TD #1 — Révisions

Exercice 1. *Les œufs mystérieux (un grand classique)*

On vous donne un panier d'œufs mystérieux et vous indique une tour de N étages. Votre tâche est de déterminer à partir de quel étage lâcher un œuf le cassera. Vous savez que cet étage existe et est le même pour tous les œufs, que si un œuf n'est pas cassé (respectivement, est cassé) à un étage il ne sera pas cassé (resp., sera cassé) à tout étage inférieur (resp., supérieur). De plus, un œuf lâché mais non cassé n'est pas endommagé : il ne cassera pas plus bas qu'initialement.

L'objectif de cet exercice est de concevoir trois algorithmes permettant de déterminer le premier étage à partir duquel les œufs cassent, avec trois limites différentes sur le nombre *maximum* (c'est à dire, dans le pire cas) d'œufs que vous pouvez être amené à casser afin de trouver le résultat.

On demande seulement une description informelle (mais néanmoins précise !) des algorithmes.

1. Donnez un algorithme cassant *un* œuf dans le pire cas. Quel est le nombre maximum de lâchés effectués dans le pire cas ?
2. Donnez un algorithme effectuant le plus petit nombre de lâchés dans le pire cas. Quel est le nombre maximum d'œufs cassés dans le pire cas (à une constante près) ?
3. Donnez un algorithme cassant *deux* œuf dans le pire cas, dont le nombre de lâchés effectués dans le pire cas est un $O(\sqrt{N})$.

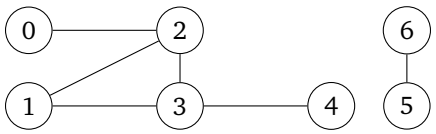
Exercice 2. *Graphes (adapté d'un roman de Georges Perec)*

Ces échanges, qui sont suscités aussi bien par des occasions propices de vente ou d'achat (il s'agit alors de faire de la place) que par des inspirations subites, des lubies, des caprices ou des dégoûts, ne se font pas au hasard, et n'épuisent pas les douze possibilités de permutations qui pourraient se faire entre ces quatre lieux et que la figure 1 *met bien en évidence ; ils obéissent strictement au schéma de la* figure 2 *: quand Madame Marcia achète quelque chose, elle le met chez elle, dans son appartement, ou dans sa cave ; de là, ledit objet peut passer dans l'arrière-boutique, et de l'arrière-boutique dans le magasin ; du magasin enfin il peut revenir — ou parvenir s'il venait de la cave — dans l'appartement. Ce qui est exclu, c'est qu'un objet revienne dans la cave, ou arrive au magasin sans être passé par l'arrière-boutique, ou repasse du magasin dans l'arrière-boutique, ou de l'arrière-boutique dans l'appartement, ou enfin passe directement de la cave à l'appartement.*

1. Dessinez les graphes des figures 1 & 2.
2. Comment appelle-t'on les graphes semblables à celui de la figure 1.

Exercice 3. *Graphes (adapté d'un exercice de Marie Durand)*

On donne le graphe non orienté suivant :



1. Donnez le degré du sommet 2.
2. Ce graphe est-il acyclique ? Si non, donnez un exemple de cycle.
3. Ce graphe est-il connexe ? Si non, donnez les ensembles de sommets formant ses composantes connexes.
4. Donnez deux chemins allant du sommet 0 au sommet 4. Combien y a-t'il de chemins possibles entre ces sommets ?
5. Donnez une représentation en Python de ce graphe par liste de listes d'adjacence, utilisant des `list`.
6. De quelle autre façon pourrait-on représenter ce graphe ?
7. Écrivez une fonction Python `reachable(G, src, dst)` qui prend en entrée un graphe et les identifiants de deux sommets (représentés comme à la question précédente) et renvoie `True` (resp. `False`) s'il existe un chemin dans le graphe entre les deux sommets.
8. Donnez un plus-court-chemin entre les sommets 2 et 3. Quel algorithme pourrait être utilisé pour calculer un ensemble de plus-court-chemins entre le sommet 2 et tous les sommets qui lui sont accessibles ?
9. Donnez une spécification adaptée à chacun des algorithmes suivants, qu'on suppose utilisés pour effectuer une recherche de plus-court-chemins :
   - **i.** Parcours en largeur.
   - **ii.** Algorithme « de Dijkstra ».
   - **iii.** Algorithme A^*

Exercice 4. *Preuve d'algorithme*

On donne l'algorithme suivant :

```
# Renvoie True si l'entier positif x est présent dans
# la liste non vide triée (par ordre croissant) L d'entiers
# positifs et False sinon
def isIn(L:list, x):
    bot = 0
    top = len(L) - 1
    while bot < top:
        mid = bot + (top - bot)//2
        if x == L[mid]:
            return True
        elif x < L[mid]:
            bot, top = bot, mid - 1
        else:
            bot, top = mid + 1, top
    if bot > top:
        return False
    else
        return L[bot] == x
```

1. Quel est le nom du principe utilisé dans cet algorithme ?
2. Montrez que cet algorithme termine toujours, en utilisant un variant approprié.
3. Montrez que cet algorithme renvoie toujours la bonne réponse, en utilisant un invariant.
4. Quelle propriété de correction est démontrée par les deux questions précédentes ?
5. Donnez la complexité pire cas de cette fonction en prenant comme métrique le nombre d'accès à la liste `L`.

N.B. On peut montrer que cette complexité est optimale pour les algorithmes qui ne peuvent apprendre des informations sur la liste qu'en comparant des éléments d'indices connus avec l'élément recherché.

Exercice 5. *Dictionnaires*

1. Donnez une définition succincte des dictionnaires Python.
2. Rappelez la (ou une) syntaxe permettant de :
   - **i.** créer un dictionnaire vide ;
   - **ii.** créer un dictionnaire avec initialisation (par exemple associant la valeur `True` à la clef `"martre des pins"`) ;
   - **iii.** accéder à la valeur associée à une clef ;
   - **iv.** associer une valeur à une clef (existante ou non) ;
   - **v.** tester si une clef est présente dans un dictionnaire ; n'est pas présente ;
   - **vi.** itérer sur les clefs d'un dictionnaire ;
   - **vii.** obtenir le nombre de couples stockés dans un dictionnaire.

Soit N un entier naturel, on considère $\mathcal{I}_N$ l'ensemble des entiers naturels strictement inférieurs à N.

3. Combien $\mathcal{I}_N$ possède-t'il de sous-ensembles distincts ?

On souhaite représenter et manipuler les sous-ensembles de $\mathcal{I}_N$, à travers cinq fonctions :

- `create(N)` qui crée et renvoie une représentation du sous-ensemble vide de $\mathcal{I}_N$;
- `test(E, i)` qui teste si l'entier `i` est présent dans le sous-ensemble `E` ;
- `add(E, i)` qui ajoute (si nécessaire) l'entier `i` au sous-ensemble `E` ;
- `rem(E, i)` qui supprime (si nécessaire) l'entier `i` du sous-ensemble `E` ;
- `size(E)` qui renvoie le cardinal du sous-ensemble `E`.

4. Proposez une solution à base de `list`.
5. Proposez une solution à base de `dict`. On pourra utiliser la construction (hors programme) `del D[k]` pour supprimer la clef `k` (et sa valeur associée) du dictionnaire `D` ; attention, la clef *doit* être présente dans le dictionnaire.

On considère maintenant un ensemble $\mathcal{S}$ implicite quelconque, dont on souhaite toujours représenter et manipuler les sous-ensembles comme précédemment (où `create` ne prend plus d'argument, et où l'on remplace simplement `i` par un élément `x` quelconque).

6. Proposez une solution à base de `list`.
   Rappel : la fonction-méthode `pop` des `list` est uniquement au programme dans sa version *sans argument*. Vous *ne pouvez pas* utiliser « `L.pop(i)` ».
7. Proposez une solution à base de `dict`.