REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi

Département MI
Faculté des sciences exactes et sciences de la nature et de la vie
Année : 2024-2025

LE : 19/01/2025
Niveau : Progressé M1 AD S1
Session normale

Examen en méthodes algorithmiques

Questionnaire à Choix Multiples:((6Pts))

1 : b ; 2 :b ; 3 :b ; 4 :b ; 5 : C ;6 : A .

Exercice 01

Solution gloutonne (3Pts)

```
def algo_glouton(matrice):
    n = len(matrice)
    i, j = 0, 0
    total_risque = matrice[i][j]

    while i < n - 1 or j < n - 1:
        if i == n - 1
            j += 1
        elif j == n - 1
            i += 1
        else:

            if matrice[i + 1][j] < matrice[i][j + 1]:
                i += 1
            else:
                j += 1
        total_risque += matrice[i][j]

    return total_risque
```

Méthode Basée sur la Programmation Dynamique(3Pts)

1. Construire une matrice de coût minimum, où chaque case représente le coût minimum pour atteindre cette position depuis la cellule (1,1).
2. Pour chaque cellule (i,j) , le coût est calculé comme le minimum du du coût pour atteindre (i−1,j) ou (i,j−1), plus le coût de la cellule (i,j).
3. Une fois la matrice remplie, la solution optimale est dans la cellule (n,n).

```
def algo_programmation_dynamique(matrice):
    n = len(matrice)
    lc = [[0] * n for _ in range(n)]
    lc[0][0] = matrice[0][0]

    for j in range(1, n):
        lc[0][j] = lc[0][j-1] + matrice[0][j]

    for i in range(1, n):
        lc[i][0] = lc[i-1][0] + matrice[i][0]

    for i in range(1, n):
        for j in range(1, n):
            lc[i][j] = min(lc[i-1][j], lc[i][j-1]) + matrice[i][j]

    return dp[n-1][n-1]
```

La **programmation dynamique** est le choix préférable si l'on souhaite une solution optimale, même si elle est plus coûteuse en termes de ressources.**(1Pts)**

Exercice 02 (7 Pts)

1. **va** une matrice utilisée pour stocker des valeurs booléennes indiquant si une somme donnée peut être atteinte en utilisant un sous-ensemble des i premiers éléments de mys.

2. Parce qu'une somme de 0 peut toujours être atteinte en sélectionnant un sous-ensemble vide, quel que soit le nombre d'éléments disponibles.

3. Cette condition vérifie si l'élément mys[i−1] peut être inclus dans le sous-ensemble pour atteindre la somme j. Si cet élément est supérieur à j, il ne peut pas être inclus.

4. Si l'algorithme retourne True, cela signifie qu'il existe un sous-ensemble des éléments de mys dont la somme est exactement égale à target.

5. L'algorithme résout le **problème de la somme de sous-ensembles** (Subset Sum Problem). Il vérifie s'il est possible de sélectionner un sous-ensemble des éléments du tableau mys dont la somme est égale à target.