Matière: Algorithmique Avancée et langage C

Optimisation et Méthodes de résolution

Enseignant(e)s: **Feyrouz HAMDAOUI & Sofiane Ben Ahmed**

Niveau: **3ème Génie Informatique**

2024/2025 SU2

Plan

- Introduction
- Diviser pour régner
- Approche gloutonne
- Programmation dynamique

Introduction

- Quelques approches génériques pour aborder la résolution d'un problème :

➢ **Approche par force brute :** résoudre directement le problème, à partir de sa définition ou par une recherche exhaustive

➢ **Diviser pour régner :** diviser le problème en sous-problèmes, les résoudre, fusionner les solutions pour obtenir une solution au problème original

Introduction

- **Programmation dynamique :** obtenir la solution optimale à un problème en combinant des solutions optimales à des sous-problèmes similaires plus petits et se chevauchant
- **Approche gloutonne :** construire la solution incrémentalement, en optimisant de manière aveugle un critère local
- **Heuristique:** Quand le glouton ne donne pas toujours une solution optimal c'est une méthode heuristique

Plan

- Introduction
- Diviser pour régner
- Approche gloutonne
- Programmation dynamique

Diviser pour régner (1)
Principe général

- Si le problème est trivial, on le résout directement
- Sinon :

1. Diviser le problème en sous-problèmes de taille inférieure (Diviser)
2. Résoudre récursivement ces sous-problèmes (Régner)
3. Fusionner les solutions aux sous-problèmes pour produire une solution au problème original

Diviser pour régner (2)
Exemples

- Merge sort :
  1. Diviser : Couper le tableau en deux sous-tableaux de même taille
  2. Régner : Trier récursivement les deux sous-tableaux
  3. Fusionner : fusionner les deux sous-tableaux

  Complexité : $\Theta(n \log n)$ (force brute : $\Theta(n^2)$)
- Quicksort :
  1. Diviser : Partionner le tableau selon le pivot
  2. Régner : Trier récursivement les deux sous-tableaux
  3. Fusionner : /

  Complexité en moyenne : $\Theta(n \log n)$ (force brute : $\Theta(n^2)$)
- Recherche binaire (dichotomique) :
  1. Diviser : Contrôler l'élement central du tableau
  2. Régner : Chercher récursivement dans un des sous-tableaux
  3. Fusionner : trivial

  Complexité : $O(\log n)$ (force brute : $O(n)$)

Diviser pour régner (3) Complexité

- Souvent la complexité en temps d'un algorithme diviser-pour-régner sur une instance de taille n peut s'écrire comme:

$$T(n) = bT(n/b) + g(n)$$

- où g(n) est le temps prit pour casser et reconstruire. Si on peut montrer que $g(n) \in \Theta(n^k)$ pour un certain alors le théorème vu dans le chapitre précédent nous donne automatiquement la complexité de l'algorithme.

Plan

- Introduction
- Diviser pour régner
- Approche gloutonne
- Programmation dynamique

Approche Gloutonne (1)
principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le meilleur choix local
- Pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix qui semble le meilleur à ce moment est effectué
- Progression descendante : choix puis résolution d'un problème plus petit

Approche Gloutonne (2)
Stratégie Gloutonne

- **Propriété du choix glouton :** Il existe toujours une solution optimale commençant par un choix glouton.
- **Propriété de sous-structure optimale :** trouver une solution optimale contenant le premier choix glouton se réduit à trouver une solution optimale pour un sous-problème de même nature.

Approche Gloutonne (3)
exemple: location d'un camion

- Un unique véhicule à louer et maximiser le nombre de clients satisfaits (autres fonctions possibles à optimiser)
- E ensemble de demandes ; chaque demande ei est caractérisée par une date de début d_i et une date de fin f_i.
- Les demandes e_i et e_j sont compatibles ssi

$$]d_i, f_i[\cap]d_j, f_j[= \emptyset.$$

- On cherche un sous-ensemble maximal de E de demandes 2 à 2 compatibles.

Approche Gloutonne (4)
exemple: location d'un camion

Exemple

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Algorithme glouton : trier selon critère et satisfaire les demandes par ordre croissant.

Quel critère ?

- durée ? NON
- date de début ? NON
- date de fin ? OUI

Approche Gloutonne (5)
exemple: solution optimale

Demandes :

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Etape 1 : tri selon la date de fin

i	4	2	3	1	5	7	6	9	8
d_i	1	2	4	1	5	9	8	11	13
f_i	3	5	7	8	9	10	11	14	16

Etape 2 : satisfaire les demandes par ordre croissant

i	4	2	3	1	5	7	6	9	8
d_i	1	2	4	1	5	9	8	11	13
f_i	3	5	7	8	9	10	11	14	16

Solution : demandes $4, 3, 7, 9$.

Approche Gloutonne (6)
exemple: variantes

E trié selon dates de fin : $f_1 \leq f_2 \leq \ldots \leq f_n$
E, d, f variables globales
(Convention : $f_0 := 0$.)

Récursive

Fonction R(i)

```
1 m = i + 1;
2 tant que m ≤ n et d_m < f_i faire
3 |   m = m + 1;
4 si m ≤ n alors
5 |   return R(m)
6 sinon
7 |   return ∅
```

Itérative

Fonction P

```
1 s_1 = e_1;
2 k = 1;
3 pour i allant de 2 à n faire
4 |   si d_i ≥ f_k alors
5 |   |   s_{k+1} = e_i;
6 |   |   k = k + 1;
7 return (s_1, ..., s_k)
```

Appel $R(0)$

Complexité O(n log n) (tri des n dates de fin)

Approche Gloutonne (7)
exemple: preuve de l'algorithme

Théorème : Le résultat de l'algorithme est optimal.

1. Il existe une solution optimale qui commence par e_1 : soit A une sol. opt., et soit e_{a_1} le premier client ; si $e_{a_1} \neq e_1$, alors $A - e_{a_1} + e_1$ est aussi une sol. opt.
2. Le problème se ramène à trouver une solution optimale d'éléments de E compatibles avec e_1. Donc si A est une solution optimale pour E, alors $A' = A - e_1$ est une solution optimale pour $E' = \{e_i; d_i \geq f_1\}$. En effet si l'on pouvait trouver B' une solution optimale pour E' contenant plus de clients que A', alors ajouter e_1 à B' offrirait une solution optimale pour E contenant plus de clients que A, ce qui contredirait l'hypothèse que A est optimale.

Plan

- Introduction
- Diviser pour régner
- Approche gloutonne
- Programmation dynamique

Programmation dynamique (1)

- inventée par Bellman (~ 1954) pour résoudre des pb de chemins optimaux (longueur max. ou min.)
- une méthode de construction d'algorithme très utilisée en optimisation combinatoire (! Recherche de solution optimale dans un ensemble fini de solutions mais très grand).
- Il s'agit d'une méthode d'énumération implicite : on retient ou rejette des sous-ensembles de solutions mais on ne construit pas toutes les solutions. On rejette certaines solutions sans les avoir construites explicitement si elles appartiennent à un sous-ensemble qui n'est pas intéressant.

Programmation dynamique (2)

- La programmation dynamique appliquée à un problème donné, consiste à trouver une formulation récursive du problème.
- En procédant ensuite à un découpage étape par étape, on obtient une formule de récurrence.
- La programmation dynamique s'applique aux problèmes d'optimisation qui peuvent se décomposer en sous-problèmes de même nature, et qui possèdent les deux propriétés suivantes :

➢ *Sous-structure optimale :* on peut calculer la solution d'un problème de taille n à partir de la solution de sous-problèmes de taille inférieure

➢ *Chevauchement des sous-problèmes :* Certains sous-problèmes distincts partagent une partie de leurs sous-problèmes

Programmation dynamique (3)
Exemple Fibonacci

- Nous avons vu que la complexité de cet algorithme en récursif est en exponentielle. La raison de cette inefficacité est due à la multiplicité de calcul d'un même nombre.
- La clef à une solution plus efficace est d'éviter la multiplicité de résolution du même sous problème. On améliore de loin la complexité temporelle si, on se contente de ne stocker que les deux dernières valeurs

Programmation dynamique (4)
Exemple Fibonacci

```
FIBONACCI-ITER(n)
 1  if n ≤ 1
 2        return n
 3  else
 4        pprev = 0
 5        prev = 1
 6        for i = 2 to n
 7              f = prev + pprev
 8              pprev = prev
 9              prev = f
10        return f
```

- Complexité O(n)

Bibliographie (1)

- Anthony Labarre « Complexité algorithmique ». Structures de données et algorithmes fondamentaux. Université Paris-Est Marne la Vallée;
- Christine Solnon «Validation théorique et évaluation expérimentale d'algorithmes ». INSA de Lyon - 5IF, 2015;
- Eric Tannier «Les mathématiques du calcul ». Module Maths Discrètes, INSA, Univ Lyon 1, 2015-2016;
- Mme Aroussi « Chapitre III NP-Complétude ». Mastère GSI (Génie des Systèmes Informatiques) Faculté des Sciences, Université Blida 1, 2015/2016 SU1.
- Sylvain Perifel, «Introduction à la calculabilité et à la complexité». Cours de M1, premier semestre 2010–2011, Université Paris Diderot – Paris 7.
- Christine Solnon «Validation théorique et évaluation expérimentale d'algorithmes », INSA de Lyon - 5IF, 2015

Bibliographie (2)

- Gilles Schaeffer, «Réduction et NP-complétude». Cours Algorithmique avancée. Ingénieur 3A/Master 1, Ecole polytechnique, Université Paris-Saclay, 2014/2015.
- Christophe PAUL, « Théorie de la NP-Complétude (1) ». Cours Complexité avancée, Master 2 Informatique, Université Montpellier 2, septembre 2007
- Pierre Geurts, «Partie 6 Résolution de problèmes » Université de Liège, Institut Montefiore, 2011.

Bibliographie (3)

- J.-F. Scheid «Chapitre 7 : Programmation dynamique ». Graphes et RO – TELECOM Nancy 2A.
- Sylvie Hamel «Algorithmes diviser-pour-régner » Université de Montréal.
- Ana Busic «Cours 7 et 8: Algorithmes Gloutons ». Cours Conception d'algorithmes et applications.
- Jean-Charles Régin «Résolution de problèmes ». Cours Licence Informatique 2ème année, 2011