PARTIE I. PRÉLIMINAIRES: LISTES SANS REDONDANCE

S1.

```
def creerListeVide(n):
    liste=creerTableau(n+1)
    liste[0]=0
    return(liste)
```

S2.

```
def estDansListe(liste,x):
    for i in range(1,liste[0]+1):
        if liste[i]==x:
            return(True)
    return(False)
```

La complexité temporelle de cette fonction est un $\mathcal{O}(n)$, où n désigne le nombre maximal d'éléments dans la liste.

S3.

```
def ajouteDansListe(liste,x):
    if not estDansListe(liste,x):
        liste[0]+=1
        liste[liste[0]]=x
```

Lorsque la liste est pleine initialement et que l'élément x n'y figure pas, la dernière ligne de cette procédure provoque une erreur car la case destinée à recevoir la valeur x n'existe pas. L'assignation d'une valeur à une case du tableau est de coût constant donc le coût temporel de cette fonction est celui de la fonction `estDansListe`, à savoir $\mathcal{O}(n)$.

PARTIE II. CRÉATION ET MANIPULATION DE PLANS

S4. Les deux plans dessinés sont respectivement représentés par les tableaux :

```
plan1=[[5, 7],[2, 2, 3, *, *],[3, 1, 3, 5, *],
[4, 1, 2, 4, 5],[2, 3, 5, *, *],[3, 2, 3, 4, *]]
plan2=[[5, 4],[1, 2, *, *, *],[3, 1, 3, 4, *],
[1, 5, *, *, *],[2, 2, 5, *, *],[1, 4, *, *, *]]
```

S5.

```
def creerPlanSansRoute(n):
    liste=creerListeVide(n)
    liste[0]=[n,0]
    for i in range(1,n+1):
        liste[i]=creerListeVide(n-1)
    return(liste)
```

S6.

```
def estVoisine(plan,x,y):
    for i in range(1,plan[x][0]+1):
        if plan[x][i]==y:
            return(True)
    return(False)
```

ou

```
def estVoisine(plan,x,y):
    return(estDansListe(plan[x],y))
```

S7.

```
def ajouteRoute(plan,x,y):
    if not estVoisine(plan,x,y):
        plan[0][1]+=1
        ajouteDansListe(plan[x],y)
        ajouteDansListe(plan[y],x)
```

Il n'y a ici aucun risque de débordement puisque chaque ville ne peut être reliée qu'à au plus $n-1$ autres villes.

S8.

```
def afficheToutesLesRoutes(plan):
    n,m=plan[0]
    affiche('Ce plan contient ',m,' route(s) :')
    for x in range(1,n+1):
        for i in range(1,plan[x][0]+1):
            y=plan[x][i]
            if x<y:
                affiche(' (', x, '-', y, ')')
    affiche('\n')
```

Les listes des voisins de chacune des villes vont être parcourues une fois chacune. Il y a n listes à parcourir et la somme des longueurs de celles-ci est égale à $2m$ donc le coût de cette fonction est un $\mathcal{O}(n+m)$.

Partie III. Recherche de chemins arc-en-ciel

S9.

```
def coloriageAleatoire(plan,couleur,k,s,t):
    for i in range(1,plan[0][0]+1):
        couleur[i]=entierAleatoire(k)
    couleur[s]=0
    couleur[t]=k+1
```

S10.

```
def voisinesDeCouleur(plan,couleur,i,c):
    liste=creerListeVide(plan[0][0])
    for j in range(1,plan[i][0]+1):
        if couleur[plan[i][j]]==c:
            ajouteDansListe(liste,plan[i][j])
    return(liste)
```

S11.

```
def voisinesDeLaListeDeCouleur(plan,couleur,liste,c):
    lst=creerListeVide(plan[0][0])
    for i in range(1,liste[0]+1):
        for j in range(1,plan[liste[i]][0]+1):
            if couleur[plan[liste[i]][j]]==c:
                ajouteDansListe(lst,plan[liste[i]][j])
    return(lst)
```

Cette fonction parcourt chacune des listes de voisins des villes présentes dans liste. Celle-ci contient au maximum n villes et la somme des longueurs des listes des voisins est majorée par $2m$. Enfin, pour chaque voisin considéré il faut déterminer sa couleur (ce qui se fait en coût constant) puis le cas échéant l'ajouter dans `lst`, ce qui a un coût en $\mathcal{O}(n)$. Le coût total de cette fonction est donc un $\mathcal{O}(n(n+m))$.

S12.

```
def existeCheminArcEnCiel(plan,couleur,k,s,t):
    liste=creerListeVide(plan[0][0])
    ajouteDansListe(liste,s)
    for c in range(1,k+2):
        liste=voisinesDeLaListeDeCouleur(plan,couleur,liste,c)
        if liste[0]==0:
            return(False)
    return(True)
```

La création de liste est un $\mathcal{O}(n)$; ensuite, pour chaque valeur de `c` dans l'intervalle compris entre 1 et $k+1$ on applique la fonction `voisinesDeLaListeDeCouleur`, de coût $\mathcal{O}(n(n+m))$. Le coût dans le pire des cas est donc un $\mathcal{O}(kn(n+m))$.

Partie IV. Recherche de chemin passant par exactement k villes intermédiaires distinctes

S13.

```
def existeCheminSimple(plan,k,s,t):
    couleur=creerListeVide(plan[0][0])
    for i in range(k**k):
        coloriageAleatoire(plan,couleur,k,s,t)
        if existeCheminArcEnCiel(plan,couleur,k,s,t):
            return(True)
    return(False)
```

La création du tableau couleur a un coût en $\mathcal{O}(n)$. La fonction `coloriageAleatoire` a un coût en $\mathcal{O}(n)$ et la fonction `existeCheminArcEnCiel` un coût en $\mathcal{O}(kn(n+m))$.
La fonction `existeCheminSimple` a donc un coût en $\mathcal{O}(k^{k+1}n(n+m))$. conforme à l'objectif du problème.

S14. Pour renvoyer un chemin détecté avec succès, on peut modifier la fonction `existeCheminArcEnCiel` pour mémoriser le chemin arc-en-ciel, s'il en existe.
Pour cela, on modifie la fonction `voisinesDeLaListeDeCouleur` : celle-ci prend maintenant en argument une liste représentant non plus un ensemble de sommets mais un ensemble de chemins (représenté par le type `list`), et pour chacun d'eux rajoute un sommet de couleur `c` si cela s'avère possible :

```
    def voisinesDeLaListeDeCouleur2(plan,couleur,liste,c):
        lst=creerListeVide(plan[0][0])
        for i in range(1,liste[0]+1):
            for j in range(1,plan[liste[i][-1]][0]+1):
                if couleur[plan[liste[i][-1]][j]]==c:
                    ajouteDansListe(lst,liste[i]+[plan[liste[i][-1]][j]])
        return(lst)
```

On modifie ensuite la fonction `existeCheminArcEnCiel` en conséquence, pour retourner un couple formé d'un booléen et d'une solution, s'il en existe :

```
    def existeCheminArcEnCiel2(plan,couleur,k,s,t):
        liste=creerListeVide(plan[0][0])
        ajouteDansListe(liste,[s])
        for c in range(1,k+2):
            liste=voisinesDeLaListeDeCouleur2(plan,couleur,liste,c)
            if liste[0]==0:
                return(False,[])
        return(True,liste[1])
```

Il reste alors à modifier légèrement la fonction principale :

```
    def existeCheminSimple2(plan,k,s,t):
        couleur=creerListeVide(plan[0][0])
        for i in range(k**k):
            coloriageAleatoire(plan,couleur,k,s,t)
            r,lst=existeCheminArcEnCiel2(plan,couleur,k,s,t)
            if r:
                return(lst)
        return(False)
```