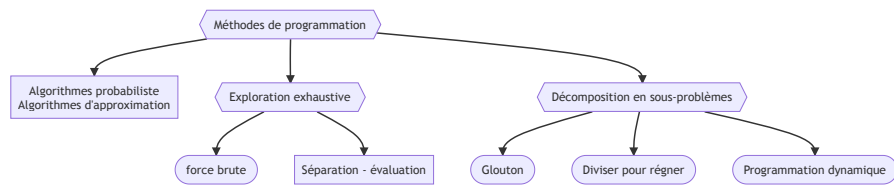


Programmation dynamique

Objectifs du cours:

- Cadre pour les techniques de programmation
- Programmation dynamique

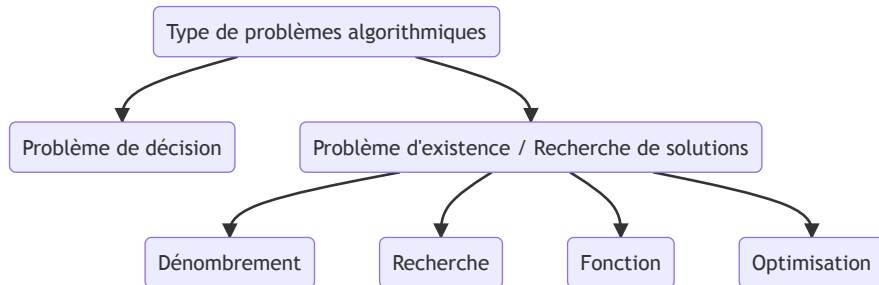
I. Introduction - Cadre pour l'algorithmique



Formulation d'un problème en algorithmique:

- Soit P un problème quelconque
- Soit I une instance de P
- Soit A un algorithme résolvant le problème P

Quelles sont les familles de problèmes P ?

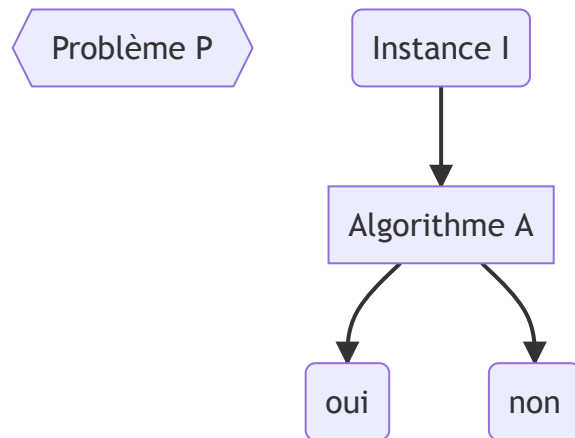


Problème de décision (ou reconnaissance)

Objectif : validation de l'existence d'une solution

A chaque instance du problème P , la réponse apportée par l'algorithme est oui ou non (résultat binaire)

Problème de classification



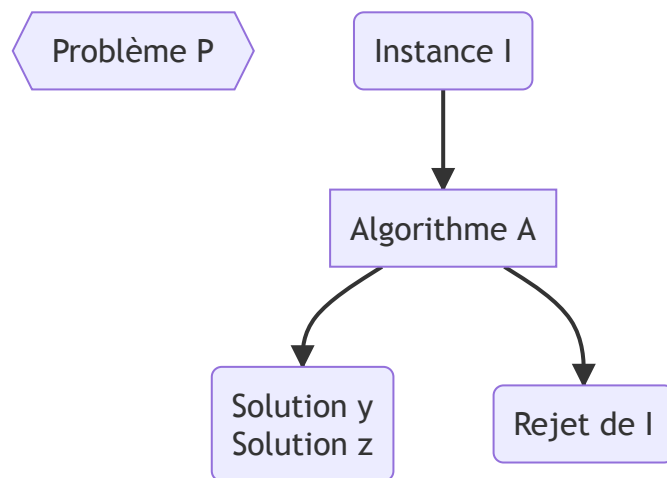
Exemple: soit n un entier naturel, déterminer si n est un nombre premier

Problème de recherche

Objectif: trouver une solution lorsqu'un algorithme n'est pas écrit mais que la spécification de la solution est connue.

Trouver, si elle existe, une solution associée à une instance d'entrée grace à une relation figurant dans l'énoncé de la recherche.

Problème de décision : restriction d'un problème de recherche



Exemple : trouver les facteurs premiers d'un entier naturel n

Problème de dénombrement

Objectif : dénombrer les solutions

Déterminer le nombre de solutions à un problème de recherche

Sous problème du problème de dénombrement et de décision :

- unicité d'une solution optimale
- énumération des solutions

Exemple: soit n un entier naturel, déterminer le nombre de facteurs premiers non triviaux de n

Problème d'optimisation

Objectif : recherche de la meilleure solution parmi toutes les solutions possibles d'un problème P sur une instance I

Détermination d'un jeu de paramètres en entrée d'une fonction donnant à cette fonction la valeur maximale ou minimale

Optimisation sous contrainte ou sans contrainte (méthode de descente, programmation linéaire, ...)

Exemple: recherche du plus grand facteur premier non trivial d'un entier naturel n

Problème de fonction

Objectif : engendrer les solution

Un seul résultat attendu pour chaque entrée

Plus complexe que le problème de décision car le résultat est une valeur plutôt qu'un résultat binaire

Exemple: problème de régression \rightarrow prévision de valeur

II. Programmation dynamique: exemple

Découpe de barres d'acier

Entreprise \rightarrow achat et revente de barres d'acier

Coupe de barres d'acier pour optimiser les profits

Hypothèses:

- Découpe d'une barre en barres de $i = 1, 2, \dots$ cm
- Prix des tronçons de i cm à la revente connus :

Longueur i (cm)	1	2	3	4	5	6	7	8	9	10
Prix	1	5	8	9	10	17	17	20	24	30

Bilan des présentations:

- Pas facile de présenter un code \implies algorithme ?
- Attention à la lisibilité (couleur et taille)
- Trop de code \rightarrow difficile à appréhender
- Position / pertinence des commentaires
- Pertinence des éléments sur les diapos
 - Utilité
 - Aide / illustration du discours
- Définir un plan
- Exemples
- Éviter trop de textes / phrases trop longues
- Travail en équipe \rightarrow répartition des rôles
 - oral
 - élaboration du diapo
 - relecture des diapos

Présentation orale:

1. Contexte / Présentation du problème
2. Plan
3. Méthodes utilisées
 - Formulation mathématique du problème (éventuellement)
 - Choix / justification du choix d'une méthode de programmation
4. Détail / Mise en œuvre des méthodes
 - Architecture global ? (liste des fonctions, et rôles de ces fonctions)
 - Algorithmes pour les principes (à dérouler ?)
 - Code pour les particularités de traduction \rightarrow exploitation des langages
 - Correction / terminaison des algorithmes
5. Bilan / Comparaison des performances ? \rightarrow complexité

Le revenu maximum est donc $r_n = \max(p_n, p_1 + r_{n-1}, \dots, p_{n-1} + r_1)$.

Revenu maximum : $r_n = \max_{i \in \mathbb{N}}(p_i + r_{n-i})$

La valeur de i qui maximise r_n est inconnue: il faut tester toutes les valeurs

Décomposition du problème initial en 2 sous problèmes du même type de taille plus petite, indépendants \rightarrow instances plus petites du problème initial

Solution optimale global: celle qui maximise chacun des 2 sous problèmes

Autres manière d'organiser la structure:

- Structure récursive : découpe = tronçon de gauche de longueur i ne pouvant être découpé + tronçon pouvant être découpé de longueur $n - i$
- Solution sans coupe
 - Le tronçon de coupe est de taille $i = n$, de revenu p_n
 - le revenu du tronçon de droite restant de taille 0 est $r_0 = 0$

Nouvelle expression de l'optimisation de la coupe :

$$r_n = \max_{0 \leq i \leq n} (p_i + r_{n-i})$$

Implémentation utilisant la récursivité (cf. présentations) \rightarrow complexité temporelle : $O(2^n)$

Optimisation de la récursivité par la programmation dynamique

Fonction récursive inefficace: plusieurs évaluations identiques \rightarrow chevauchements des sous problèmes

Idée: mémoriser les résultats pour ne les évaluer qu'une seule fois

- consulter les résultats mémorisés
- Si évaluation déjà produite, la renvoyer
- Sinon, procéder à l'évaluation

Compromis temps/mémoire

- Programmation dynamique plus coûteuse en mémoire mais gain temporel
- Recherche d'une complexité polynomiale plutôt qu'exponentielle
 - Coût polynomial si chacun des sous problèmes distincts se résout en temps polynomial par rapport à la taille des données d'entrée : $O(n^\alpha)$.

Idée: mémoriser les résultats pour ne les évaluer qu'une seule fois

- approche descendante : mémoïsation
- approche ascendante

Approche descendante

- Écriture de la fonction récursive
- Modification
 - Introduction d'une structure permettant de sauvegarder les évaluations

- structure utilisée : tableau ou table de hachage
- Vérification si le sous problème est déjà évaluée avant les appels récursifs

Approche ascendante:

- Associée à une notion de taille dans les sous problèmes
- Tri des sous problèmes par taille
- Résolution des sous problèmes en commençant par les plus petits

Fonction CoupeBarreMemo(p,n):

Entrée: p, le tableau des revenus et n taille de la barre (entier)

Sortie: q, revenu maximum (entier)

Début

memo = CreerTableau(dimension: n, valeurs: -1)

Fonction Aux(p, n, memo):

Début

Si memo[n] >= 0 alors

Retourner memo[n]

Fin Si

Si n = 0 alors

q = 0

Sinon

q = -1

Pour i allant de 1 à n faire

q = max(q, p[i] + Aux(p, n-i, memo))

Fin Pour

Fin Si

memo[n] = q

Retourner q

Fin

Retourner Aux(p, n, memo)

Fin

Ordre des appels récursifs pour $n = 4$:

$4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

donc programmation dynamique *descendante*

Fonction CoupeBarreAsc(p, n)

Entrée: p, tableau des revenus et n, taille de la barre (entier)

Sortie: q, revenu maximum (entier)

```

Début
  asc = CreerTableau(dimension : n + 1, valeurs : -1)
  asc[1] = 0

  Pour j allant de 2 à n + 1 faire
    q = -1

    Pour i allant de 1 à j - 1 faire
      q = max(q, p[i] + asc[j-i])
    Fin Pour

    asc[j] = q
  Fin Pour

  Retourner asc[n]
Fin

```

III. Principe de la programmation dynamique

Historique

Programmation dynamique → planification de tâches (Richard Bellman 1953)

Résolution de problèmes d'optimisation

Cadre d'utilisation

Chevauchement de sous problèmes: évaluation redondante (plusieurs appels d'une fonction avec les mêmes arguments)

Ordre total → possibilité d'ordonner les sous problèmes

La solution d'un problème se calcule à partir des solutions des sous problèmes

→ Optimisation d'une programmation récursive

Conception

Définition des sous problèmes

Identifier la relation de récurrence entre les sous problèmes

Nécessité de comprendre les interdépendances entre les sous problèmes

En déduire un algorithme récursif avec approche ascendante et descendante

Méthode ascendante et descendante

Résoudre le sous problème original à partir des solutions des sous problèmes

Nécessité de comprendre les interdépendances entre les sous problèmes

Arbre des appels récursifs → identification des chevauchements

Mise en place du graphe des sous problèmes:

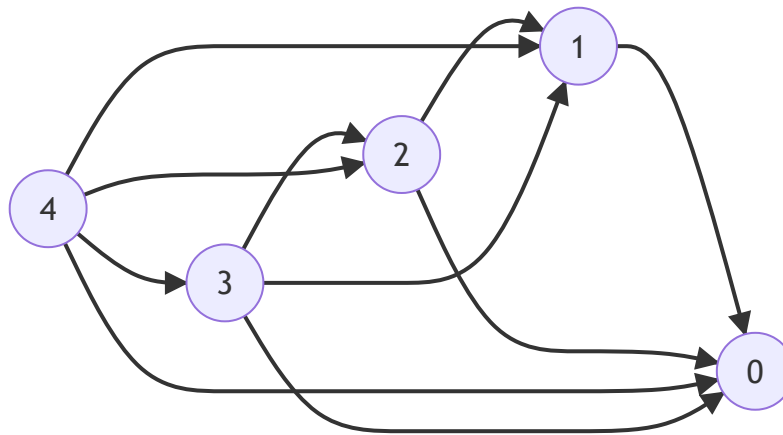
Graph orienté

- sommet pour chaque sous problème
 - arc orienté allant du sous problème x au sous problème y
 - solution optimale de x dépend de la solution optimale de y

Taille du graph donne une indication sur le temps d'exécution

Chaque sous problème n'est résolu qu'une fois → somme des temps

Exemple: découpe de barre avec $n = 4$



Méthode ascendante:

Résolution des sous problèmes les plus petits en premier

Nécessité d'un ordre total sur l'ensemble des sous problèmes → sous problème traité lorsque les sous problèmes dont il dépend ont été résolus

Méthode descendante (mémoïsation):

Mémoïser: Conserver les résultats à la fin de l'exécution d'une fonction le résultat associé aux arguments d'appels pour éviter d'avoir à recalculer

Parcours en profondeur du graph de sous problèmes

Complexité

Temporelle \rightarrow polynomiale n^α

Nombre de sous problèmes à évaluer

Temps nécessaire pour évaluer chaque sous problème

nombre de sous problèmes \times complexité par sous problèmes (sans compter les appels récursifs)

Spatiale

Nombre de sous problèmes dont il faut garder la trace à chaque étape

Bilan

Exploration exhaustive \rightarrow Force brute

Détermination de toutes les solutions

Choix d'une des solutions

Décomposition en sous problèmes

- Diviser pour régner
 - Décomposer en sous problèmes
 - Instances plus petites du problème initial
 - Obtention de sous problèmes triviaux
 - Combinaison des solutions triviales
- Gloutons
 - Choix d'une solution optimale à un problème élémentaire
 - Incorporation progressive des solutions
- Programmation dynamique
 - Décomposer en sous problèmes
 - Instances plus petites du problème initial
 - Chevauchement des sous problèmes
 - Ordre total parmi les sous problèmes
 - Mémorisation des résolutions