fiche de i21 de Mehdi Ben Ahmed

Conception d'algorithme

Définition 1: Invalider un algo Pour montrer qu'un algo n'est pas valide, il suffit de montrer un contre exemple, soit un cas ou l'algorithme ne marcherait pas

Analyse asymptotique

Définition 2: Analyser un algorithme c'est analyser les couts par rapport au temps d'execution, l'espace memoire, et la consommation electrique

Définition 3: le modele random access machine machine hypothetique ou:
- les operands consomment une unite de temps
- les boucles depend du nombre d'iterations et des operation inside
- un read consomme une unite de temps
- la memoire est illimite

l'efficacite d'un algo est defini par une fonction notee C(n) ou T(n), meme si dans un cas reel ca serait plutot note O(n)

exemple:
- recherche d'un element:
  - n cases a tester
  - 5 cases: > 5 tests
  - 10 cases: > 10 tests
- ramasssage de plots:
  - n! chemins a tester
  - 5 plots: 120 chemins possible

la notation est qui suit:
- $\Omega(n)$: meilleur cas
- $O(n)$: pire cas
- $\Theta(n)$: cas moyen

Définition 4: $f(n) = O(g(n))$ il existe une constance c et un entier n_0 tels que $\forall n \geq n_0, f(n) =< cg(n)$

exemples:
- $3n^2 - n + 6 = O(n^2)$ en prenant $c = 3$ et $n_0 = 6$
- $3n^2 - n + 6 = O(n^3)$ en prenant $c = 1$ et $n_0 = 4$
- $3n^2 - n + 6 \neq O(n)$ car $\forall c, cn < 3n^2 - n + 6$ quand $n > c + 1$

Définition 5: $f(n) = \Omega(g(n))$ il existe une constance c et un entier n_0 tels que $\forall n \geq n_0, f(n) \Rightarrow cg(n)$

exemples:
- $3n^2 - n + 6 = \Omega(n^2)$ en prenant $c = 2$ et $n_0 = 2$
- $3n^2 - n + 6 \neq \Omega(n^3) \forall c, 3n^2 - n + 6 < cn^3$ quand $cn > 3$ et $n > 6$
- $3n^2 - n + 6 = \Omega(n)$ en prenant $c = 1$ et $n_0 = 1$

Définition 6: $f(n) = \Theta(g(n))$ il existe une constance c et un entier n_0 tels que $\forall n \geq n_0 c_1 g(n) \leq f(n) \leq c_2 g(n)$
($f(n) - O((g(n)$ et $f(n) = \Omega(g((n)))$)

- $3n^2 - n + 6 = \Theta(n^2)$
- $3n^2 - n + 6 = \Theta(n^3)$
- $3n^2 - n + 6 = \Theta(n)$

Bases d'algo

parcours de tableau

Algos de recherche

Algos de tri

3 types d'algos de tri:
- tri par selection
- tri par propagation
- tri par insertion

tout les algorithmes reposent sur une méthode d'echange d'items basé sur leurs indice:

```
def swap(T, i, j):
    a = T[i]
    T[i] = T[j]
    T[j] = a
```

tri par selection

bien que simple, l'algorithme est considere comme inefficace a cause de son temps d'execution quadratique

$$\Omega(n) = n^2$$
$$O(n) = n^2$$
$$\Theta(n) = n^2$$

il consiste a parcourir une liste, et echanger le plus petit element par le premier, puis d'avencer l'indice du premier jusqu'a finir de parcourir la liste

```
procédure tri_selection(tableau t)
    n ← longueur(t)
    pour i de 0 à n - 2
        min ← i
        pour j de i + 1 à n - 1
            si t[j] < t[min], alors min ← j
        fin pour
        si min ≠ i, alors échanger t[i] et t[min]
    fin pour
fin procédure
```

tri par propagation (Bubble sort)

il a une complexite de n^2, sauf pour le meilleur cas, ou $\Omega(n) = n$ il consiste a echanger les elements qui sont dans le désordre ($n + 1 < n$), a la fin de chaque iteration, le dernier element est le plus grand, donc l'indice est soustre a chaque fois

```
tri_à_bulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               (T[j+1], T[j]) ← (T[j], T[j+1])
```

tri par insertion

il parcours la liste, et a chaque fois que l'algo trouve un nombre inferieur au precedent, il revient en arriere pour le placer correctement

```
procédure tri_insertion(tableau T)
```

```
       pour i de 1 à taille(T) - 1

            # mémoriser T[i] dans x
            x ← T[i]

            # décaler les éléments T[0]..T[i-1] qui sont
plus grands que x, en partant de T[i-1]
            j ← i
            tant que j > 0 et T[j - 1] > x
                     T[j] ← T[j - 1]
                     j ← j - 1

            # placer x dans le "trou" laissé par le
décalage
            T[j] ← x
```