Centres étrangers – 2025 – sujet1 - Correction

Exercice 1 (6 points)

---

1. Compléter la liste des voisins

D'après la figure (réseau initial à 5 sommets) :

- 0 est relié à 1, 2, 3 et 4
- 1 est relié à 0, 2 et 3
- 2 est relié à 0 et 1
- 3 est relié à 0 et 1
- 4 est relié à 0

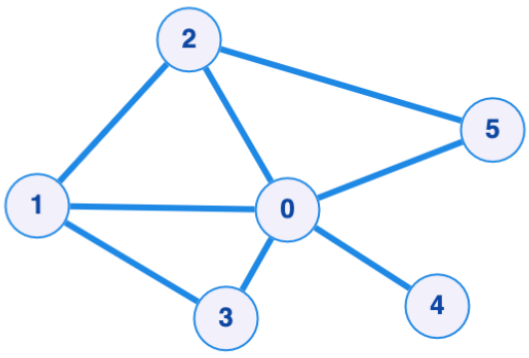
Donc :

```
voisins = [
    [1, 2, 3, 4],
    [0, 2, 3],
    [0, 1],
    [0, 1],
    [0]
]
```

👉 *Commentaire : dans un graphe non orienté, si a est voisin de b alors b est voisin de a.*

---

2. Dessin du graphe



3. Ajout du sommet 5

Le sommet 5 est accessible uniquement depuis 0 et 2.

On ajoute donc les arêtes :

- 5 ↔ 0
- 5 ↔ 2

Nouvelle définition :

```
voisins = [
    [1, 2, 3, 4, 5],
    [0, 2, 3],
    [0, 1, 5],
    [0, 1],
    [0],
    [0, 2]
]
```

👉 *Commentaire : on modifie aussi les listes des sommets 0 et 2.*

---

4. Fonction voisin_alea

```
import random

def voisin_alea(voisins, s):
    indice = random.randrange(len(voisins[s]))
    return voisins[s][indice]
```

ou directement, en compréhension:

```
def voisin_alea(voisins, s):
    return voisins[s][ random.randrange(len(voisins[s]))]
```

👉 *Commentaire : on choisit un indice aléatoire dans la liste des voisins du sommet s.*

---

5. Justification du caractère récursif de marche_alea

La fonction s'appelle elle-même :

```
return marche_alea(voisins, voisin_alea(voisins, i), n-1)
```

👉 *Commentaire : une fonction est récursive lorsqu'elle se rappelle elle-même.*

6. Ce que modélise marche_alea

Cette fonction modélise le déplacement aléatoire du virus pendant n étapes à partir du sommet i.

À chaque étape :

- le virus choisit un voisin au hasard,
- puis recommence jusqu'à épuisement des étapes.

👉 *Commentaire : il s'agit d'une marche aléatoire sur un graphe.*

---

7. Fonction simule

```
def simule(voisins, i, n_tests, n_pas):
    results = [0] * len(voisins)

    for _ in range(n_tests):
        sommet_final = marche_alea(voisins, i, n_pas)
        results[sommet_final] += 1

    for j in range(len(results)):
        results[j] = results[j] / n_tests

    return results
```

👉 *Commentaire : on compte les fréquences de présence finale du virus. D'autres méthodes étaient possibles.*

---

8. Ordinateur le plus rentable à protéger

Résultat donné : `[0.328, 0.195, 0.18, 0.12, 0.059, 0.118]`

La probabilité la plus élevée est 0.328 au sommet 0. Il est donc le plus rentable de protéger l'ordinateur 0.

👉 *Commentaire : c'est celui où le virus termine le plus souvent sa marche.*

---

9. Propagation totale du virus

On cherche le nombre d'étapes nécessaires pour contaminer tout le réseau.

Il faut effectuer un parcours en largeur du graphe (BFS).

Principe :

- initialiser un ensemble contaminé avec le sommet de départ
- à chaque étape :
  - ajouter tous les voisins non encore contaminés
- compter le nombre d'étapes nécessaires jusqu'à ce que tous les sommets soient contaminés

Algorithme possible :

```
def temps_propagation(voisins, s):
    contamines = {s}
    frontiere = {s}
    temps = 0

    while len(contamines) < len(voisins):
        nouvelle_frontiere = set()
        for sommet in frontiere:
            for v in voisins[sommet]:
                if v not in contamines:
                    nouvelle_frontiere.add(v)

        contamines.update(nouvelle_frontiere)
        frontiere = nouvelle_frontiere
        temps += 1

    return temps
```

👉 *Commentaire : il s'agit d'un parcours en largeur ; le temps correspond à la distance maximale depuis le sommet initial.*