IFT2125 - Introduction à l'algorithmique

Programmation dynamique (B&B chapitre 8)

Pierre McKenzie

DIRO, Université de Montréal

Automne 2017

Two inefficient uses of divide-and-conquer

Deux usages inefficaces de diviser-pour-régner

1) Calcul des coefficients binomiaux, B&B Section 8.1.1

Calculation of binomial coefficients

binomial coefficients

COEFFICIENT BINOMIAL

DONNÉE: integer entiers $0 \leq k \leq n$

Let

CALCULER: $\binom{n}{k}$

Calculate

Here is a divide-for-conquer algo for Binomial Coefficient:

Voici un algo diviser-pour-régner pour COEFFICIENT BINOMIAL :

function $C(n, k)$

  if $k = 0$ **or** $k = n$ **then return** 1

  else return $C(n-1, k-1) + C(n-1, k)$

Deux usages inefficaces de diviser-pour-régner

2) Probabilité de l'emporter en série mondiale, B&B Section 8.1.2

2) Probability of winning a world series, B & B Section 8.1.2

world series
SÉRIE MONDIALE

probability $0 \leq p \leq 1$ that A beats B during a single match, integers $n > 0$ and $0 \leq i, j < i + j < 2n$

DONNÉE: probabilité $0 \leq p \leq 1$ que A batte B lors d'un seul match, entiers $n > 0$ et $0 \leq i, j < i + j < 2n$

Let

CALCULER: probabilité $P(i, j)$ que A gagne n matchs avant B, sachant qu'il manque à A i victoires et à B j victoires

calculate

probability P (i, j) that A wins n matches before B, knowing that A i wins and B j wins are missing

Quel serait un algo diviser-pour-régner ?

What would an algo divide-to-conquer?

Deux usages inefficaces de diviser-pour-régner

2) Probabilité de l'emporter en série mondiale, B&B Section 8.1.2

2) Probability of winning a world series, B & B Section 8.1.2

world series
SÉRIE MONDIALE

probability $0 \leq p \leq 1$ that A beats B during a single match, integers $n > 0$ and $0 \leq i, j < i + j < 2n$

DONNÉE: probabilité $0 \leq p \leq 1$ que A batte B lors d'un seul match, entiers $n > 0$ et $0 \leq i, j < i + j < 2n$

Let

CALCULER: probabilité $P(i, j)$ que A gagne n matchs avant B, sachant qu'il manque à A i victoires et à B j victoires

calculate

probability P (i, j) that A wins n matches before B, knowing that A i wins and B j wins are missing

Quel serait un algo diviser-pour-régner ?

What would an algo divide-to-conquer?

function $P(i, j)$
  if $i = 0$ **then return** 1
  else if $j = 0$ **then return** 0
  else return $pP(i-1, j)+qP(i, j-1)$

Much better solution for World Series

Bien meilleure solution pour Série mondiale

Par “programmation dynamique”

By "dynamic programming"

$$\underbrace{P(i,j) = pP(i-1,j) + qP(i,j-1)}$$

suggère de remplir un tableau une diagonale à la fois de haut en bas

suggests filling a chart one diagonal at a time from top to bottom

world series by Dynamic programming

Série mondiale par programmation dynamique

B&B Section 8.1.2

```
function series(n, p)
    array P[0..n, 0..n]
    q ← 1 − p
    {Fill from top left to main diagonal}
    for s ← 1 to n do
        P[0, s] ← 1;  P[s, 0] ← 0
        for k ← 1 to s − 1 do
            P[k, s − k] ← pP[k − 1, s − k] + qP[k, s − k − 1]
    {Fill from below main diagonal to bottom right}
    for s ← 1 to n do
        for k ← 0 to n − s do
            P[s + k, n − k] ← pP[s + k − 1, n − k] + qP[s + k, n − k − 1]
    return P[n, n]
```

Money change

Rendre la monnaie

B&B Section 8.2

Reminder: although effective, the greedy approach sometimes missed a solution

- Rappel : bien qu'efficace, l'approche vorace parfois ratait une solution

The dynamic programming approach works, whatever the coin values

- L'approche programmation dynamique fonctionne, quelles que soient les valeurs des pièces :
  - démo du 15 novembre. demo of November 15th.
  - L'idée : the idea :
    $c[i,j]$ = nombre min de pièces pour rendre j en i dénominations
    min number of pieces to make j in i denominations
  - Alors : then :
    $$\underbrace{c[i,j] = \min(c[i-1,j], 1 + c[i, j - \text{dénom}[i]])}$$
    suggère à nouveau de remplir par diagonales, de haut en bas
    again suggests filling diagonally, from top to bottom

Principle of optimality

Principe d'optimalité

B&B Section 8.3

Dynamic programming is to be preferred when

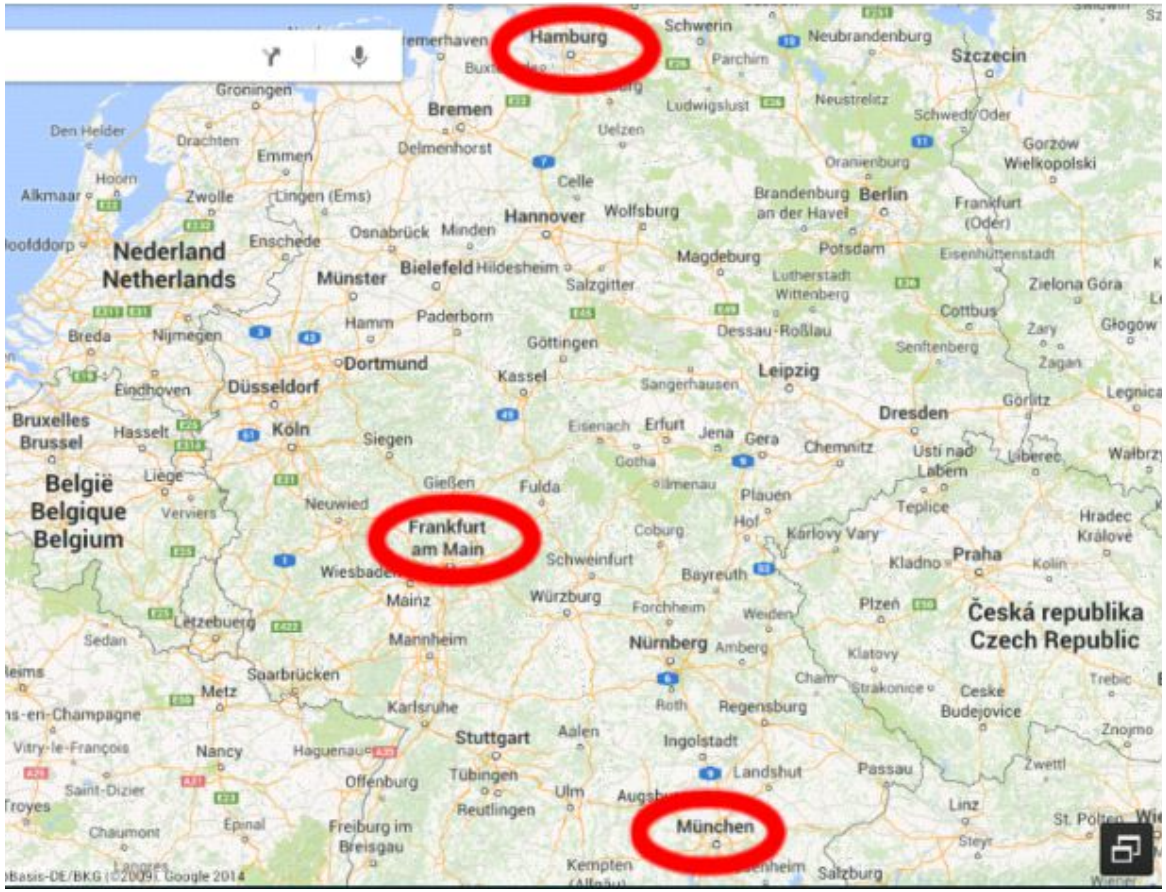
La programmation dynamique est à privilégier lorsque

- the problem to solve breaks down into similar sub-problems
  le problème à résoudre se décompose en sous-problèmes semblables
- these sub-problems tend to overlap
  ces sous-problèmes ont tendance à se chevaucher
- le principe d'optimalité s'applique : chaque sous-séquence d'une séquence de choix optimale est optimale

  the principle of optimality applies: each subsequence of an optimal choice sequence is optimal

Principe d'optimalité

Exemples



shortest way: yes

- chemin le plus court : oui
- chemin le plus rapide : non fastest way: no
- chemin simple le plus long : non, ex : graphe complet longest single path: no, ex: full graph

Sac à dos

B&B Section 8.4

backtrack

SAC À DOS

DONNÉE: capacité $W \in \mathbb{R}^{\geq 0}$ et objets $1, 2, \ldots, n$ de poids $w_1, \ldots, w_n \in \mathbb{R}^{\geq 0}$ et de valeurs $v_1, \ldots, v_n \in \mathbb{R}^{\geq 0}$

capacity W ∈ R ≥ 0 and objects 1,2, ..., n of weight

Let w1, ..., wn ∈ R ≥ 0 and values v1, ..., vn ∈ R ≥ 0

CALCULER: objets de valeur maximale et de poids n'excédant pas W

Calculate objects of maximum value and weight not exceeding W

- Rappel : bien qu'efficace, l'approche vorace ne parvenait à résoudre que SAC À DOS FRACTIONNAIRE

  Reminder: Although effective, the greedy approach could only solve Fractional knapsack

- L'approche programmation dynamique résoud SAC À DOS Comment ?

  The dynamic programming approach resolves Backpack How?

knapsack
Sac à dos

(suite)
(cont.)

The idea
- L'idée :
  $V[i,j] =$ max value with objects {1,2, ..., i} and capacity
  valeur max avec objets $\{1, 2, \ldots, i\}$ et capacité $\leq j$
- On cherche : $V[n, W]$
  We search

Then :
- Alors :

knapsack

Sac à dos

(suite)

(cont.)

The idea

- L'idée :
  max value with objects {1,2, ..., i} and capacity
  $V[i,j] =$ valeur max avec objets $\{1,2,\ldots,i\}$ et capacité $\leq j$
- On cherche : $V[n,W]$
  We search

Then :

- Alors :

$$\underbrace{V[i,j] = \max(V[i-1,j], v_i + V[i-1,j-w_i])}$$

suffit donc de remplir ligne par ligne, de haut en bas

So just fill in line by line, from top to bottom

- L'algorithme détaillé coûtera $\Theta(nW)$ opérations (accès au tableau, sommes, comparaisons)

  The detailed algorithm will cost Θ (nW) operations (table access, sums, comparisons)

Plus courts chemins

B&B Section 8.5

Shortest paths

PLUS COURTS CHEMINS

graph (N, A) with non-negative lengths (or ∞) at arcs

DONNÉE: graphe (N, A) avec longueurs non négatives (ou ∞) aux arcs

CALCULER: chemins les plus courts de chaque sommet i à chaque sommet j

CALCULATE: shortest paths from each vertex i to each vertex j

Reminder: Dijkstra (voracious) calculated in Θ (| N | 2) the distances from one vertex fixed to all the others,

- Rappel : Dijkstra (vorace) calculait en $\Theta(|N|^2)$ les distances d'un sommet fixé à tous les autres,
  donc résoud PLUS COURTS CHEMINS en $\Theta(|N|^3)$
  So solve shorter paths in Θ (| N | 3)

- Floyd (programmation dynamique) fournit une solution alternative
  Comment ?
  Floyd (dynamic programming) provides an alternative solution How?

Plus courts chemins

(suite)
(cont.)

- L'idée : The idea:
  Dk [i, j] = pcc length from i to j restricted to vertices ≤ k
  $D_k[i,j] =$ longueur du pcc de i à j restreint aux sommets $\leq k$
- On cherche : les n^2 entrées de D_n
  We are looking for: the n^2 entries of Dn
- so
  Alors $\forall i,j : D_k[i,j] = \min(\ D_{k-1}[i,j]\ ,$

Plus courts chemins

(suite)
(cont.)

- L'idée : $D_k[i,j] =$ longueur du pcc de i à j restreint aux sommets $\leq k$

  The idea:
  Dk [i, j] = pcc length from i to j restricted to vertices ≤ k

- On cherche : les n^2 entrées de D_n

  We are looking for: the n^2 entries of Dn

- Alors $\forall i,j : D_k[i,j] = \min(\ D_{k-1}[i,j]\ ,\ D_{k-1}[i,k] + D_{k-1}[k,j]\ )$

  so

- L'algo :

  the algo

Plus courts chemins

(suite)
(cont.)

- L'idée : The idea:
  Dk [i, j] = pcc length from i to j restricted to vertices ≤ k
  $D_k[i,j] =$ longueur du pcc de i à j restreint aux sommets $\leq k$
- On cherche : les n^2 entrées de D_n
  We are looking for: the n^2 entries of Dn

so
- Alors $\forall i,j : D_k[i,j] = \min(\ D_{k-1}[i,j]\ ,\ D_{k-1}[i,k] + D_{k-1}[k,j]\ )$

the algo
- L'algo : fill D1, then D2, then D3, then $\cdots$, then Dn
  - remplir D_1, puis D_2, puis D_3, puis $\cdots$, puis D_n
  - coup de bol : un seul tableau peut stocker $D_1, D_2, \ldots, D_n$
    coup de bol: only one table can store D1, D2, ..., Dn

Plus courts chemins

(suite)
(cont.)

```
function Floyd(L[1..n, 1..n]): array [1..n, 1..n]
    array D[1..n, 1..n]
    D ← L
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min(D[i, j], D[i, k] + D[k, j])
    return D
```

Produit chaîné de matrices

B&B Section 8.6

Scheduling of matrix products

Ordonnancement de produits matriciels

matrices M1, M2, . . . , Mn of compatible dimensions

DONNÉE: matrices $M_1, M_2, \ldots, M_n$ de dimensions compatibles

CALCULER: l'ordre optimal dans lequel effectuer les produits pour obtenir $\prod_{i=1}^{n} M_i$

the optimal order in which to perform the products to get prod_{i=1,...,n}{M_i}

En classe.

In class.

```
def matrices(D):
    m = len(D)-1
    T = [ [0]*m for i in range(m)]
    P = [[-1]*m for i in range(m)]

    for i in reversed(range(m)):
        for j in range(i+1, m):
            c, pos = float("inf"), -1

            for k in range(i, j):
                d = T[i][k] + T[k+1][j] + D[i]*D[k+1]*D[j+1]

                if (d < c):
                    c, pos = d, k

            T[i][j] = c
            P[i][j] = pos

    return P
```

Transform into recursion ...

Transformer en récursion...

B&B Section 8.7

idea

Idée :

Instead of an array like T [i, j], a recursive function fT [i, j]

- Au lieu d'un tableau comme $T[i, j]$, une fonction récursive $fT[i, j]$
- Au lieu d'accéder à $T[i, j]$, appeler récursivement $fT[i, j]$

Instead of accessing T [i, j], recursively call fT [i, j]

Why?

Pourquoi ?

- Pour remplir "top down" au lieu de "bottom up"

To fill "top down" instead of "bottom up"

the trouble?

Le malheur ?

- Même inconvénient de recoupements abusifs d'exemplaires que les usages inefficaces de diviser-pour-régner !

Even disadvantage of abusive cross-checks of copies as ineffective uses of divide-and-conquer!

and use memory functions
...et employer les fonctions à mémoire

B&B Section 8.8

the idea
L'idée :

An array mtab which stores the values fT [i, j] already calculated

- Un tableau mtab qui mémorise les valeurs $fT[i,j]$ déjà calculées
- Avant de recalculer $fT[i,j]$, vérifier mtab[i,j]

before recalculating fT[i, j], check mtab[i,j]

- On récupère (presque) l'efficacité de la programmation dynamique

We recover (almost) the efficiency of dynamic programming

Exemple pour ORDONNANCEMENT DE PRODUITS MATRICIELS :

Example for Scheduling Matrix Products:

```
function fm-mem(i, j)
    if i = j then return 0
    if mtab[i, j] ≥ 0 then return mtab[i, j]
    m ← ∞
    for k ← i to j − 1 do
        m ← min(m, fm-mem(i, k) + fm-mem(k + 1, j)
                                + d[i − 1]d[k]d[j])
    mtab[i, j] ← m
    return m
```