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 : M1 AD S1
Session normale

Corrigé type de l'Examen en méthodes algorithmiques

Questions de cours :

1. Expliquez la différence entre la programmation dynamique et la méthode gloutonne. Donnez un exemple où la programmation dynamique est nécessaire alors que la méthode gloutonne échouerait.**(2Pts)**

   La **méthode gloutonne** choisit des décisions locales optimales sans considérer l'ensemble du problème, ce qui peut ne pas mener à la solution globale optimale. En revanche, la **programmation dynamique** résout les sous-problèmes de manière exhaustive et optimale, garantissant ainsi une solution globale optimale.La programmation dynamique, résout des problèmes en les décomposant en sous-problèmes plus petits, et en mémorisant (ou en mémorisant) les résultats de ces sous-problèmes pour éviter de les recalculer plusieurs fois.

2. Expliquez la différence fondamentale entre un algorithme déterministe et un algorithme probabiliste.**(2Pts)**

   Algorithmes Déterministes :
   Un algorithme déterministe produit toujours le même résultat pour une entrée donnée. Toutes les étapes de l'algorithme sont définies et prévisibles.

   Algorithmes Probabilistes :
   Un algorithme probabiliste intègre des choix aléatoires dans son exécution. Il peut produire des résultats différents pour une même entrée, et son comportement repose en partie sur des probabilités.

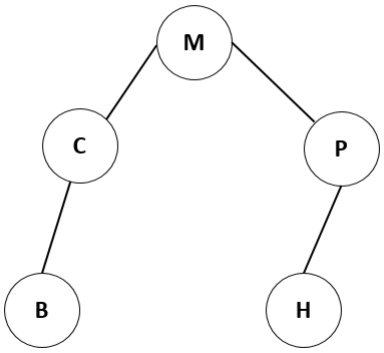
1. Complétez le tableau ci-dessous en vous basant sur les caractéristiques des algorithmes Monte Carlo et Las Vegas**(2Pts)**

Type d'algorithme	Temps d'Exécution	Résultat
Monte Carlo	Fixe	Approximatif
Las Vegas	Variable	Exact

Exercice 01 :

On affecte aux modules les noms suivants : Mathématiques (M),Physique (P), Chimie (C) , Histoire (H) et Biologie (B).

1. Représentez ce problème sous forme de graphe en indiquant clairement les sommets et les arêtes.**(1 Pts)**



2. Donnez un algorithme de backtracking pour trouver le nombre minimal de créneaux horaires nécessaires pour organiser les examens.**(6Pts)**

```
def est_compatible(module, horaires, conflits, creneau)

    for voisin in conflits[module]
        if horaires[voisin] == creneau
            return False
    return True

def backtracking(conflits, m, horaires, module)
    if module == len(conflits):
        return True

    for creneau in range(1, m + 1)
        if est_compatible(module, horaires, conflits, creneau):
            horaires[module] = creneau
            if planifier_examen(conflits, m, horaires, module + 1):
                return True

            horaires[module] = -1
def planification_examen(conflits, m)
    horaires = [-1] * len(conflits)

    if planifier_examen(conflits, m, horaires, 0):
        print("La planification des examens est possible avec les créneaux horaires
suivants :")
        for module in range(len(conflits)):
            print(f"Module {module} a été assigné au créneau horaire
{horaires[module]}")
    else:
print("Aucune solution possible avec le nombre de créneaux horaires donné.")
```

Exercice 02
(7Pts)

1. La variable x représente un tableau (ou liste) de parents pour chaque sommet. Chaque sommet commence avec lui-même comme parent, ce qui signifie qu'au départ, chaque sommet est son propre représentant dans l'ensemble des composantes connexes.

2. On initialise x[i] à i pour chaque sommet afin de représenter qu'au départ, chaque sommet est dans sa propre composante connexe, et il est son propre parent.

3. La condition if p != q vérifie si les deux sommets u et v appartiennent à deux composantes différentes. Si c'est le cas, on peut les connecter sans créer de cycle, et l'arête qui relie ces deux sommets peut être ajoutée à l'arbre couvrant minimal.

4. La ligne z.append((i, j, k)) ajoute une arête à l'arbre couvrant minimal, où i et j sont les deux sommets connectés et k est le poids de l'arête.

5. L'algorithme de Kruskal cherche à résoudre le problème de **trouver un arbre couvrant minimal** dans un graphe pondéré. Il fait cela en triant les arêtes par poids, en ajoutant progressivement les arêtes les plus légères sans créer de cycles, et en utilisant une structure de données d'union-find pour gérer les composants connexes.