CentraleSupélec

Algorithmique et complexité

TD 4/7 – Programmation dynamique

Exercices complémentaires

Remarque: Les éléments de correction fournis dans le sujet ne sont pas forcément complets. Ils sont là pour vous guider dans votre travail personnel. Nous vous invitons à rédiger une solution comme vous le feriez à l'examen et, si vous avez des questions, à vous tourner vers votre chargé de TD.

Exercice 1 : Pyramide de nombres

Une pyramide de nombres est un ensemble d'entiers quelconques empilés à la manière d'un triangle de Pascal. Dans une telle pyramide, on cherche le chemin (strictement descendant) entre le sommet et la base tel que la somme des nombres le long de ce chemin est maximum. Exemple, dans la pyramide ci-dessous, le chemin maximum vaut $3 + 7 + 4 + 9 = 23$.

```
   3
  7 4
 2 4 6
9 5 9 3
```

Question 1

Pour une pyramide de base n, combien y a t-il de chemins ? Est-il envisageable d'utiliser un algorithme dénombrant les chemins pour trouver le maximum ?

Élements de correction :

Une pyramide de base n a une hauteur n. En construisant les chemins en partant du sommet, à chaque niveau (à l'exception de la base), il est possible de prolonger un chemin en descendant vers la gauche ou la droite. Il y a donc 2^{n-1} chemins possibles. Il n'est pas envisageable de les dénombrer.

Question 2

On note x une position dans la pyramide, $v(x)$ la valeur de l'entier à cette position, $g(x)$ l'entier en dessous à gauche de x, $d(x)$ l'entier en dessous à droite et $c(x)$ le total des nombres traversés dans un chemin maximal issu de x.

En utilisant ces définitions, proposez une fonction récursive permettant de calculer la valeur $c(x)$ pour tout x.

Élements de correction :

Voici une définition possible :
- $c(x) = v(x)$ pour tout x à la base de la pyramide ;
- $c(x) = v(x) + max(c(g(x))), c(d(x)))$ pour toute autre position.

Question 3

Si l'on implémente directement cette fonction récursive, quelle serait la complexité de notre algorithme pour une pyramide de base n ?

Élements de correction :

Implémenter directement la fonction récursive, revient à parcourir tous les chemins, la complexité est donc exponentielle.

Question 4

Pour l'implémentation, on souhaite mettre en place une technique de mémoïsation, c'est à dire stocker les valeurs $c(x)$ déjà calculées. Proposez un algorithme pour cela. Quelle est sa complexité ?

La structure de données la plus simple à utiliser pour stocker la pyramide est une liste de listes.

```
[
[3],
[7,4],
[2,4,6],
[9,5,9,3]
]
```

Élements de correction :

Complexité en $\mathcal{O}(n^2)$.

```
pyramide = [
    [3],
    [7, 4],
    [2, 4, 6],
    [9, 5, 9, 3]
]


def c(pyramide):
    mem = {}

    def c_rec(i, j):
        if i == len(pyramide) - 1:
            return pyramide[i][j]
        else:
            if (i + 1, j) not in mem:
                mem[(i + 1, j)] = c_rec(i + 1, j)

            if (i + 1, j + 1) not in mem:
                mem[(i + 1, j + 1)] = c_rec(i + 1, j + 1)

            return pyramide[i][j] + max(mem[(i + 1, j)], mem[(i + 1, j + 1)])

    return c_rec(0, 0)


print(c(pyramide))
```

Exercice 2 : Problème du voyageur de commerce

Étant donné un ensemble S de sites et une fonction totale $d : S \times S \to \mathbb{N}$ des distances entre sites, le problème du voyageur de commerce consiste, en partant d'un site donné, à trouver un cycle de distance minimale passant une et une seule fois par chacun des sites, et retournant au site de départ.

Sans perte de généralité, on identifie les n sites de S par les entiers $\{1, 2, \ldots, n\}$, et on suppose que le site de départ est 1.

Pour $S' \subseteq \{2, \ldots, n\}$ et $x \in S \setminus S'$, on note $D(S', x)$ la longueur du plus petit chemin partant du site de départ (de valeur 1), visitant tous les sites de S' (une et une seule fois) et terminant en x. Il est clair que $D(\emptyset, x) = d(1, x)$ et que $D(\{2, \ldots, n\}, 1)$ correspond à la solution optimale de ce problème.

Question 1

Donnez une formule de récurrence calculant $D(S', x)$.

Élements de correction :

$$D(S', x) = min_{y \in S}(D(S' \setminus \{y\}, y) + d(y, x))$$

Question 2

Écrivez une fonction récursive $D(S', x)$ (en Python ou en pseudo-code) appliquant cette formule de récurrence.

Élements de correction :

```
  def D(S,x):
    if S == []:
        return d(1,x)
    minD = sys.maxsize
    for i in range(len(S)):
        y = S[i]
        S.remove(y)
        minD = min(minD, D(S,y) + d(y,x))
        S.insert(i,y)
    return minD
```

Ci-dessous un jeu de données pour pouvoir tester le code précédent ainsi que le code Question 6.

```
####### Données à tester ##########

import random
import math

def genere_carte(lx, ly, n):
    carte = []
    for i in range(n):
        while True:
            p = (random.randint(0, lx-1), random.randint(0, ly-1))
            if p in carte:
                continue
            carte.append(p)
            break
    return carte

def distance(carte, i, j):
    return round(math.hypot(carte[j][0]-carte[i][0], carte[j][1]-carte[i][1]))

def generer_matrice_distances(carte):
    return [[distance(carte, i, j) for j in range(len(carte))] for i in range(len(carte))]

matrice_distances = generer_matrice_distances(carte)

def d(x, y):
  return matrice_distances[x-1][y-1]
```

Question 3

Quelle est la complexité de la fonction $D(S', x)$ en fonction de la taille de S' ?

Élements de correction :

On définit $C(n)$ la complexité de $D(S', x)$ avec n la taille de S'. On a $C(0) = O(1)$ et $C(n) = n \times (C(n-1) + O(1))$.
On en déduit que $C(n) = O(n!)$

Question 4

En déroulant l'exécution du calcul $D(\{x_1, x_2, x_3, x_4, \ldots, x_m\}, x)$ sous forme d'un arbre des appels récursifs, que constatez-vous ?

Élements de correction :

Dans l'arbre des appels, on retrouve des calculs redondants (même calcul sur plusieurs branches). par exemple les 2 branches suivantes aboutissent au même calcul :

- $D(\{x_1, x_2, x_3, x_4, \ldots, x_m\}, x) \longrightarrow D(\{x_2, x_3, x_4, \ldots, x_m\}, x_1) \longrightarrow D(\{x_3, x_4 \ldots, x_m\}, x_2) \longrightarrow D(\{x_4 \ldots, x_m\}, x_3)$
- $D(\{x_1, x_2, x_3, x_4, \ldots, x_m\}, x) \longrightarrow D(\{x_1, x_3, x_4, \ldots, x_m\}, x_2) \longrightarrow D(\{x_3, x_4 \ldots, x_m\}, x_1) \longrightarrow D(\{x_4 \ldots, x_m\}, x_3)$

Question 5

En appliquant les principes de la programmation dynamique, proposez une table pour sauvegarder les calculs intermédiaires. Précisez son contenu et sa dimension.

Élements de correction :

Pour éviter les calculs redondants lors de l'exécution de $D(\{2, \ldots, n\}, 1)$, on sauvegarde les résultats intermédiaires de $D(S', x)$ avec $S' \subset \{2, \ldots, n\}$ et $x' \in \{2, \ldots, n\} - S'$.

On a besoin d'une table à 2 dimensions de taille $(n-1) \times (2^{n-1} - 1)$.

- une dimension (par exemple les colonnes) correspond aux sous-ensembles S' de $\{2, \ldots, n\}$ de taille $2^{n-1}-1$ car on ne considère pas l'ensemble lui-même $\{2, \ldots, n\}$
- l'autre dimension (les lignes) de taille $n-1$ correspond aux différentes valeurs de x.

On peut constater que pour chaque ligne (c-à-d chaque valeur de x) seule la moitié sera remplie à cause de la condition $x \notin S'$. Cela pourrait servir à un calcul précis de la complexité dans la suite.

Question 6

Écrivez en Python ou en pseudo-code un algorithme de résolution du problème du voyageur de commerce.

Élements de correction :

```
# cette fonction sauvegarde dans la table
# une table T est un dictionnaire (clés : les x) de dictionnaires (clés: les sous-ensembles de S)
def save(S, x, T, v):
  if not x in T:
    T[x] = {}
  # on transforme la liste S en un String hashable pour l'utiliser comme clé du dico
  T[x][str(S)] = v
  return v


def D_dyn(S, x, T = {}):
  # verifier la présence du calcul dans la table
  if x in T and str(S) in T[x]:
    return T[x][str(S)]

  # sinon on applique la formule de récurence (partie inchangée)
  # on sauvegarde dans la table avant de retourner
  if S == []:
    return save(S, x, T, d(1, x))

  minD = sys.maxsize
  for i in range(len(S)):
    y = S[i]
    S.remove(y)
    minD = min(minD, D_dyn(S, y, T) + d(y, x))
    S.insert(i, y)

  return save(S, x, T, minD)
```

Question 7

Donnez sa complexité et comparez avec la version récursive.

Élements de correction :

La complexité de cet algorithme correspond à la taille de la table ($\approx n \times 2^n$) multiplié par le temps de calcul d'une case en $O(n)$. On a donc une complexité $O(n^2 \times 2^n)$ qui est exponentielle mais bien moindre que $O(n!)$ la complexité de l'algo récursif.