TD7 - Structure de données

Code	LU2IN006
Type de cours	Travaux dirigés
Complété ?	☐
Jour du cours	@20/03/2023

`PINHO FERNANDES Enzo - L2 Mono-Info S4 - GR6`

TD7 : Graphes

📖 **Résumé :**

- **Graph :**
  - Liste de noeuds
  - Liste d'arrêtes
- **La matrice d'adjacence**
  - $M[i][j] = 1$ <=> Il y a une arête (i, j)
  - $M[i][j] = 0$ <=> Il y a pas une arête (i, j)
- **La liste d'adjacence**
  - G :
    - `Liste[Noeud]`
  - Noeud :
    - `int num;`
    - `Liste[Arête]`

▼ Exercice 1 : Structure de graphes

📖 **Enoncé :**

Exercice 1 – Structure de graphes

On considère une carte routière où l'on connaît les coordonnées GPS x et y de chaque ville. La carte est très grande et contient un grand nombre n de villes. On veut pouvoir retrouver rapidement différentes informations sur cette carte, comme déterminer si deux villes sont reliées directement par une route et la distance entre elles. On dit ici que deux villes sont reliées directement s'il existe une route (à double sens) reliant directement les deux villes sans passer par une autre ville.

Q 1.1 Proposer une structure de données abstraite permettant de stocker les villes et l'existence d'une route reliant directement deux villes.

Q 1.2 Quelle implémentation de cette structure abstraite est adaptée à ce cas de carte routière.

Q 1.3 Proposer une structure C.

Q 1.4 Donner une représentation graphique de la mémoire pour une carte routière :
- composées de quatre villes,
- comprenant seulement deux routes directes,
- en sachant qu'il existe une ville qui n'est reliée à aucune autre ville (sommet isolé).

Q 1.5 Donner une fonction de création de la structure en l'initialisant avec n villes.

Q 1.6 Donner une fonction de mise à jour des informations d'un sommet dont le numéro est donné en argument de la fonction.

Q 1.7 Écrire un `main` permettant de créer un graphe et d'entrer quelques villes.

Q 1.8 Proposer une fonction d'ajout d'une route entre deux villes. Utilisez la dans votre `main`.

Q 1.9 Donner une fonction d'affichage en mode texte de la structure.

Q 1.10 Comment désallouer la structure utilisée ?

💡 **Réponses :**

▼ **Question 1 :**

- On préfèrera utiliser une structure de graphe.

▼ **Question 2 :**

- On peut utiliser **la liste d'adjacence**. Pour chaque ville, il y a peu de route qui parte de cette ville.
  - Donc une matrice d'adjacence serait composée de beaucoup de 0, donc il est préférable d'utiliser la liste d'adjacence plutôt.

▼ **Question 3 :**

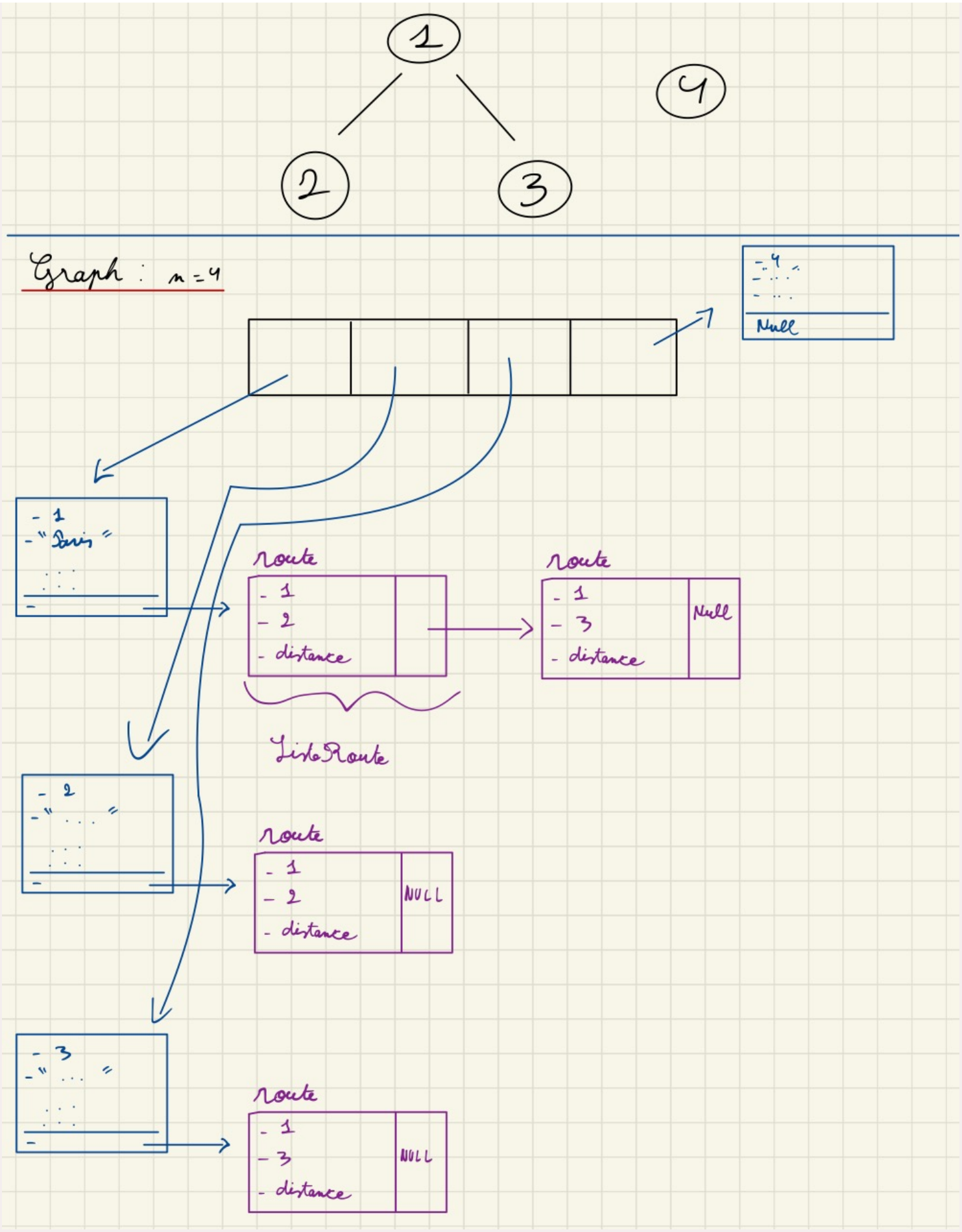
```
//Ici, le graphe n'est pas fléché.

typedef struct _route{
  int depart;
  int arrivee;
  int distance;
} Route;

typedef struct _ville{
  int num;
  char *nom;
  int x;
  int y;
  ListeRoute *routes;
} Ville;

typedef struct _graph{
  int n;
  Ville** villes;
} Graph;
```

▼ **Question 4 :**



▼ **Question 5 :**

```
/*
  Ici, on suppose que JUSTE APRES l'initialiser, on remplit ENTIEREMENT le tableau.
  On a aucune variable ici qui montre combien de cases sont remplies.
*/

Graph* initGraph(int n){
  Graph *g = (Graph*)malloc(sizeof(Graph));
  g->villes = (Ville**)malloc(sizeof(Ville*)*n);
  g->n = n;

  return g;
}
```

▼ **Question 6 :**

```
void maj(Graph *g, int u, char *nom, int x, int y){
  Ville *v = g->villes[u];
  v->nom = strdup(nom);
  v->x = x;
  v->y = y;
}
```

▼ Question 7 :

```
//Il faudrait free btw.

int main(){
  Graph *g = intGraph(3);

  ajouter_ville(g, 0, "Paris", 50, 60);
  ajouter_ville(g, 1, "Poitiers", 70, 10);
  ajouter_ville(g, 2, "Marseille", 2, 3);

  ajouter_route(g, 0, 1);
}
```

▼ Question 8 :

```
void ajouter_route(Graph *g, int u, int v){
  Route *r = (Route*)malloc(sizeof(Route));
  r->depart = min(u,v);
  r->arrivee = max(u,v);

  ajouter_en_tete(g->villes[u]->routes, r);
  ajouter_en_tete(g->villes[v]->routes, r);
}
```

▼ Question 9 :

- Flemme ig ???

▼ Question 10 :

- Stocker les routes que l'on désalloue.
- On note le nombre de routes par ville et on diminue ce nombre de 1 pour les deux villes concernées par chaque route que l'on désalloue.
  - ATTENTION, garder la liste de routes connectée pour les deux villes.

▼ Exercice 2 : Degrés des sommets

📖 **Enoncé :**

Exercice 2 – Degrés des sommets

On considère un graphe orienté contenant n sommets et m arcs. Le but de cet exercice est de déterminer le *degré sortant* (resp. *degré entrant*) d'un somment, autrement dit le nombre d'arcs sortant (res. entrant) d'un sommet. On désire comparer deux implémentations classiques de ce graphe : matrice d'adjacence et listes d'adjacence.

Q 2.1 On considère l'implémentation de ce graphe par une matrice d'adjacence dont les cases sont des valeurs 0/1 indiquant la présence ou non d'un arc. Donner une fonction `void degre_matrice(int** M, int n, int u, int* deg_e, int* deg_s);` qui prend en entrée M la matrice d'adjacence du graphe, le nombre n de sommets et le sommet u dont on veut déterminer les degrés entrant et sortant.

Q 2.2 On considère l'implémentation de ce graphe par listes d'adjacence (donné dans le cours). Donner une fonction `int degre_sortant_LA(Graphe* G, int u);` qui prend en entrée un pointeur sur le graphe et le sommet u dont on veut déterminer le degré sortant.

Q 2.3 On considère à nouveau l'implémentation de ce graphe par listes d'adjacence. Donner maintenant une fonction `int degre_entrant_LA(Graphe *G, int u);` qui prend en entrée un pointeur sur le graphe et le sommet u dont on veut déterminer le degré entrant.

Q 2.4 Comparer la complexité des fonctions de calcul de degré dans le cas des deux implémentations.

💡 **Réponses :**

- Pas le temps, contrôle.
