrices associées, et en valuant ces sommets par leur évaluation par défaut.
- **Si** le sommet **(S + (i, j))** correspond à un circuit hamiltonien **alors** écrire le circuit, et calculer sa valeur f1.
- poser **f0 = min (f0, f1)**

Fin tant que.

Un exemple

Intialisation

	A	B	C	D	E	F
A	∞	6	7	3	1	3
B	7	∞	8	2	9	7
C	5	10	∞	10	1	7
D	8	6	5	∞	5	1
E	7	7	6	7	∞	4
F	9	8	8	5	3	∞

	A	B	C	D	E	F
A	∞	2	4	2	0	2
B	2	∞	4	0	7	5
C	1	6	∞	9	0	6
D	4	2	2	∞	4	0
E	0	0	0	3	∞	0
F	3	2	3	2	0	∞

Choix de l'arc avec le plus grand regret

	A	B	C	D	E	F
A	∞				0^2	
B		∞		0^{2+2}		
C			∞		0^1	
D				∞		0^2
E	0^1	0^2	0^2		∞	00
F					0^2	∞

Un exemple (suite)

	A	B	C	D	E	F
A	∞				0^2	
B		∞		0^{2+2}		
C			∞		0^1	
D				∞		0^2
E	0^1	0^2	0^2		∞	00
F					0^2	∞

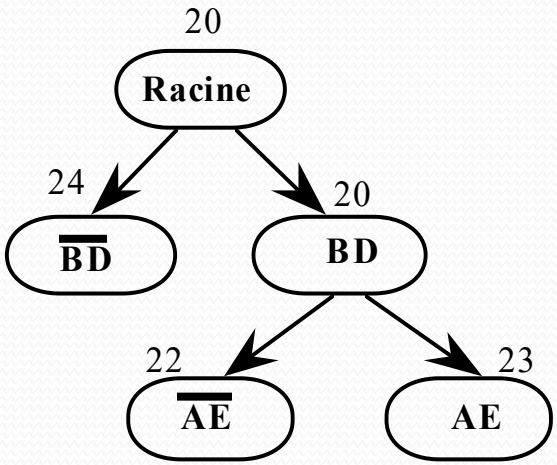
Modification de la matrice (DB=∝)

	A	B	C	D	E	F
A	∞	2	4	~~2~~	0	2
B	~~2~~	~~∞~~	4	~~0~~	~~7~~	~~5~~
C	1	6	∞	~~9~~	0	6
D	4	∞	2	~~∞~~	4	0
E	0	0	0	~~3~~	∞	0
F	3	2	3	~~2~~	0	∞

	A	B	C	E	F
A				0^2	
C				0^1	
D					0^2
E	0^2	0^2	0^2		0^2
F				0^2	

Un exemple (suite)

Arborescence courante



	A	B	C	F	
C	1	6	∞	6	1
D	4	∞	2	0	0
E	∞	0	0	0	0
F	3	2	3	∞	2

Matrice modifiée (EA= ∝)

Un exemple (suite)

	A	B	C	E	F
A	∞	0	2	∞	0
C	1	6	∞	0	6
D	4	∞	2	4	0
E	0	0	0	∞	0
F	3	2	3	0	∞

	A	B	C	E	F
A		0^0			0^0
C				0^1	
D					0^2
E	0^1	0^0	0^2		0^0
F				0^2	

Matrices associées au sommet ¬AE

Un exemple (suite)

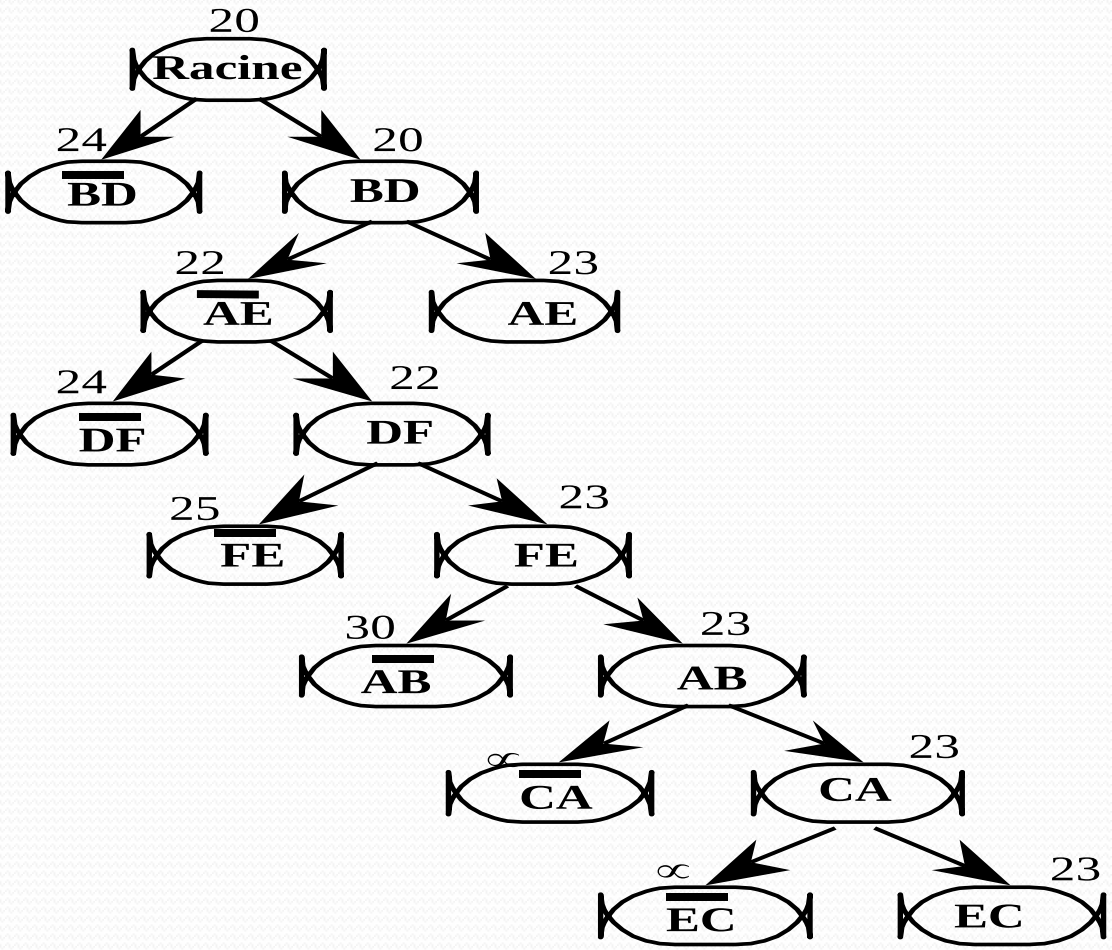
	A	B	C	E
A	∞	0	2	∞
C	1	6	∞	0
E	0	0	0	∞
F	3	∞	3	0

	A	B	C	E
A		0^2		
C				0^1
E	0^1	0^0	0^2	
F				0^3

et ainsi de suite ...

Matrices associées au sommet DF.

Un exemple: arborescence final



Méthodes arborescentes

Suite : applications aux programmes linéaires…