TOMIC Blanc - Graphe et Complexité

Exercice 1 : Livraisons de pizzas et complexité des parcours

Une pizzeria reçoit **6 commandes** à livrer dans un quartier. Le livreur doit se déplacer entre les adresses en scooter.

Voici la liste des **commandes** et les **temps de trajet en minutes** entre chaque adresse (non symétrique à cause du sens unique des rues) :

	A	B	C	D	E	F
A	–	4	2	–	–	–
B	–	–	1	5	–	–
C	–	–	–	8	10	–
D	–	–	–	–	2	3
E	–	–	–	–	–	1
F	–	–	–	–	–	–

Partie 1 : Représentation

1. Représente ce réseau de livraisons sous forme d'une **liste d'adjacence pondérée**.

> On l'écrira sous la forme d'un dictionnaire Python où les clés sont les adresses et les valeurs sont des listes de tuples (adresse, temps de trajet). Par exemple, pour l'adresse A, on écrira `A: [(B, 4), (C, 2)]`.

2. Dessine le **graphe orienté pondéré** correspondant.
3. Quel est le **type de graphe** (orienté ? connexe ? acyclique ?) Justifie.

Partie 2 : Attribution des livreurs

La pizzeria dispose de **plusieurs livreurs**, mais chaque livreur ne peut gérer **qu'un seul itinéraire de livraison sans retour**. Pour garantir l'efficacité des tournées :

- Un livreur **ne peut livrer** deux commandes **que si aucune des deux ne dépend de l'autre** dans le trajet (c'est-à-dire qu'on ne passe pas par l'une pour atteindre l'autre).

> Par exemple, si pour aller de A à E on passe par C, alors un **même livreur ne peut pas faire A et E**.

1. En t'appuyant sur le graphe du trajet, indique **quelles commandes ne peuvent pas être livrées par le même livreur** que la commande A.
2. Combien de **livreurs minimum** sont nécessaires pour garantir que toutes les commandes soient livrées **sans conflit de parcours** ?
3. Propose une **répartition des commandes** entre les livreurs, en respectant la contrainte énoncée.
4. Justifie que ce nombre est **minimal**, c'est-à-dire que l'on ne peut pas faire avec moins de livreurs.

> Tu peux t'aider du dessin du graphe pour repérer les **chemins directs ou indirects** entre les commandes.

Partie 3 : Dijkstra et complexité

1. Applique **Dijkstra** à ce graphe pour calculer les **plus courts chemins** depuis la commande A vers toutes les autres.
2. Pour chaque commande atteignable depuis A, donne :
    - Le **temps minimal**
    - Le **chemin suivi**
3. Quelle est la **complexité temporelle** de Dijkstra dans ce graphe si :
    - (a) tu utilises une **liste d'adjacence + file de priorité**
    - (b) tu utilises une **matrice d'adjacence brute**

En bonus : Complexité et ordonnancement

1. Ce graphe représente-t-il un **ordre partiel** ? Peut-on effectuer un **tri topologique** ?
2. Que représente ce tri dans le contexte de livraison ?
3. Peut-on déterminer un **chemin de livraison optimal** visitant chaque commande une seule fois ? Quel est le **coût en complexité** d'un algorithme naïf pour le faire ?

Exercice 2 : Optimisation de pizza et complexité algorithmique

Un client veut une pizza personnalisée avec **au plus 5 garnitures**, choisies parmi celles disponibles. Chaque garniture a :

- un **prix** (en €),
- un **score de goût** (sur 10),
- certaines garnitures sont **exclues pour raisons allergiques**.

On retient les **6 garnitures autorisées** suivantes :

Garniture	Prix (€)	Score goût
Tomate	1.0	6
Fromage	2.0	9
Olives	1.5	4
Poivrons	1.2	5
Jambon	2.5	8
Basilic	0.5	3

Le client impose :

- un **maximum de 5 garnitures**,

- un **budget maximal de 10 €**,
- un **score de goût le plus élevé possible**.

Partie 1 : Algorithme glouton

1. Propose un **algorithme glouton** permettant de générer une pizza respectant les contraintes. Tu peux trier les garnitures selon leur **rapport score/prix**, puis sélectionner tant que possible.
2. Applique ton algorithme aux 6 garnitures ci-dessus.
    - Quelles garnitures sont sélectionnées ?
    - Quel est le **score total** ?
    - Quel est le **coût total** ?
3. Donne la **complexité temporelle** de cet algorithme en fonction de n (nombre de garnitures valides).

Partie 2 : Algorithme naïf (exhaustif)

1. Décris un **algorithme naïf** qui teste **toutes les combinaisons possibles** de garnitures (maximum 5 éléments), en filtrant celles qui respectent le budget.
2. Quelle est la **complexité temporelle** de cet algorithme en fonction de n ?
3. Applique-le au tableau : quelle combinaison donne le **score maximal exact**

Partie 3 : Analyse comparative et complexité

1. Les deux algorithmes donnent-ils toujours le même résultat ? Pourquoi ?
2. Ce problème est-il **dans P**, **NP**, **NP-complet**, **NP-difficile** ? Justifie ta réponse clairement.
3. Propose un lien avec un **problème classique de complexité** connu.

Exercice bonus : Pizza parfaite avec contraintes supplémentaires

Le client souhaite maintenant composer **plusieurs pizzas** pour un buffet. Il impose :

- **Chaque ingrédient ne peut apparaître qu'une seule fois au total** (pas de doublons sur l'ensemble du buffet),
- Chaque pizza contient **entre 3 et 5 garnitures**,
- Le **score total cumulé du buffet** doit être **strictement supérieur à 25**,
- Le **coût total ne doit pas dépasser 20 €**.

1. Décris un algorithme pour générer **un ensemble de pizzas valides** respectant ces contraintes.

> Indice : tu peux utiliser une approche par **backtracking**, **programmation dynamique**, ou **heuristique gloutonne par lot**.

2. Quelle est la **complexité de ton algorithme** dans le pire des cas ? En fonction de n, le nombre d'ingrédients disponibles.
3. Implémenterais-tu une **heuristique de type "plus grosse pizza d'abord"**, ou plutôt "pizzas légères mais nombreuses" ? Explique ta stratégie.
4. Si ce problème devait être généralisé pour n ingrédients et k pizzas, à quelle **classe de complexité** appartiendrait-il ?
5. Ce problème est-il **polynomialement réductible** à un autre problème connu ? Lequel ?