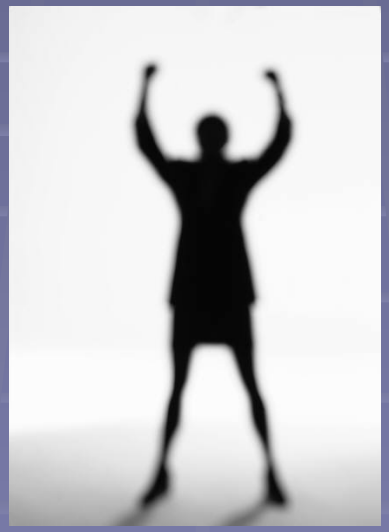
Méthodes exactes pour la résolution de problèmes NP

Jean-Philippe Préaux



- Pour résoudre un problème NP-difficile de façon ‘relativement efficace’, soit :
  - on applique une heuristique (qui ne donne pas toujours une solution optimale, ou ne s’arrête pas toujours, …).
  - On énumère une partie de l’espace des solutions (fini mais gigantesque), de façon intelligente, de sorte que dans la plupart des cas on arrive assez vite à la solution. Cela donne un algorithme de complexité théorique exponentielle, mais qui dans la pratique sont en ‘moyenne’ plus rapide :
  - Nous verrons deux telles méthodes :
    - Méthode par séparation et évaluation
    - Programmation dynamique

Méthode par séparation et évaluation (branch & bound).

- Consiste à parcourir l'espace des solutions seulement en partie, en s'assurant que la partie non explorée ne contienne aucune solution de meilleur coût.
- De complexité exponentielle en théorie. En pratique, et en moyenne la complexité devient meilleure, souvent $O(1.5^n)$ (faiblement exponentiel).
- On verra un exemple important pour le PVC : l'algorithme de Little.

- L'espace des solutions S est construit implicitement par une arborescence. Chaque sommet est une partie de S. Les feuilles sont les singletons(=solutions) ou vides(=impossibles).
- Le noeud initial correspond à S en entier.
- Si un sous-ensemble S' correspond à un noeud x, ses (2 ou plus) successeurs correspondent à une partition de S' : S_1 et S_2 avec $S_1 \cup S_2 = S'$ et $S_1 \cap S_2 = \emptyset$.
- Chaque sommet est muni d'une évaluation. Obtenu par une heuristique, pour un problème de min (max) c'est un minorant (majorant) du coût sur le sous-espace correspondant au sommet.
- Quand on a construit une solution dont le coût est inférieur (sup) à l'évaluation des autres sommets, on a une solution optimale.

Paramètres d'un branch & bound

- Heuristique donnant la fonction d'évaluation (minorant ou majorant).
- Critère de séparation (ou sur quel critère on partitionne un 'noeud' en 2 (ou +) sous-noeuds).
- Développement de l'arborescence : construction des noeuds successeurs, 'à l'aide' d'une heuristique.
- Stratégie de développement de l'arborescence : en profondeur ou en largeur d'abord.

Exemple : algorithme de Little pour le PVC

- Fonction d'évaluation = borne min de Little (restreinte au sous-problème).
- A chaque sommet 2 successeurs : soit on va ensuite dans la ville X, soit on n'y va pas.
- Développement d'un sommet (= choix de la ville suivante X) : par la méthode de pénalité (=regret maximal) du sous-problème.
- Stratégie en largeur d'abord.

Problème du voyageur de commerce

- On doit effectuer un circuit en avion en passant une seule fois par chaque ville, et en minimisant le coût du trajet.
- On le modélise par la recherche d'un cycle hamiltonien de coût minimal dans un graphe non orienté valué, simple et complet.
- C'est un problème NP-complet : 'prototype' des problèmes NP-difficiles.

Algorithme de Little

- On modélise le PVC par une matrice donnant les distances entre les différentes villes (pas nécessairement symétriques)

	A	B	C	D
A	-	3	2	3
B	1	-	4	2
C	2	3	-	2
D	1	3	3	-

Villes d'arrivée

Villes de départ

- On réduit la matrice :
  - Soustraire le plus petit élément de chaque ligne,
  - Puis soustraire le plus petit élément de chaque colonne.
  - Ce faisant on n'a pas changé le problème.
  - La somme des nombres ainsi soustraits est l'*évaluation* initiale.
  - C'est un minorant de la longueur du trajet.

Réduction

	A	B	C	D	
A	-	3	2	3	-2
B	1	-	4	2	-1
C	2	3	-	2	-2
D	1	3	3	-	-1

Réduction

	A	B	C	D	
A	-	1	0	1	2
B	0	-	3	1	1
C	0	1	-	0	2
D	0	2	2	-	1

Coût minimal pour quitter les 4 villes : 6

Réduction

	A	B	C	D
A	-	1	0	1
B	0	-	3	1
C	0	1	-	0
D	0	2	2	-

-1

Coût minimal pour quitter les 4 villes : 6

Réduction

	A	B	C	D
A	-	0	0	1
B	0	-	3	1
C	0	0	-	0
D	0	1	2	-

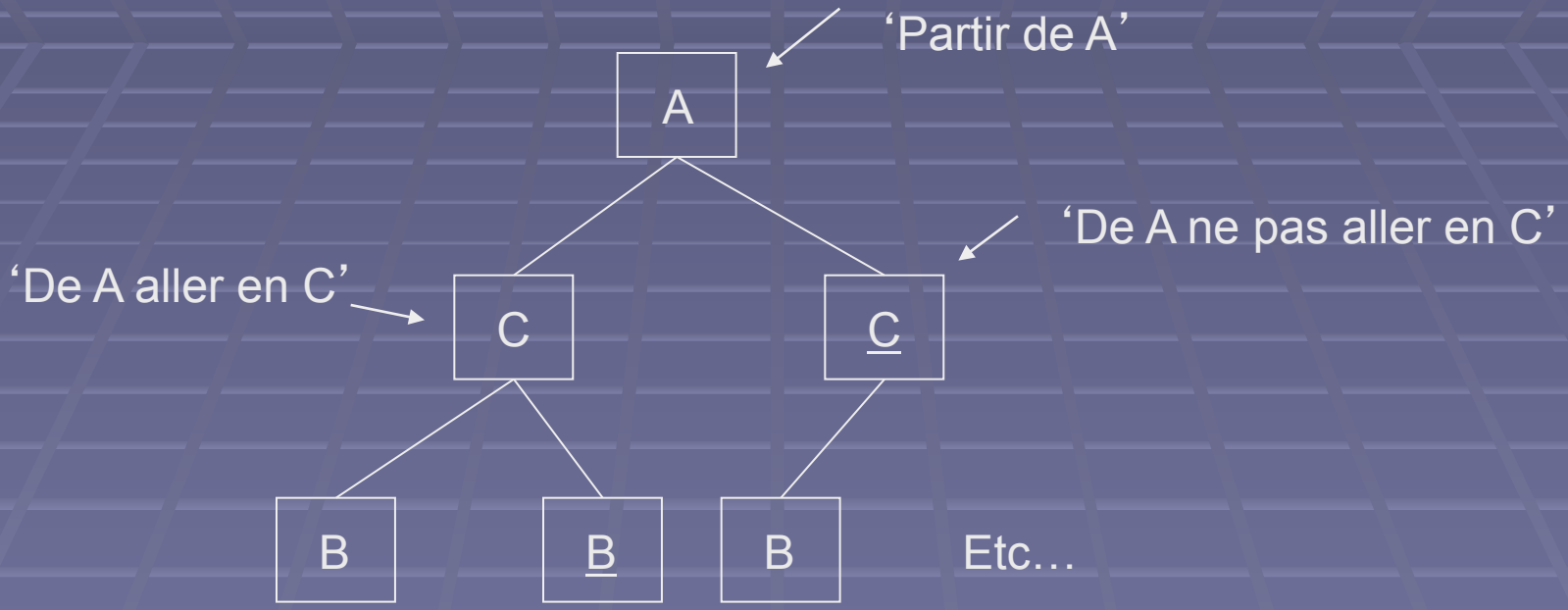
1

Coût minimal pour quitter les 4 villes : 6

Sur-coût minimal pour regagner les 4 villes : 1

Evaluation initiale : 7

- On va procéder à un développement par arborescence :



Chaque sommet est muni d'une évaluation. C'est un minorant du coût d'un parcours empruntant ce sommet. La racine est munie de l'évaluation initiale.

Comment développer l'arborescence.

- On emprunte un trajet de coût minimal (0 dans la matrice réduite).
- Pour chacun de ces trajets on calcule son **regret** : pour l'élément AC c'est le surcoût minimum pour quitter A sans aller en C, plus le surcoût minimum pour aller en C sans venir de A.
- C' est la somme du plus petit coût de la ligne A hormis AC, et du plus petit coût de la colonne C hormis AC.
- On prend initialement un trajet de coût minimal et de regret maximal.

Comment attribuer une évaluation à un sommet

- Pour un sommet ‘choix négatif’ son évaluation est la somme de l’évaluation de son père et de son regret.
- Pour un sommet ‘choix positif’, son évaluation est la somme de l’évaluation de son père et de celle provenant de sa nouvelle matrice réduite

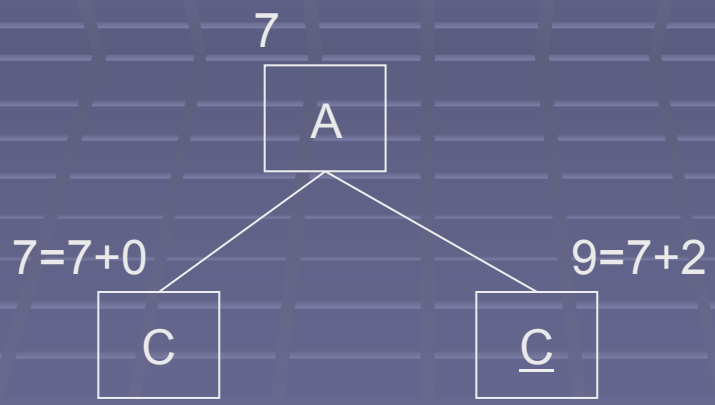
- On a choisi initialement AC
- La nouvelle matrice associée à ce choix est obtenue en supprimant ligne A, colonne C, et en interdisant le retour CX pour tout sommet X déjà rencontré dans cette branche (CA en particulier).
- (les regrets seront a recalculer par la suite)
- La réduction de cette matrice donne un surcoût (ici 0).
- Ajouté à l'évaluation du sommet père, cela donne l'évaluation du sommet 'positif' AC.

	A	B	C	D
A	-	0(0)	0(2)	1
B	0(1)	-	3	1
C	0(0)	0(0)	-	0(1)
D	0(1)	1	2	-

	A	B	D
B	0	-	1
C	-	0	0
D	0	1	-

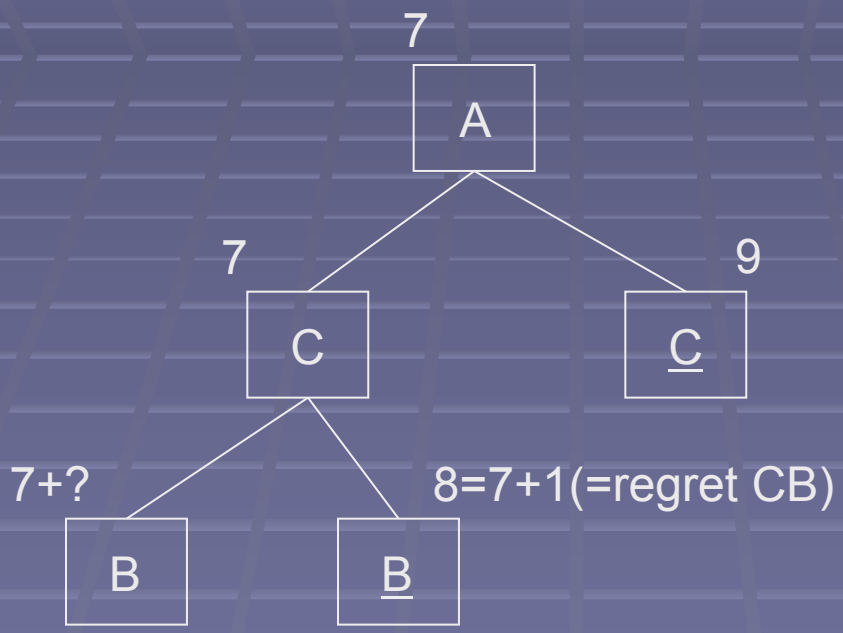
Développement de l'arborescence

- A chaque étape, on développe en prenant dans la matrice associée le choix de surcoût minimal et de regret maximal.
- On calcule les évaluation des choix (positifs et négatifs).
- On poursuit l'exploration d'une branche tant que son évaluation est inférieure aux autres.
- Sinon on rebranche à partir d'un autre choix (négatif). (Par exemple) pour le choix AC on reprend la matrice associée à ce niveau, en remplaçant la valeur de AC par '-' : on rend le choix AC impossible.



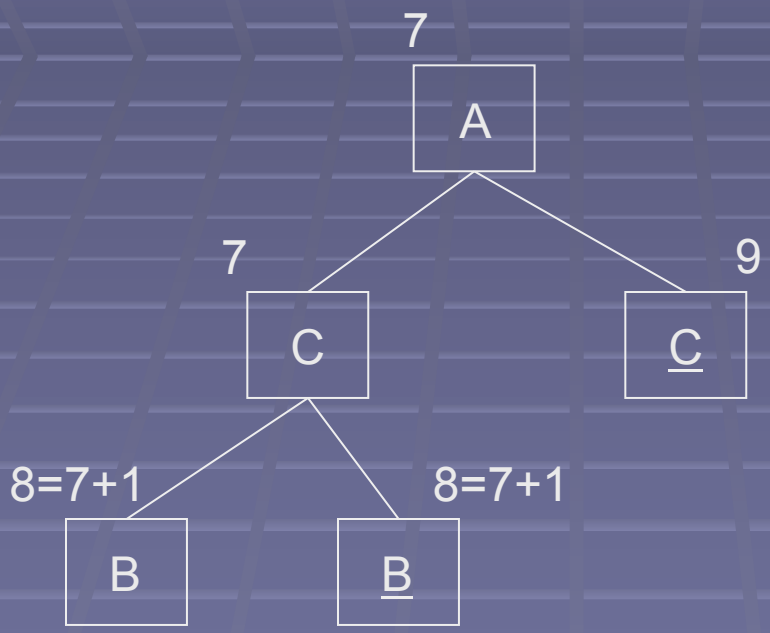
	A	B	D
B	0	-	1
C	-	0(1)	0(1)
D	0	1	-

0

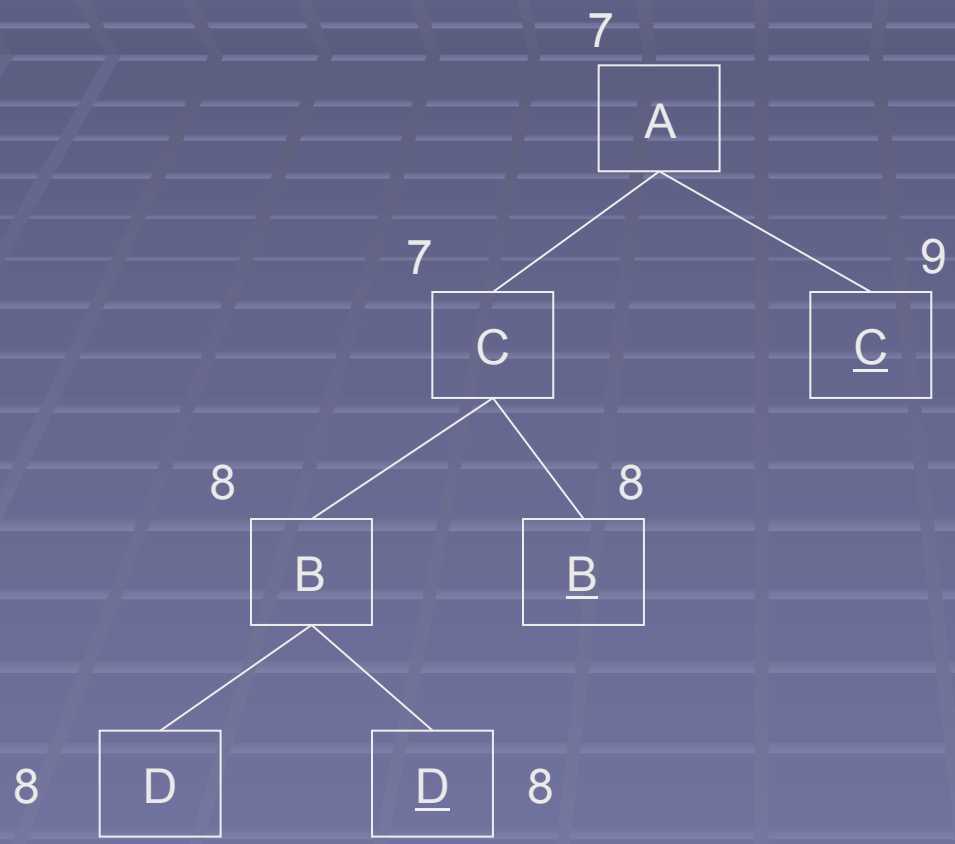


	A	D
B	-	1
D	0	-

-1

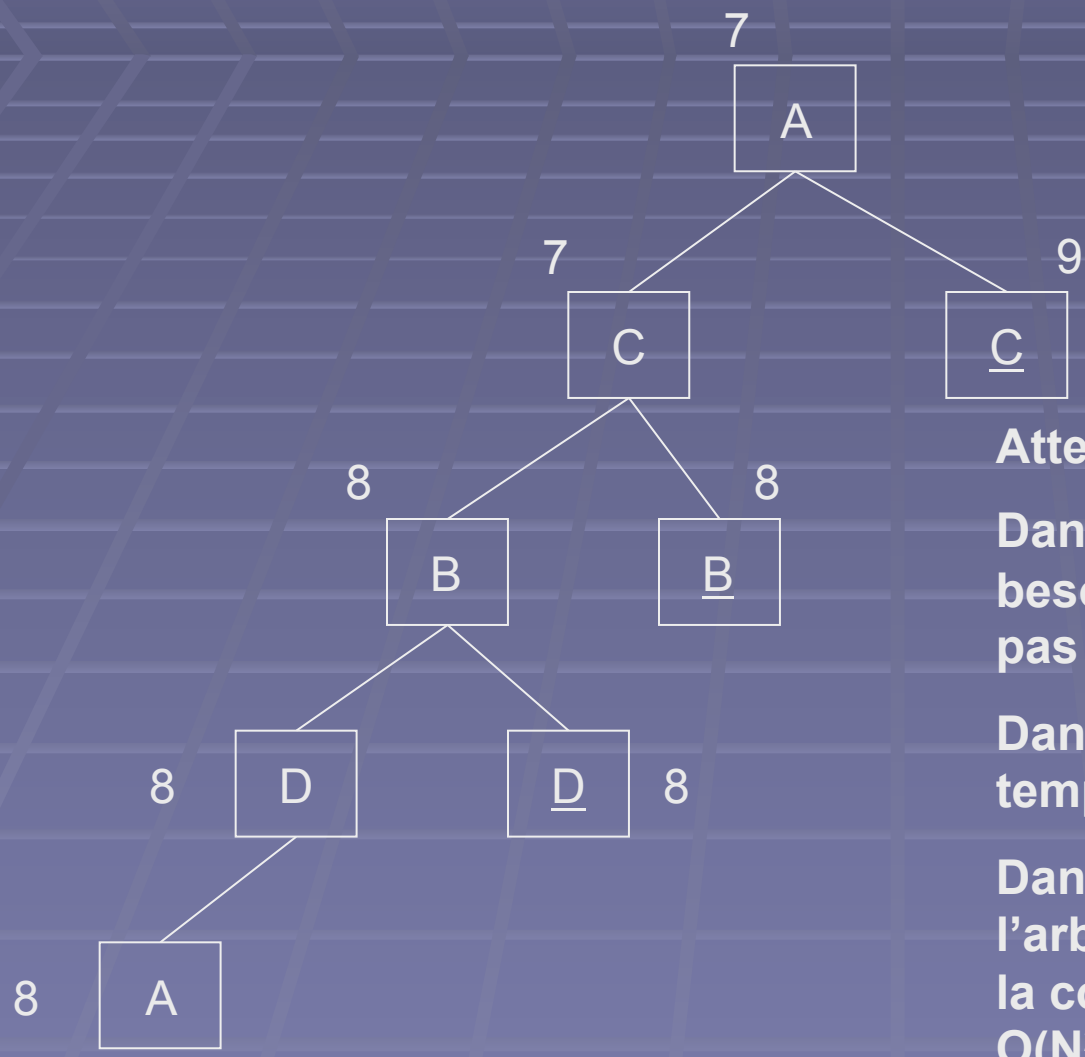


	A	D
B	-	0(0)
D	0(0)	-



	A
D	0

Le chemin hamiltonien A-C-B-D-A est de cout optimal = 8



Attention !

Dans cet exemple on n'a pas eu besoin de rebrancher … Ce n'est pas toujours le cas (cf. TD).

Dans ce cas la résolution est en temps polynomial $O(N^3)$!!!

Dans le pire des cas où toute l'arborescence serait développée la complexité est exponentielle $O(N^2 \times 2^N)$!!

Autre exemple : le problème du sac à dos 0-1

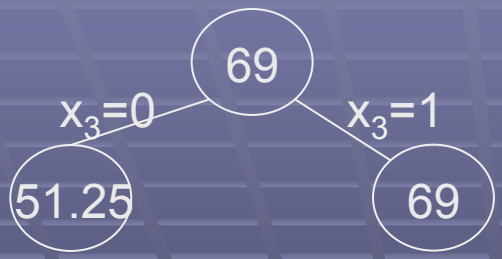
- n objets disponibles. L'objet i a un poids $p_i>0$ et une valeur $v_i>0$. Emporter un sous-ensemble d'objets de valeur maximale dans un sac de capacité P.
- **Exemple :**
  - n=5,
  - p=(18,12,15,16,13),
  - v=(27,9,30,16,6.5),
  - P=45.

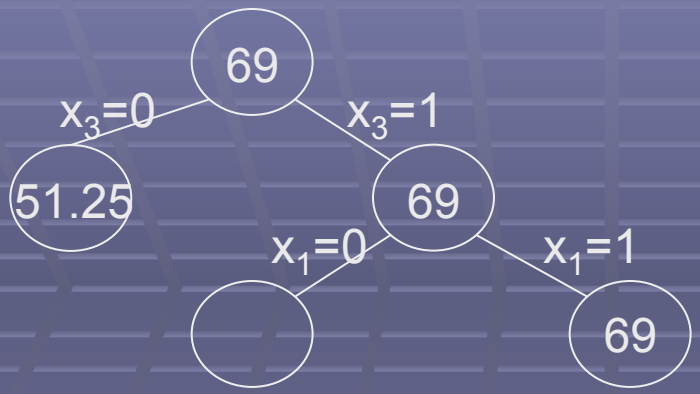
- **Règle de séparation.** En un noeud 2 successeurs, selon si l'on prend ou non l'objet x_i, i=1,...,5.
- Heuristique pour le développement : le choix des x_i dans le critère ci-dessus se fait par ordre décroissant du rapport v_i/p_i valeur sur poids.
- Fonction d'évaluation. Par cette même heuristique gloutonne on remplit tant que possible par v_i/p_i décroisant. Puis on ajoute une portion du suivant de sorte à arriver au poids P. La valeur obtenue est un majorant (problème max).

Table d'utilité vi/pi décroissante :

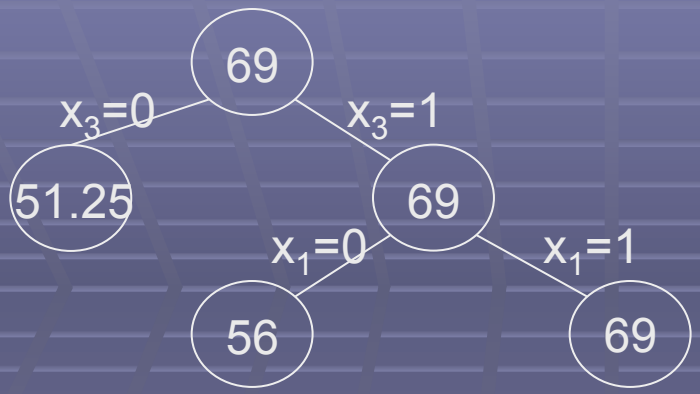
Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5

- **Evaluation initiale** : on prend les objets 3, 1 (poids=23) et 12/16 de l'objet 4. On parvient au poids P=45. La valeur est 30+27+16x12/16=**69**.
- On **développe** ensuite selon si l'on prend ou non l'objet 3.
- Si x_3=0. Evaluation : objets 1, 4 et 11/12 de l'objet 2 : 27+16+9x11/12=**51.25**.
- Si x_3=1. Evaluation : **69**.


69
x_3=0
x_3=1
51.25
69



Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5

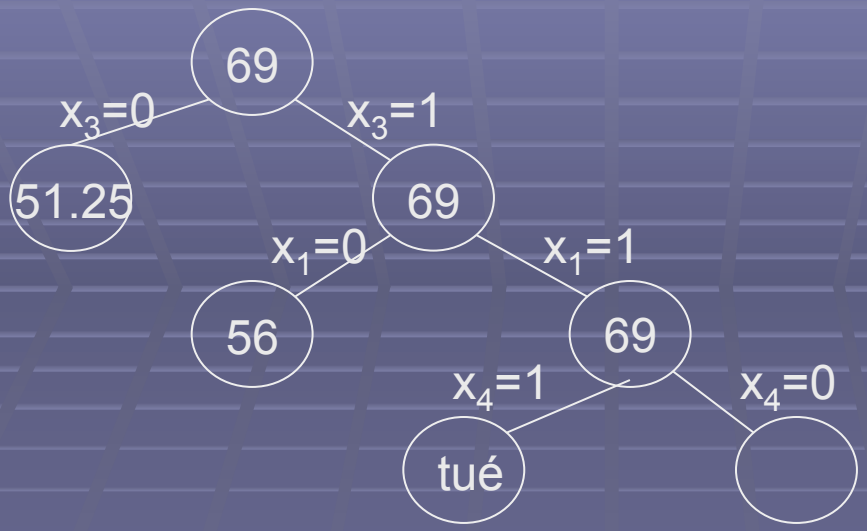


objets 3,4,2 :p=43

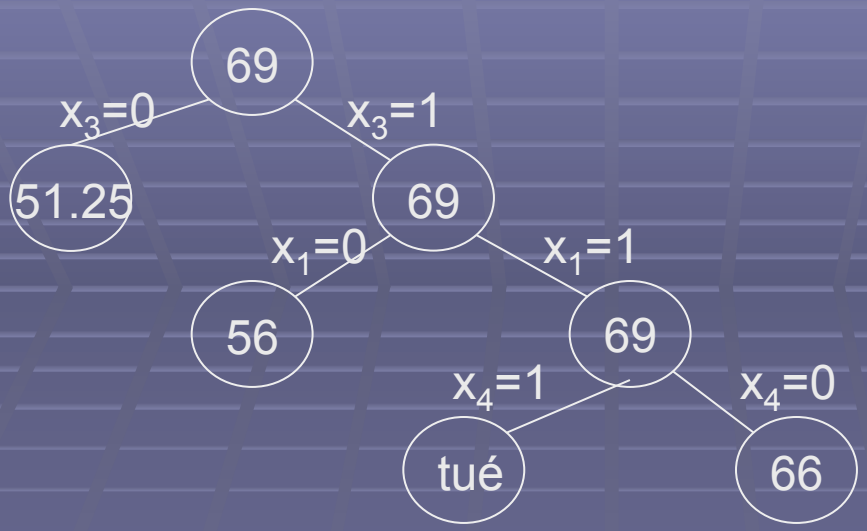
objets 3,4,2, 2/13 de 5

V=30+16+9+6.5x2/13=56

Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



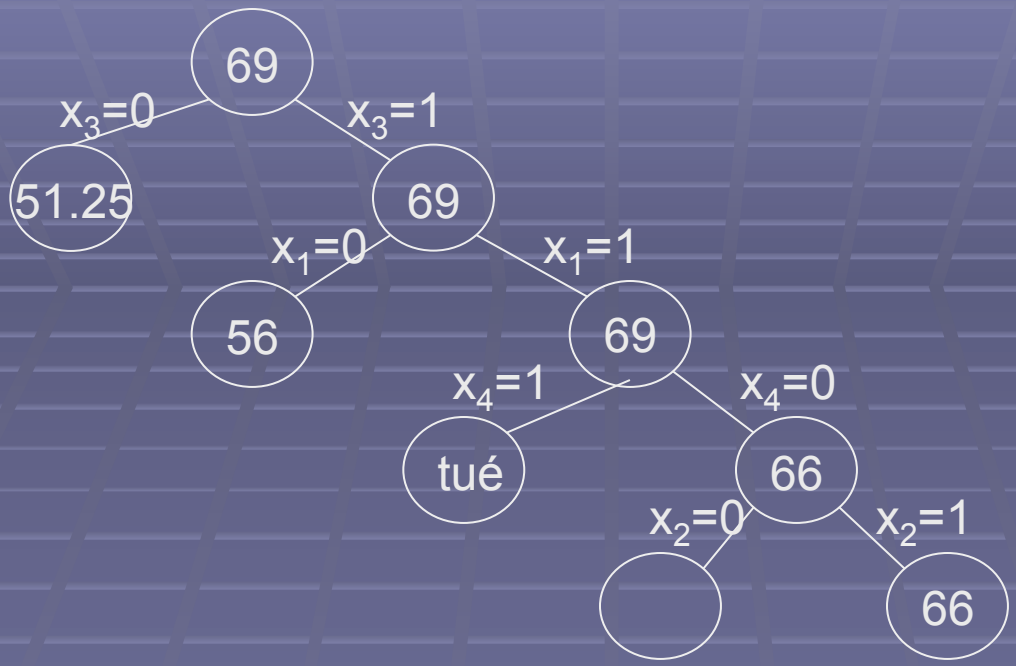
Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



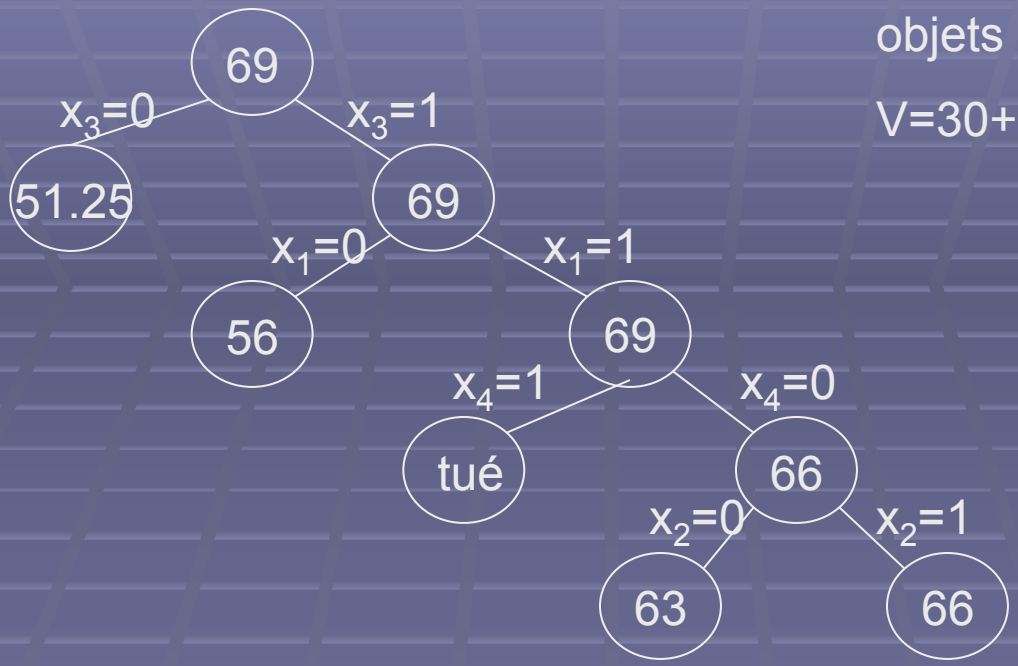
objets 3,1,2 :p=45

V=30+27+9=66

Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



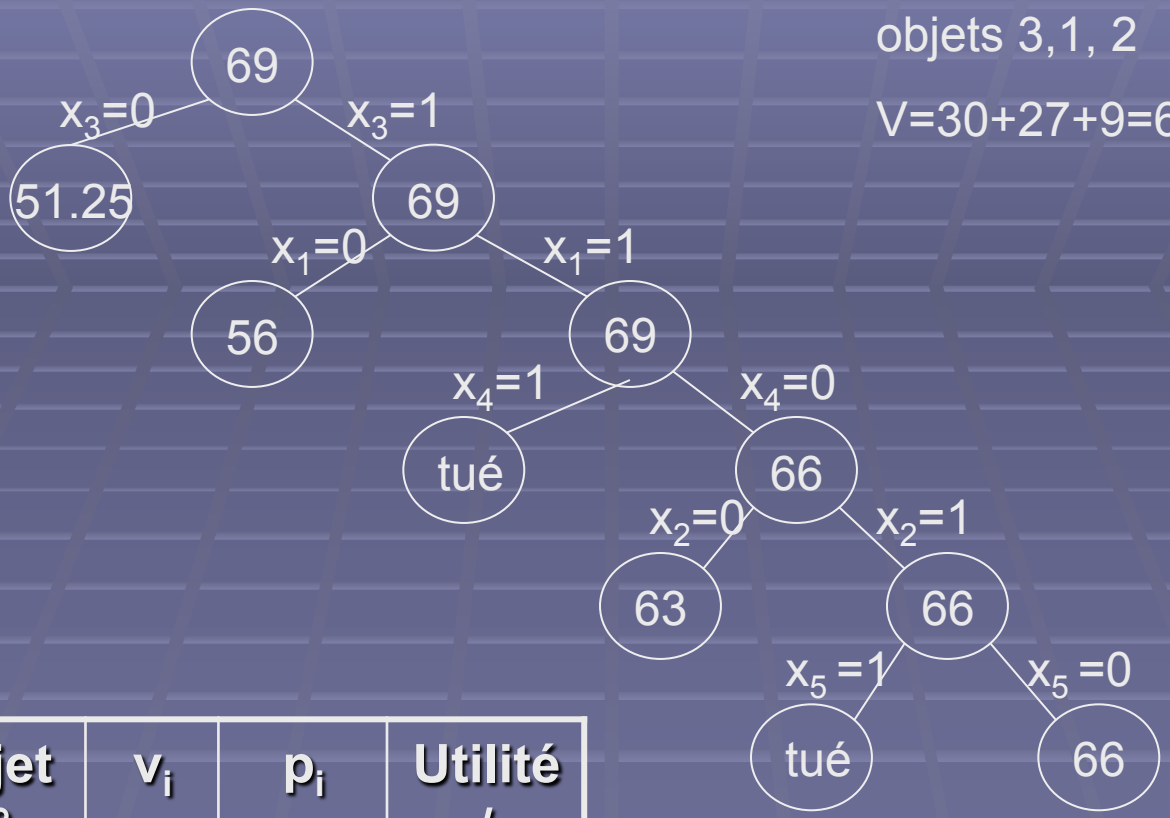
Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



objets 3,1, 12/13 de 5

V=30+27+6.5x12/15=63

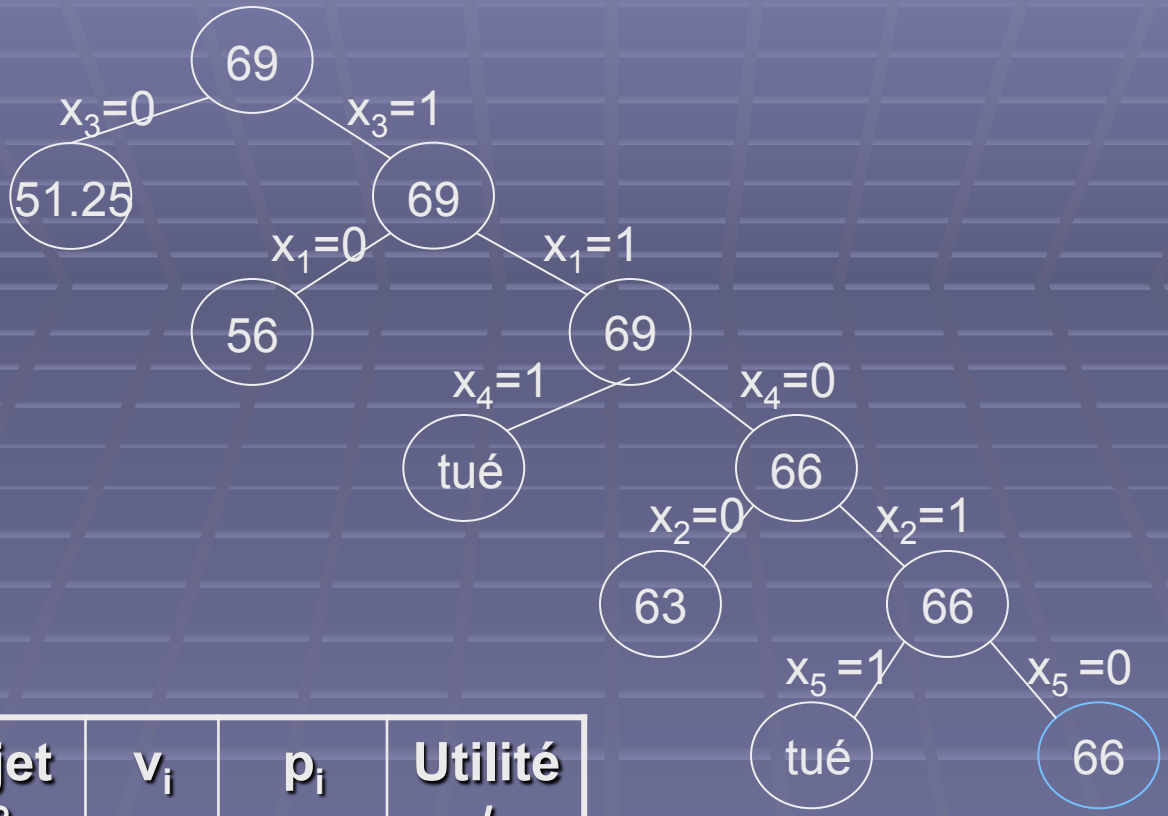
Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



objets 3,1, 2

V=30+27+9=66

Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5



Rang	Objet n°	v_i	p_i	Utilité v_i/p_i
1	3	30	15	2
2	1	27	18	1.5
3	4	16	16	1
4	2	9	12	0.75
5	5	6.5	13	0.5

Solution optimale :

objets 3 & 1 & 2,

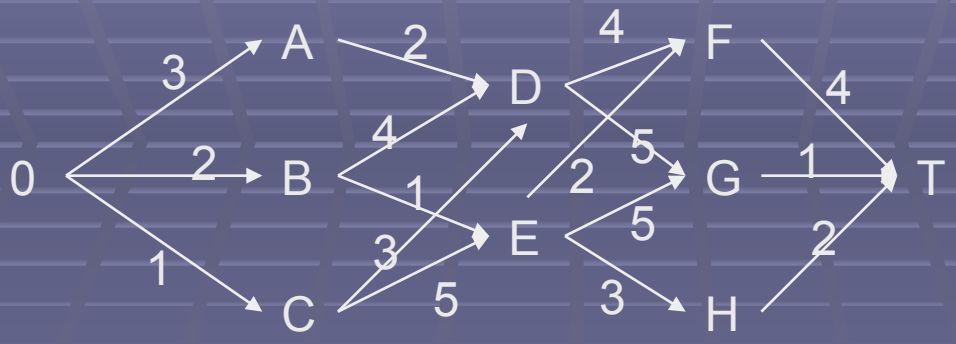
Valeur =66

Poids=45.

Programmation dynamique

- C'est un paradigme de résolution qui s'applique dès qu'un problème vérifie le principe de Bellman :
- Soit un problème d'optimisation sur n objets. Une solution optimale vérifie que toute sous-solution restreinte à n-1 objets est aussi optimale.
- Exemple : Recherche de géodésique. Tout sous-chemin d'une géodésique est aussi une géodésique.

Exemple : recherche de géodésique

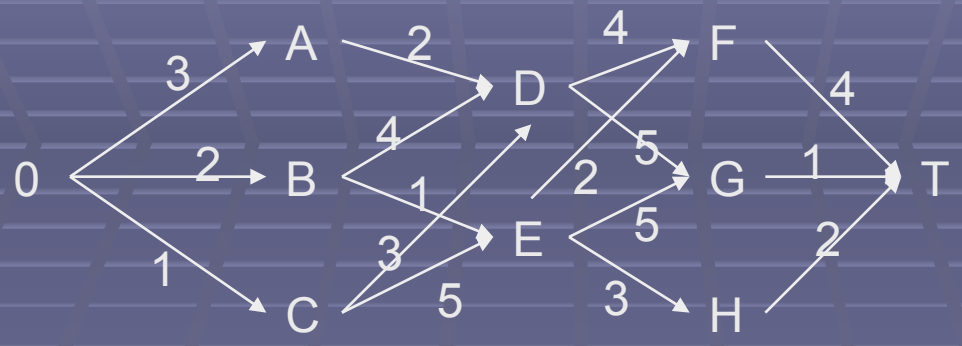


Recherche de géodésique de $x_0=0$ à $x_4=T$

$x_0=0$; $x_1=\{A,B,C\}$; $x_2=\{D,E\}$; $x_3=\{F,G,H\}$; $x_4=T$

x_1	Géodésique	Valeur
A	OA	3
B	OB	2
C	OC	1

Exemple : recherche de géodésique

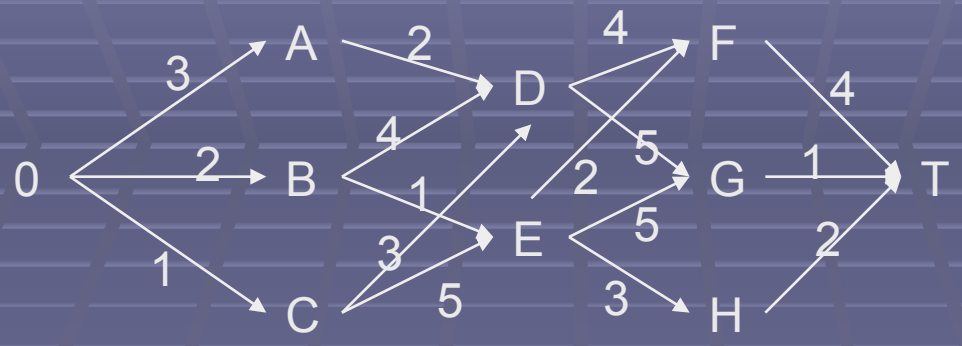


Recherche de géodésique de $x_0=0$ à $x_4=T$

$x_0=0$; $x_1=\{A,B,C\}$; $x_2=\{D,E\}$; $x_3=\{F,G,H\}$; $x_4=T$

x_2	chemins	géodésique	valeur
D	OAD OBD OCD	OCD	4
E	OBE OCE	OBE	3

Exemple : recherche de géodésique

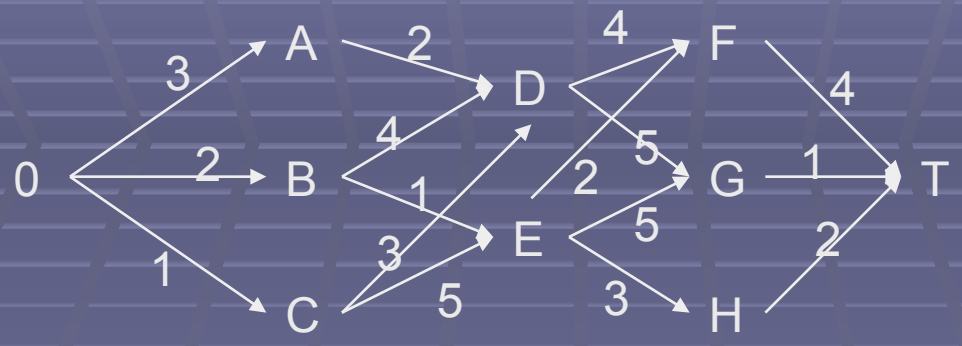


Recherche de géodésique de $x_0=0$ à $x_4=T$

$x_0=0$; $x_1=\{A,B,C\}$; $x_2=\{D,E\}$; $x_3=\{F,G,H\}$; $x_4=T$

x_3	Chemins	Géodésique	Valeur
F	OCDF OBEF	OBEF	5
G	OCDG OBEG	OBEG	8
H	OBEH	OBEH	6

Exemple : recherche de géodésique



Recherche de géodésique de $x_0=0$ à $x_4=T$

$x_0=0$; $x_1=\{A,B,C\}$; $x_2=\{D,E\}$; $x_3=\{F,G,H\}$; $x_4=T$

x_4	Chemins	Géodésique	Valeur
T	OBEFT OBEGT OBEHT	OBEHT	8

Programmation dynamique pour le PVC

- On note {1,2,…,n} les villes et l'on suppose que l'on part de la ville 1. Soit d_{ij} la distance entre la ville i et la ville j.
- Soit D(i,S) la longueur d'une géodésique de i à 1 passant une et une seule fois par chaque ville de S.
- $D(i,S)=\min_{j \in S} \{d_{ij}+D(j,S-\{j\})\}$.
- La solution optimale est D(1,{2,…,n}).
- Clairement $D(i,\{1\})=d_{i1}$.
- De complexité dans le pire des cas $O(n^2 2^n)$.

- On construit par récurrence tous les D(i,P) pour P une partie de {2,...,n} et i=2,..,n, $i \in P$, à l'aide de la relation de récurrence.
- On en déduit à l'aide de cette même relation D(1,{2,...,n}) qui donne le coût optimal.
- Pour construire le cycle optimal on retient à partir du coût optimal l'indice j l'ayant déterminé (c'est la ville suivante visitée), puis l'on continue de proche en proche.

- Exemple avec 4 villes :
- $D(2,\emptyset)=5$; $D(3,\emptyset)=6$; $D(4,\emptyset)=8$.
- $D(2,\{3\})=d_{23}+D(3,\emptyset)=9+6=15$;
- $D(2,\{4\})=d_{24}+D(4,\emptyset)=10+8=18$;
- $D(3,\{2\})=d_{32}+D(2,\emptyset)=13+5=18$;
- $D(3,\{4\})=d_{34}+D(4,\emptyset)=12+8=20$;
- $D(4,\{2\})=d_{42}+D(2,\emptyset)=8+5=13$;
- $\underline{D(4,\{3\})}=\underline{d_{43}+D(3,\emptyset)}=9+6=15$.
- $\underline{D(2,\{3,4\})}=\min\{d_{23}+D(3,\{4\})\ ;\ \underline{d_{24}+D(4,\{3\})}\}=25$;
- $D(3,\{2,4\})=\min\{d_{32}+D(2,\{4\})\ ;\ d_{34}+D(4,\{2\})\}=25$;
- $D(4,\{2,3\})=\min\{d_{43}+D(3,\{3\})\ ;\ d_{42}+D(2,\{4\})\}=23$.
- $\underline{D(1,\{2,3,4\})}=\min\{\underline{d_{12}+D(2,\{3,4\})}\ ;\ d_{13}+D(3,\{2,4\})\ ;\ d_{14}+D(4,\{2,3\})\}$ $=\min\{\underline{35},40,43\}=35$.

-	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

Cycle optimal : 1-2-4-3-1