Techniques algorithmiques

Marian Scuturici

Plan

- Force brute
- Greedy (*algorithme glouton*)
- Récursivité (-> Cours 2)
- Backtracking (*retour sur trace*)
- Programmation dynamique

Force brute (BF)

- Brute force / Complete search / Recursive backtracking
- *If all you have is a hammer, everything looks like a nail* (Abraham Maslow, 1962)

- Parcourir tout l'espace de recherche pour trouver la/les solution(s).

Exemple – BF1

- Pour un tableau d'entiers, trouver la plus grande différence entre deux éléments.

Procédure $BF1(tab, k)$
    Entrée : entier $tab[1..k]$
    Sortie : entier $diffMax$
    début
        $diffMax \leftarrow 0$;
        pour $(i = 1; i < k; i \leftarrow i + 1)$ **faire**
            pour $(j = 1; i < k; k \leftarrow k + 1)$ **faire**
                si $(abs(tab[i] - tab[j]) > diff)$ **alors**
                    $diffMax \leftarrow abs(tab[i] - tab[j])$;

Complexité ?

Exemple – BF1

- Solution différente possible

Procédure $BF1(tab, k)$
  Entrée : entier $tab[1..k]$
  Sortie : entier $diffMax$
  début
    $diffMax \leftarrow max(tab) - min(tab);$

Complexité ?

Problème – BF2

Pour deux entiers *A* et *B*, avec $1 \leq A, B \leq 10000$ énumérer tous les entiers *x*, *y* respectant les contraintes suivantes :

$$x * y = A$$
$$x^2 + y^2 = B$$

Problème – BF3

(UVa507) : Pour un tableau d'entiers A[0..n-1] trouvez les indices i <= j déterminant une sous-séquence de somme maximale : A[i]+A[i+1]+…+A[j]

(n < 20000)

Algorithme glouton - GR

- Greedy
- Heuristique !
- Optimisation locale avec l'espoir d'arriver éventuellement à une solution globale.
- Cette méthode ne garanti pas l'obtention d'une solution globale, mais :
  - Complexité faible
  - Approximation de la solution globale

Exemple – GR1

- Problème du rendu de monnaie
- Etant donné un système de monnaie S (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?
- Exemple : 16 euros

Exemple – GR1

- Cas 1 :
  - S = (1, 2, 5, 10, 20, 50, 100, 200, 500)
  - 1989 = 500×3 + 200×2 + 50×1 + 20×1 + 10×1 + 5×1 + 2×2
- Cas 2 :
  - CE : S = (1, 3, 4)
  - 6 = 4+1+1 = 3+3

Exemple – GR1

- Algorithme glouton
  - tant qu'il reste quelque chose à rendre, choisir la plus grosse pièce qu'on puisse rendre

Problème - GR

Des locaux de *poste* doivent être construits dans un ensemble de villes. Chaque ville est définie par une position (x,y), et chaque ville doit se trouver à moins de 10 km d'un local de poste.

Proposez une stratégie pour installer le moins de locaux de poste pour un ensemble de villes donné, tout en respectant la contrainte précédente (<= 10 km).

(*Set cover problem*)

Récursivité

- Proposez un algorithme récursif pour calculer :

$$2^{2^k}$$

- Proposez un algorithme récursif pour vérifier si un nombre réel *x>=0* est naturel.

Récursivité

```
Fonction entier puissance2puissancek(k)
    Entrée          : entier k
    Précondition    :
    Postcondition   : ...
    Déclaration     :  entier puissanceTemp
    début
        si k = 0 alors
            retourner 2;
        sinon
            puissanceTemp ← puissance2puissancek(k − 1);
            retourner puissanceTemp ∗ puissanceTemp;
```

Backtracking (Retour sur trace) - BK

- Explorer l’espace de recherche dans plusieurs directions, et éliminer le plus tôt les directions sans solution.

Exemple - BK

- Positionner *N* dames sur une table d'échecs de taille *N*. (N-Queens)

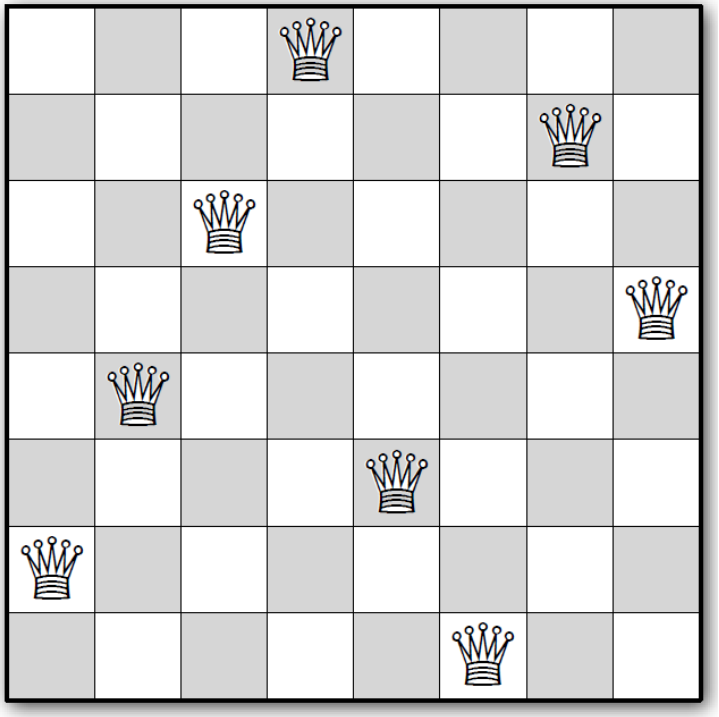
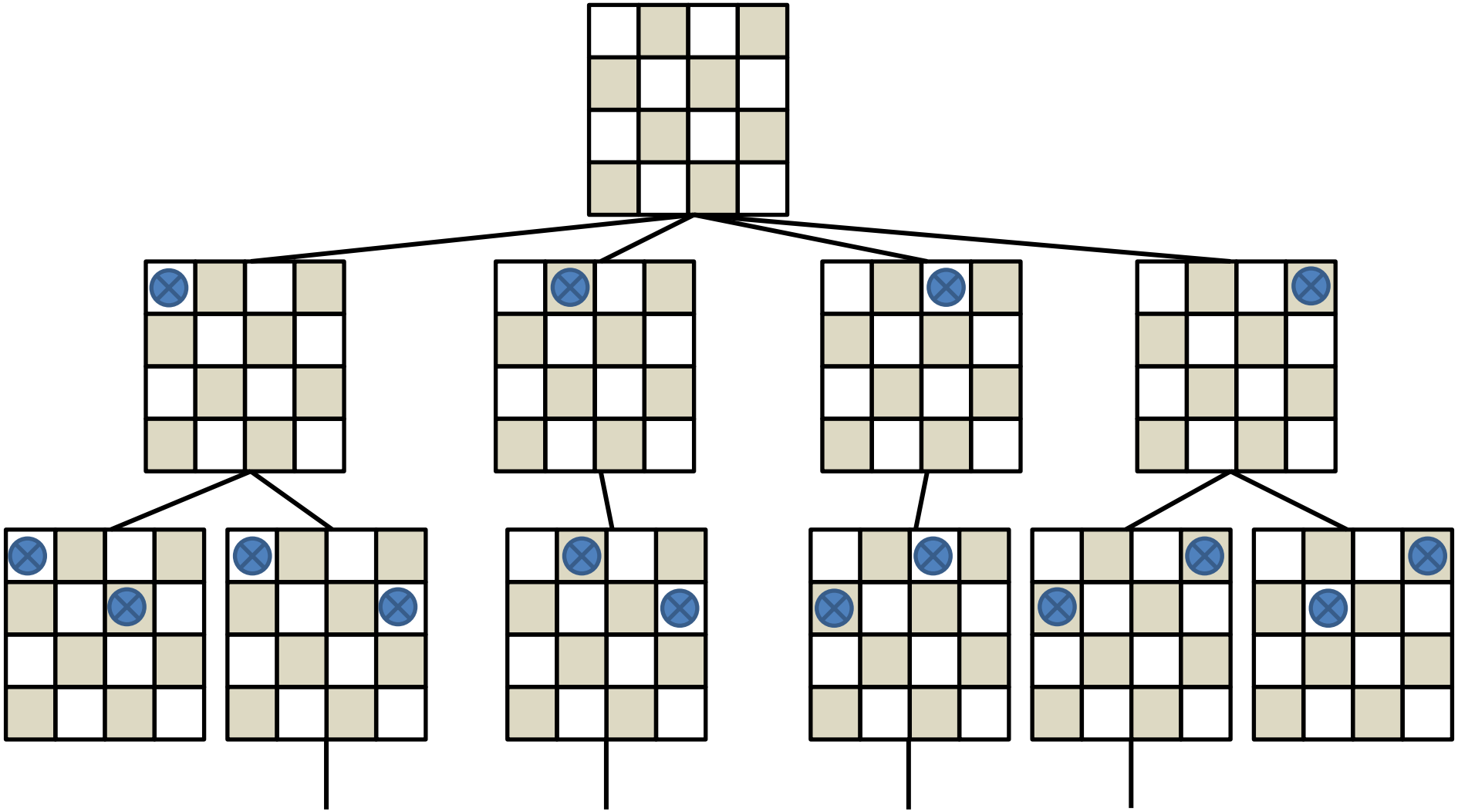
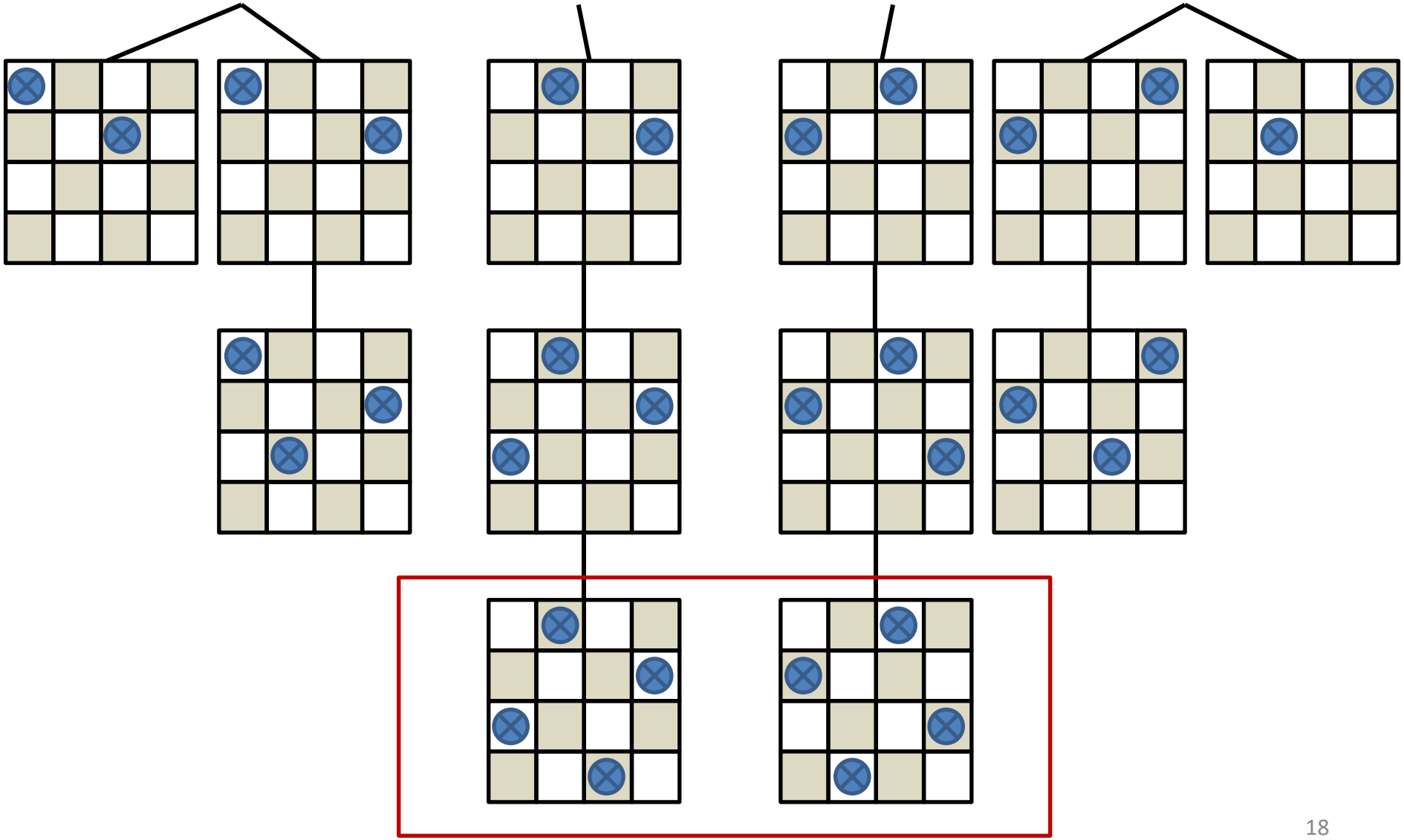


Image : Jeff Erickson, http://www.cs.illinois.edu/~jeffe/teaching/algorithms

Exemple - BK



Exemple - BK



Exemple - BK

```
Procédure n_queens(pos[1..n], level)
    Entrée          : entier pos[1..n]
    Sortie          : entier level
    début
        si level = n + 1 alors
            afficher(Q);
        sinon
            pour (j = 1; j <= n; j ← j + 1) faire
                valid ← true;
                pour (i = 1; i < level; i ← i + 1) faire
                    si (pos[i] = j) ∨ (pos[i] = j + level − i) ∨ (pos[i] = j − level + i) alors
                        valid ← false;
                si valid alors
                    pos[level] ← j;
                    n_queens(pos[1..n], level + 1);
```

Problème - BK

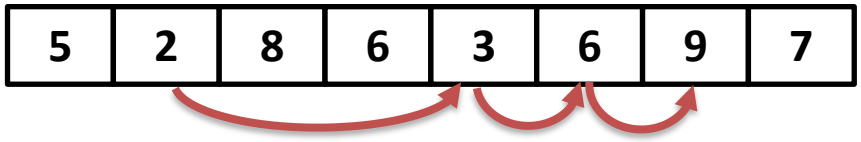
- Générer les permutations d'un vecteur
- Exemple :

  (1, 2, 3) ->

  (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)

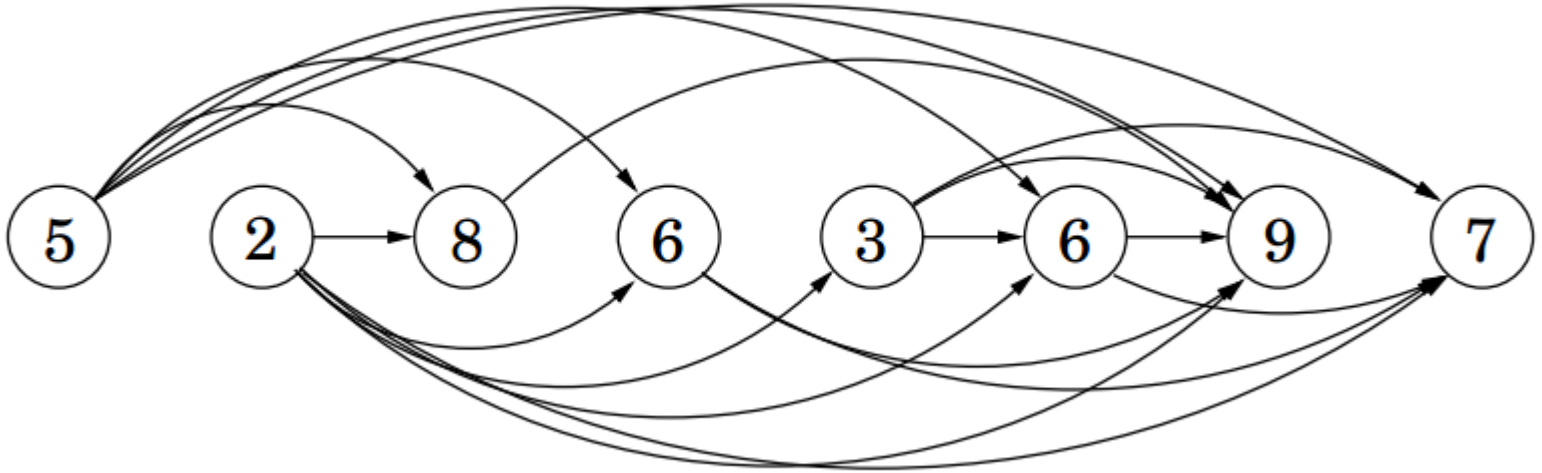
Programmation dynamique - exemple

- Trouver la plus longue subséquence monotone
  - *longest increasing/decreasing subsequence*



Exemple

- Construction d'un graphe acyclique (DAG) linéarisé
- *L(i)* = la subséquence la plus longue entre le premier nœud et le nœud *i*



Exemple

- Invariant :

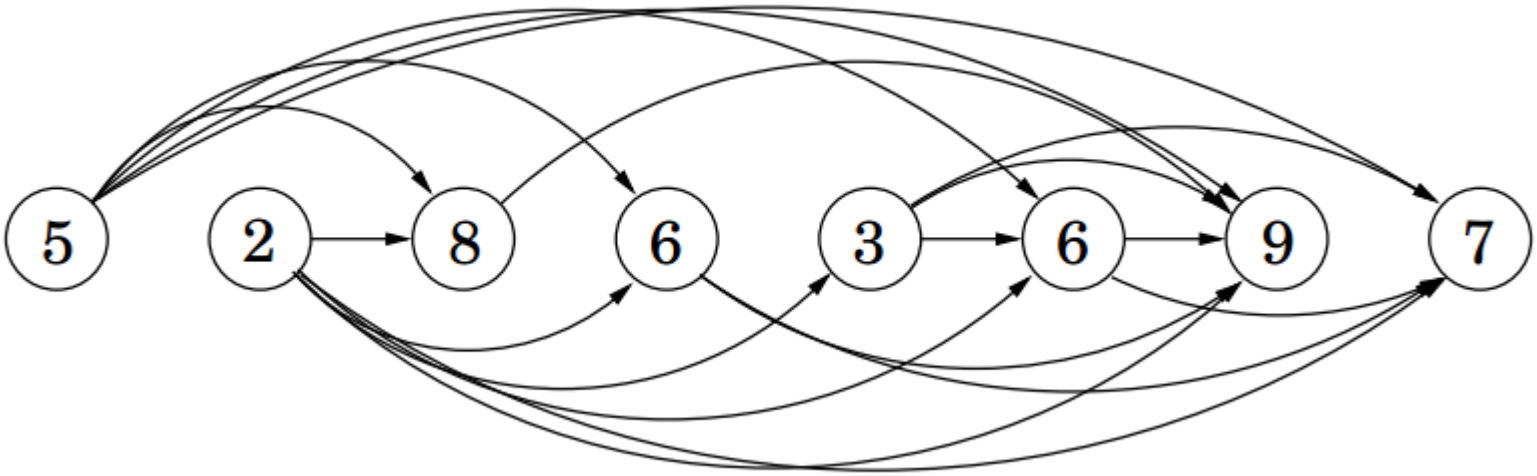
$$L(j) = 1 + max\{L(i)|j = succ(i)\}$$

- Algorithme :

```
1 Fonction entier LIS(A)
2     début
3         pour (i ← 0 ; i < n ; i ← i + 1) faire
4             construire succ(i)
5         pour (j ← 0 ; j < n ; j ← j + 1) faire
6             L(j) = 1 + max{L(i)|j ∈ succ(i)}
7         retourner max_j(L(j))
```

Exemple

i	0	1	2	3	4	5	6	7
Vect(i)	5	2	8	6	3	6	9	7
Succ(i)	()	()	(0, 1)	(0, 1)	(1)	(0, 1, 4)	(0, 1, 2, 3, 4, 5)	(0, 1, 3, 4, 5)
L(i)	1	1	2	2	2	3	4	4



Principe

- Pour résoudre un problème, nous la décomposons dans une collection de sous-problèmes, respectant une relation d'ordre. La relation d'ordre nous permet de résoudre le problème avec un seul passage sur l'ensemble de sous-problèmes.

Quelques commentaires

- Divide et impera ?
- Récursivité ?
- Trouver le DAG associé !

Programmation Dynamique - DP

- Problème (Sac à dos / Knapsack) : pour un ensemble d'objets de poids connus (entiers) et pour un sac à dos avec une capacité connue, trouvez une façon optimale de remplir le sac.

- Observation : chaque type d'objet se trouve en quantité illimitée (exemple : une infinité d'objets de poids 1 kg).

Problème

- Pour un entier N, nous pouvons effectuer l'une des 3 opérations élémentaires :
  - N-1
  - N/2, si N%2==0
  - N/3, si N%3==0
- Pour un N donné, quel est le nombre minimal d'opérations élémentaires pour arriver à 0 ?

Problème – DP

- Problème du rendu de monnaie
- Etant donné un système de monnaie S (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?
- Proposez une solution avec un algorithme DP

Programmation Dynamique 1 – DP1

- Problème (Sac à dos / Knapsack) : pour un ensemble d'objets de poids connus (entiers) et pour un sac à dos avec une capacité connue, trouvez une façon optimale de remplir le sac.

- Observation : chaque objet peut être utilisé une infinité de fois !

Programmation Dynamique 2 – DP2

- Problème (Sac à dos / Knapsack) : pour un ensemble d'objets de poids connus (entiers) et pour un sac à dos avec une capacité connue, trouvez une façon optimale de remplir le sac.

- Observation : chaque objet peut être utilisé une seule fois !

Programmation dynamique - Références

- S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*, Berkeley (disponible en PDF)
- Cours MIT (Cormen, …)

?

Problème : palindromes

Soit un ensemble de nombres naturelles positives $X = \{x_1, x_2, \ldots, x_n\}$. Proposez un algorithme qui pour l'entrée X produit la valeur 1, si X est un palindrome, et la sortie 0 en cas contraire. X est un palindrome s'il peut se lire dans les deux sens avec la même valeur.

Problème 2

Pour un entier N, nous pouvons effectuer l'une des 3 opérations élémentaires :

- N-1
- N/2, si N mod 2 = 0
- N/3, si N mod 3 = 0

Pour un N donné, quel est le nombre minimal d'opérations élémentaires pour arriver à 0 ?