

Corrigé du Problème du DS1

le 28/09/2024

Épreuve X info PSI-PT 2015

Partie I

Question 1 –

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

Question 2 –

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

On parcourt dans le pire des cas une fois la liste, avec un test à chaque fois ; la complexité est donc linéaire en n .

Question 3 –

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

Si la liste était pleine initialement, la commande `liste[liste[0]+1]=x` ne pourra pas aboutir car `liste[0]+1` ne fait pas partie des indices de `liste` ("out of range").

On fait appel à la fonction `estDansListe(liste,x)`, qui est de complexité linéaire, puis on effectue deux affectations. La complexité de cette nouvelle fonction est donc linéaire.

Partie II

Question 4 –

```
Plan1=[[5,7],  
        [2,2,3,*,*],  
        [3,1,3,5,*],  
        [4,2,1,4,5],  
        [2,5,3,*,*],  
        [3,2,3,4,*]]  
  
Plan2=[[5,4],  
        [1,2,*,*,*],  
        [3,1,3,4,*],  
        [1,2,*,*,5],  
        [2,2,5,*,*],  
        [1,4,*,*,*]]
```

Question 5 –

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

Question 6 –

```
def estVoisine(plan,x,y):
    if estDansListe(plan[x],y):
        return True
    else :
        return False
```

Question 7 –

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

Il n'y a pas de risque de dépassement de la capacité des listes, car si une liste est pleine, il ne peut y avoir de ville à rajouter. La structure de liste choisie au début du problème est donc pertinente.

Question 8 –

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

A priori la boucle externe est de longueur n et la boucle interne de longueur n dans le pire des cas, donc la complexité est en $\mathcal{O}(n^2)$.

Mais on peut en fait affiner cela. Le nombre de routes étant égal à m , le nombre total de ville y parcouru ne dépasse pas $2m$. Il faut également compter le parcours des n villes (qui nécessite n lectures de `plan[x][0]`). La complexité est donc en $\mathcal{O}(n + m)$.

Partie III

Question 9 –

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

Question 10 –

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

Question 11 –

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

Tout comme à la question 8, le parcours des boucles imbriquées est en $\mathcal{O}(n + m)$. Comme la fonction `ajouteDansListe(tab, y)` est linéaire en n , cela donne une complexité totale en $\mathcal{O}(n(n + m))$.

Question 12 –

```
def existeCheminArcEnCiel(plan, couleur, k, s, t):
    n=plan[0][0]
    liste=creerListeVide(n-1)
    ajouteDansListe(liste, s)
    for j in range(1, k+2):
        liste=voisinesDeLaListeDeCouleur(plan, couleur, liste, j)
    return estDansListe(liste, t)
```

La boucle externe est de longueur $k+1$ et on appelle à chaque passage la fonction `voisinesDeLaListeDeCouleur(plan, couleur, liste, j)`. On en déduit que la complexité est en $\mathcal{O}(k n(n + m))$.

Partie IV

Question 13 –

```
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 fonction `existeCheminArcEnCiel(plan, couleur, k, s, t)` est en $\mathcal{O}(k n(n + m))$ et `coloriageAleatoire(plan, couleur, k, s, t)` est en $\mathcal{O}(k n(n + m))$. La complexité de cette fonction est donc au total en $\mathcal{O}(k^k \cdot k n(n + m))$, c'est à dire $\mathcal{O}(k^{k+1} n(n + m))$.

Question 14 –

Il faut faire en sorte que la fonction `existeCheminArcEnCiel(plan, couleur, k, s, t)`, en plus de renvoyer `True` si le chemin voulu existe, renvoie ce chemin.