Cours NSI 1ère: Tris

Dans ce cours, nous allons étudier deux méthodes permettant de trier un tableau. L'étude des tris nous permettra de mettre en évidence certains concepts algorithmiques.

Compétences:

- Écrire un algorithme de tri.
- Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection

Sommaire

1 Introduction, tris de cartes

Vous avez sur votre table, un jeu de cartes non trié. Le but va être de piocher les cartes une par une et de les trier au fur et a mesure de la pioche.
Vous ne pouvez piocher qu'une carte a la fois.
Essayez de trier en utilisant cette méthode.
Décrivez sous forme de pseudo-code ou de liste d'instructions/marche à suivre la méthode que vous utilisez pour trier ces cartes. Votre méthode de tri doit être décrite de la manière la plus détaillée possible.

Un autre tri

Une fois l'activité précédente terminée, mélangez les cartes et prenez les toutes en main, et essayez de trier les cartes en respectant la marche à suivre ci-dessous. Pour la "partie triée" vous pouvez soit faire une autre pile soit, comme quand vous triez des cartes, les mettre d'un coté du paquet.

```
tant qu'il reste des cartes dans le jeu qui ne sont pas trié:
    choisir la première carte non trié.
    Pour chaque carte non trié:
        Si cette carte est plus petite que la précédente:
            sélectionner la carte plus petite.
        fin si
    fin pour
    Déplacer la carte sélectionnée au bout de la partie triée.
fin tant que.
```

2 Tri par insertion

Commençons tout de suite par le pseudo-code du tri par insertion:

```
Rôle: Trier le tableau.
Entrée:
    t: Un tableau non trié
Sortie:
    t: maintenant trié
Pseudo-code:

fonction tri_insertion(t)
    n = taille(t)
    pour i allant de 1 à n-1 # Boucle 1
        x = t[i]
        j = i
        tant que j > 0 et t[j-1] > x # Boucle 2
            t[j] = t[j-1]
            j = j - 1
        fin tant que
        t[j] = x
    fin pour
```

Soit le tableau t = [621, 362, 201, 807, 723]

Continuez sur une feuille a part la simulation de l'algorithme suivant ce modèle:

$i = 1$
$x = 362$
$j = 1$
$j > 0 : VRAI$ et $t[j-1](621) > x(362) : VRAI$
Donc on rentre dans boucle 2
$t[j] = t[j-1](t[j] = 362, t[j-1] = 621)$ donc le tableau devient:
$t = [621, 621, 201, 807, 723] j = 0$ retour à la boucle 2.

$i = 1$
$x = 362$
$j = 0$
$j > 0 : FAUX$ donc on ne rentre pas dans la boucle 2.
$t[j] = x$ Donc le tableau devient:
$t = [362, 621, 201, 807, 723]$ retour à la boucle 1.

$i = 2$
$x = 201$
$j = 2$
$j > 0 : VRAI$ et $t[j-1](621) > x(201) : VRAI$ donc on rentre dans la boucle 2.
$t[j] = t[j-1]$ donc le tableau devient:
$t = [362, 621, 621, 807, 723]$
$j = 1$
retour à la boucle 2.

$i = 2$
$x = 201$
$j = 1$
$j > 0 : VRAI$ et $t[j-1](362) > x(201) : VRAI$ donc on rentre dans la boucle 2.
$t[j] = t[j-1]$ donc le tableau devient:
$t = [362, 362, 621, 807, 723]$
$j = 0$
retour à la boucle 2.

$i = 2$
$x = 201$
$j = 0$
$j > 0 : VRAI$ donc on ne rentre pas dans la boucle 2.
...

3 Tri par sélection

Voici le pseudo-code du tri par insertion:

```
Rôle: Trier le tableau.
Entrée:
    t: Un tableau non trié
Sortie:
    t: maintenant trié
Pseudo-code:

fonction tri_selection(t)
    n = longueur(t)
    pour i de 0 à n - 2 # Première boucle
    min = i
        pour j de i + 1 à n - 1 # Deuxième boucle
            si t[j] < t[min]
                min = j
            fin si
        fin pour
        si min != i
            échanger les valeurs de t[i] et de t[min]
        fin si
    fin pour
```

Sur une feuille à part, simuler le fonctionnement de ce tri tel que vous l'avez fait avec l'algorithme précèdent pour le tableau suivant jusqu'à avoir fait 2 tours de la première boucle:
t = [54, 32, 12, 3]

Expliquer le fonctionnement de cet algorithme de tri:

4 Invariant de boucle

On a pu tester nos algorithmes de tri sur différents tableaux, et il nous semble qu'ils marcheront pour tous les tableaux de nombres qu'on leur donnera. Néanmoins pour en être sur il faudrait prouver le bon fonctionnement de notre algorithme. Pour cela nous allons utiliser un invariant de boucle.
Un invariant de boucle est une propriété qui est vrai a la fois avant de rentrer dans la boucle; a chaque tour de la boucle et a la fin de boucle.
Si on identifie et prouve qu'un invariant existe cela permet en général de prouver qu'un algorithme est correct.
Avant de nous attaquer aux tris que nous avons vu plus tôt dans le cours, nous allons nous intéresser a l'algorithme suivant:

```
Role: Trouver le nombre le plus grand dans le tableau.
Entrée: t: Un tableau d'entiers.
Sortie: indice_val_max: un entier, représentant l'indice dans le
    tableau t de la valeur maximale.
Pseudo code:
valeur_max = t[0]
indice_val_max = 0
i = 0
tant que i < len(t)
    si t[i] < valeur_max
        valeur_max = t[i]
        indice_val_max = i
    fin si
fin tant que
```

Si on veut prouver que cet algorithme fonctionne, une solution serait de trouver un invariant.