TP3: Analyse amortie et Algorithmes probabilistes

Ce sujet est constitué de trois exercices dont le but est d'implémenter deux algorithmes probabilistes du cours, QUICKSORT et COUPEMIN, et un algorithme vu en TD qui permet d'avoir un compteur binaire avec incrémentation et décrémentation en temps amorti constant.

Exercice 1. QUICKSORT

Pour cet exercice, un code partiellement rempli est fourni dans le fichier `TP3-Exo1-Quicksort.py`. Pensez à tester tous vos tris sur des petits exemples. N'oubliez de recopier vos tableaux pour effectuer plusiers tris sur un même tableau. Cela peut se faire par l'appel `T2=T1.copy()`.

1. Implémenter une fonction `Rempli_Hasard(T,n)` qui remplit le tableau T de taille n avec des entiers pris au hasard entre 1 et $n*n$.
2. Écrire une fonction `Tri_Bulles(T)` qui implémente un tri à bulles. [1]
3. Écrire une fonction `Tri_Fusion(T)` qui implémente un tri fusion. [2]
4. Écrire une fonction `QuickSort(T)` qui implémente le tri rapide vu en cours.
5. Comparer les temps d'exécution des tris implémentés. On voit que le tri fusion et le tri par Quicksort ont des performances comparables.
6. Rajouter dans la batterie de test le tri fourni par Python par l'intermédiaire de la commande `sorted(T)`. On pourra se reporter à `https://stackoverflow.com/questions/10948920/what-algorithm-does-pythons-sorted-use` pour avoir quelques informations.

Exercice 2. COUPEMIN

Le but de cet exercice est d'implémenter l'algorithme probabiliste qui calcule une coupe minimale d'un graphe vu en cours. L'algorithme est simple à implémenter mais la gestion des graphes et des contractions d'arêtes peut s'avérer délicate. Pour cela on codera un graphe au cours de l'algorithme par :

- Son nombre `n` de sommets. Les sommets du graphe seront codés par les entiers 0 à $n-1$.
- Le tableau `E` de ses arêtes, chaque arête étant une liste de taille 2 contenant ses extrémités. On prendra bien garde à n'avoir jamais de boucle au cours de l'algorithme, c'est-à-dire d'arête du type $[x,x]$ pour une valeur de sommet x quelconque.
- Le tableau `Contract` de taille n qui contient des entiers de 0 à $n-1$ tels que deux sommets i et j ont été contractés précédemment par l'algorithme si, et seulement si, $Contract[i] = Contract[j]$.

Un exemple de déroulement de l'algorithme et de ces diverses structures de données est fourni ci-dessous.

```
Nombre de sommets:  6
Liste d'aretes initiale:  [[2, 0], [2, 1], [4, 3], [5, 1], [5, 2], [5, 4]]
Tableau des sommets contractes:  [0, 1, 2, 3, 4, 5]
Arete contractee:  [4, 3]
Liste d'aretes courante:  [[2, 0], [2, 1], [5, 1], [5, 2], [5, 3]]
Tableau des sommets contractes courant:  [0, 1, 2, 3, 3, 5]
Arete contractee:  [2, 0]
Liste d'aretes courante:  [[0, 1], [5, 1], [5, 0], [5, 3]]
Tableau des sommets contractes courant:  [0, 1, 0, 3, 3, 5]
Arete contractee:  [5, 0]
Liste d'aretes courante:  [[0, 1], [0, 1], [0, 3]]
Tableau des sommets contractes courant:  [0, 1, 0, 3, 3, 0]
Arete contractee:  [0, 1]
Liste d'aretes courante:  [[1, 3]]
Tableau des sommets contractes courant:  [1, 1, 1, 3, 3, 1]
Coupe:
[0, 1, 2, 5]
[3, 4]
Nombre d aretes traversant la coupe:  1
```

1. Au cas où : `https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles`
2. Au cas où : `https://fr.wikipedia.org/wiki/Tri_fusion`

1. Écrire une fonction `GenereGraph(n,p)` qui génére un graphe sur n sommets de densité p. Plus précisément, votre fonction ajoutera l'arête $[i, j]$ avec probabilité p à la liste d'arêtes `E` du graphe pour tout $0 \leq i < j \leq n-1$. Le tableau `Contract` sera initialisé à $[0, 1, \ldots, n-1]$ et la fonction retournera le couple `E,Contract`.
   Pour tester vos fonctions, choisissez des graphes sur 8 sommets de densité 0.3 par exemple.
2. Écrire une fonction `ContractArete(E,Contract)` qui tire au hasard une arête uniformément dans le tableau `E`. Si on note `[u,v]` cette arête, on va ensuite la contracter, en remplaçant les occurrences de u par v dans les tableaux `E` et `Contract`. Pour finir, il faudra éliminer les possibles boucles dans le tableau `E`.
3. Écrire une fonction `CoupeMin(E,Contract)` qui appellera $n-2$ fois la fonction `ContractArete` sur le graphe correspondant passé en paramètre. À la suite de ces appels, le tableau `Contract` doit contenir normalement deux valeurs correspondant à la partition des sommets obtenus à la fin de l'algorithme. Votre fonction `CoupeMin` doit retourner cette partition ainsi que le nombre d'arêtes traversant la coupe obtenue.
4. Implémenter le lemme de répétition en répétant l'appel à `CoupeMin` un nombre $2.n^2$ de fois. Récupérer le nombre minimal d'arêtes obtenu.
5. Implémenter un algorithme de recherche exhaustif de coupe minimale et comparer avec les solutions trouvées précédemment. Pour générer toutes les coupes d'un graphe, on pourra utiliser un compteur binaire `compteur` de taille n parcourant toutes les séquences de 0 et 1. Pour une valeur du compteur fixée, la partition du graphe correspondante sera donnée par $(\{i \ : \ \texttt{compteur}[i] = 0\}, \{i \ : \ \texttt{compteur}[i] = 1\})$. On calculera alors le nombre minimal d'arêtes traversant les partitions obtenues.

Exercice 3. *Compteur binaire avec incrément et décrément*

Le but de l'exercice est d'implémenter le compteur binaire avec incrément et décrément de l'Exercice 2 de la fiche de TD 3.

1. Dans un premier temps, implémenter le compteur (P,N) sans représentation exclusive de P et N.
2. À partir de 0, effectuer une séquence de 10000 INCRÉMENT et DÉCRÉMENT successifs. Observer les valeurs de P et N.
3. Finir l'implémentation du compteur en mettant en place la représentation exclusive de P et N. Reprendre la question précédente.