TP5 d'Informatique
Les chemins arc-en-ciel

— Le but de ce TP est de décider s'il existe, entre deux villes données, un chemin passant par exactement k villes intermédiaires distinctes, dans un plan contenant au total n villes reliées par m routes.

— La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de P. Lorsque cette complexité dépend de plusieurs paramètres n, m et k, on dira que P a une complexité en $O(\phi(n,m,k))$ lorsqu'il existe quatre constantes A, n_0, m_0 et k_0 telles que la complexité de P soit inférieure ou égale à $A \times \phi(n,m,k)$ pour tout $n \geq n_0, m \geq m_0$ et $k \geq k_0$. Lorsqu'il est demandé de préciser la complexité d'un programme, le candidat devra justifier cette dernière.

— On suppose que l'on dispose d'une fonction $creerTableau(n)$ qui alloue un tableau de taille n indexé de 0 à $n-1$ (les valeurs contenues dans le tableau initialement sont arbitraires). L'instruction $b = creerTableau(n)$ créera un tableau de taille n et l'affectera à la variable b.

— On pourra ainsi créer un tableau a de p tableaux de taille q par la suite d'instructions suivante :

```
A=creerTableau(p)
for i range(p):
        a[i]=creerTableau(q)
```

— On accédera par l'instruction $a[i][j]$ à la j-ième case du i-ième tableau contenu dans le tableau a ainsi créé. Par exemple, la suite d'instructions suivante remplit le tableau a avec les sommes des indices i et j de chaque case :

```
for i in range(p):
        for j in range(q):
                a[i][j]=i+j
```

On supposera que cet accès se fait en temps constant.

— Dans la suite, nous distinguerons fonction et procédure : les fonctions renvoient une variable tandis que les procédures ne renvoient rien.

Première partie

On souhaite stocker une liste l_0 non-ordonnée d'au plus n entiers sans redondance (i.e. où aucun entier n'apparaît plusieurs fois). Nous nous proposons d'utiliser une liste "$liste$" de longueur $n+1$ telle que :

- $liste[0]$ contient le nombre d'éléments dans la liste l_0.
- pour tout $i \in [\![1, liste[0]]\!]$, $liste[i]$ contient $l_0[i-1]$.

Lorsque la liste l_0 est vide, $liste$ est une liste contenant $n+1$ fois la valeur 0.
Pour éviter toute confusion, nous appèlerons "liste" la liste l_0 et "tableau" la liste représentant la liste l_0 de taille $n+1$.

1. Écrire une fonction $creerTabVide(n)$ qui crée, initialise et renvoie un tableau de longueur $n+1$ correspondant à la liste vide. Ces deux données seront représentées en Python par des listes.
2. Écrire une fonction $estDansTab(tab, x)$ qui renvoie $True$ si l'élément x apparaît dans la liste représentée par le tableau tab, et renvoie $False$ sinon. On ne s'autorisera pas à utiliser le test in, ni ici ni dans la suite du problème.
   Quelle est la complexité en temps de votre fonction dans le pire des cas en fonction du nombre maximal n. d'éléments dans la liste ?
3. Écrire une procédure $ajouteDansTab(tab, x)$ qui modifie de façon appropriée le tableau tab pour y ajouter x si l'entier x n'appartient pas déjà à la liste, et ne fait rien sinon.
   Quel est le comportement de votre procédure si la liste tab est "pleine" initialement et que x n'est pas encore dans la liste initiale ? (on ne demande pas de gérer informatiquement le problème rencontré)
   Quelle est la complexité en temps de votre procédure dans le pire cas, en fonction du nombre maximal d'éléments dans la liste ?

Deuxième partie

Un plan P est défini par : un ensemble de n villes numérotées de 1 à n et un ensemble de m routes (toutes à double sens) reliant chacune deux villes ensemble. On dira que deux villes $x, y \in [\![1, n]\!]$ sont voisines lorsqu'elles sont reliées par une route, ce que l'on notera par $x \sim y$. On appellera chemin de longueur k toute suite de villes $v_1, \cdots, v_k$ telle que $v_1 \sim v_2 \sim \cdots \sim v_k$. On représentera les villes d'un plan par des ronds contenant leur numéro et les routes par des traits reliant les villes voisines.

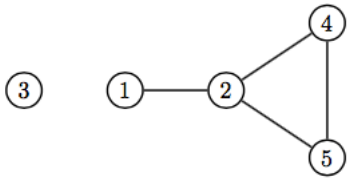


FIGURE 1 – Un plan à 5 villes et 4 routes

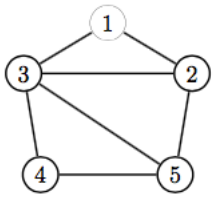
Nous représenterons tout plan P à n villes par un tableau $plan$ de $n+1$ tableaux où :

- $plan[0]$ contient un tableau à deux éléments :
  - ⋆ $plan[0][0] = n$ est le nombre de villes du plan P.
  - ⋆ $plan[0][1] = m$ est le nombre de routes du plan P.
- Pour chaque ville $x \in [\![1, n]\!]$, $plan[x]$ contient un tableau à n éléments représentant la liste à au plus $n-1$ éléments des villes voisines de la ville x dans P dans un ordre arbitraire, en utilisant la structure de liste sans redondance définie dans la partie précédente.
  - ⋆ $plan[x][0]$ est le nombre de villes voisines de x.
  - ⋆ $plan[x][1], \cdots, plan[x][i]$ (où i est le contenu de $plan[x][0]$) sont les indices des villes voisines de x.

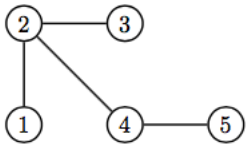
  Par exemple, le plan de la première figure sera représenté par le tableau de tableaux suivant (les $*$ représentent des données arbitraires, correspondant aux valeurs non utilisées des tableaux. $*$ sera représenté par 0 dans vos codes) :

```
plan=[ [5 4],
       [ 1 2  * * * ],
       [ 3 4 1 5 * ],
       [ 0 *  *  * * ],
       [ 2 2 5  * * ],
       [ 2 4 2 * *  ] ]
```

1. Représenter sous forme de tableaux de tableaux les deux plans suivants :



Plan 1



Plan 2

   On pourra utiliser dans la suite les fonctions et procédures de gestion de listes définies dans la partie précédente.
2. Écrire une fonction $creerPlanSansRoute(n)$ qui crée, remplit et renvoie le tableau de tableaux correspondant au plan à n villes n'ayant aucune route.
3. Écrire une fonction $estVoisine(plan, x, y)$ qui renvoie $True$ si les villes x et y sont voisines dans le plan codé par le tableau de tableaux $plan$, et renvoie $False$ sinon.
4. Écrire une procédure $ajouteRoute(plan, x, y)$ qui modifie le tableau de tableaux $plan$ pour ajouter une route entre les villes x et y si elle n'y est pas encore. On prendra garde à bien mettre à jour toutes les cases concernées. Y a-t-il un risque de dépassement de capacité des listes ?
5. Écrire une procédure $afficheToutesLesRoutes(plan)$ qui affiche à l'écran la liste des routes du plan codé par $plan$, où chaque route apparaît exactement une seule fois. Par exemple, pour le graphe de la figure 1, votre procédure pourra afficher la liste :

$$[(1-2), (2-4), (2-5), (4-5)]$$

   Quelle est la complexité en temps de votre procédure dans le pire des cas en fonction de n et m ?

Troisième partie

Étant données deux villes distinctes s et $t \in [\![1, n]\!]$, nous recherchons un chemin de s à t passant pas exactement k villes intermédiaires toutes distinctes. L'objectif de cette partie et de la suivante est de construire une fonction qui va détecter en temps linéaire en $n(n+m)$ l'existence d'un tel chemin avec une probabilité indépendante de la taille du plan $n+m$.
Le principe de l'algorithme est d'attribuer à chaque ville $i \in [\![1, n]\!] \backslash \{s, t\}$ une couleur aléatoire codée par un entier aléatoire uniforme $couleur[i] \in \{1, \cdots, k\}$ stocké dans un tableau couleur de taille $n+1$ (la case 0 n'est pas utilisée).
Les villes s et t reçoivent respectivement les couleurs spéciales 0 et $k+1$, i.e. $couleur[s] = 0$ et $couleur[t] = k+1$.
L'objectif de cette partie est d'écrire une procédure qui détermine s'il existe un chemin de longueur $k+2$ allant de s à t dont la j-ième ville intermédiaire a reçu la couleur j. Dans l'exemple de la figure 2, le chemin $6 \sim 7 \sim 8 \sim 3 \sim 4$ de longueur $5 = k+2$ qui relie $s = 6$ à $t = 4$ vérifie cette propriété pour $k = 3$.
On suppose l'existence d'une fonction $entierAleatoire(k)$ qui renvoie un entier aléatoirement (de manière uniforme) entre 1 et k (i.e. telle que $\mathbb{P}(entierAleatoire(k) = c) = \frac{1}{k}$ pour tout entier $c \in [\![1, k]\!]$, $\mathbb{P}$ désignant la probabilité).

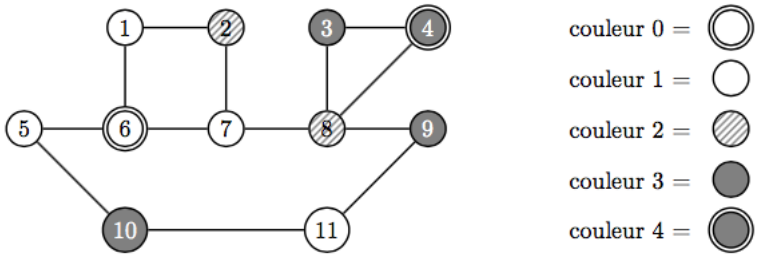


FIGURE 2 – Exemple de plan colorié pour $k = 3$, $s = 6$, $t = 4$

1. Écrire une procédure $coloriageAleatoire(plan, couleur, k, s, t)$ qui prend en argument un plan de n villes, un tableau couleur de taille $n+1$, un entier k et deux villes s et $t \in [\![1, n]\!]$, et remplit le tableau couleur avec : une couleur aléatoire choisie uniformément dans $[\![1, k]\!]$, pour chaque ville $i \in [\![1, n]\!] \backslash \{s, t\}$, les différents choix étant indépendants les uns des autres ; et les couleurs 0 et $k+1$ pour s et t respectivement.

   Nous cherchons maintenant à écrire une fonction qui calcule l'ensemble des villes de couleur c voisines d'un ensemble de villes donné. Dans l'exemple de la figure 2, l'ensemble des villes de couleur 2 voisines des villes $\{1, 5, 7\}$ est $\{2, 8\}$.
2. Écrire une fonction $voisinesDeCouleur(plan, couleur, i, c)$ qui crée et renvoie un tableau codant la liste sans redondance des villes de couleur c voisines de la ville i dans le plan plan colorié par le tableau couleur.
3. Écrire une fonction $voisinesDeLaListeDeCouleur(plan, couleur, liste, c)$ qui crée et renvoie un tableau codant la liste sans redondance des villes de couleur c voisines d'une des villes de la liste sans redondance liste, dans le plan plan colorié par couleur.
   Quelle est la complexité de votre fonction dans le pire des cas en fonction de n et m ?
4. Écrire une fonction $existeCheminArcEnCiel(plan, couleur, k, s, t)$, dont la complexité est dans le pire des cas en $O(n(m+n))$ (ce que l'on justifiera), qui renvoie $True$ s'il existe un chemin $s \sim v1 \sim \cdots \sim vk \sim t$ tel que $couleur[v_j] = j$ pour tout $j \in [\![1, k]\!]$; et renvoie $False$ sinon.
5. Réctifier la fonction précédente et proposer une fonction $CheminArcEnCiel(plan, couleur, k, s, t)$ qui renvoie $[s, v_1, \cdots, v_k, t]$ où $s \sim v1 \sim \cdots \sim vk \sim t$ tel que $couleur[v_j] = j$ pour tout $j \in [\![1, k]\!]$ si un tel chemin existe ; et renvoie $False$ sinon.

Quatrième partie

Si les couleurs des villes sont choisies aléatoirement et uniformément dans $[\![1, k]\!]$, la probabilité que j soit la couleur de la j-ième ville d'une suite fixée de k villes vaut $\frac{1}{k}$, indépendamment pour tout j. Ainsi, étant données deux villes distinctes s et $t \in [\![1, n]\!]$, s'il existe dans plan un chemin de s à t passant par exactement k villes intermédiaires toutes distinctes et si le coloriage couleur est choisi aléatoirement conformément à la procédure $coloriageAleatoire(plan, couleur, k, s, t)$, la procédure $existeCheminArcEnCiel(plan, couleur, k, s, t)$ répond $True$ avec une probabilité au moins $\frac{1}{k^k}$, et répond toujours $False$ sinon. Ainsi, si un tel chemin existe, la probabilité qu'une parmi k^k exécutions indépendantes de $existeCheminArcEnCiel$ réponde $True$ est supérieur ou égale à (inégalité étant admise) :

$$1 - (1 - k^{-k})^{k^k} \geq 1 - \frac{1}{e}$$

1. Écrire une fonction $existeCheminSimple(plan, k, s, t)$ qui renvoie $True$ avec une probabilité au moins $1 - \frac{1}{e}$ s'il existe un chemin de s à t passant par exactement k villes intermédiaires toutes distinctes dans le plan "plan", et renvoie toujours $False$ sinon.
   Quelle est sa complexité en fonction de k, n et m dans le pire des cas ?
   Exprimez-la sous la forme $O(f(k) \times g(n, m))$, pour f et g bien choisies.
2. Expliquer comment modifier votre programme pour renvoyer un tel chemin s'il est détecté avec succès.