* Aziz Moukrim
  + [aziz.moukrim@utc.fr](mailto:aziz.moukrim@utc.fr)
  + 49 52
* Salles
  + Cours : FA108
  + TD : FA314
  + TP : FB101
* Examens
  + 50% F + 30% M + 20% TP
  + Les TP : il faut y aller, ね
* TD
  + vincent.drevelle@utc.fr
* Spécification d’un algorithme
  + Décrit ce que l’algorithme fait, sans détailler comment il le fait
* Performances croissantes des ordinateurs en rapidité et en taille-mémoire
* Coût de l’algorithme
  + Nombre d’opérations élémentaires => temps
  + Quantité de mémoire requise => espace
  + Coût polynomial désiré ; coût exponentiel à éviter
* Certains problèmes
  + demandent un temps prohibitif
  + demandent un temps prohibitif avec certains algorithmiques
* Définition : Algorithme
  + Ensemble de règles opératoires
  + Résout un problème énoncé au moyen d’un nombre fini d’opérations
* Définition : Instance d’un problème
  + Ensemble des valeurs attendues pour le problème
  + Exemple : pour le problème voyageur de commerce, il s’agit de la liste des villes et des distances entre elles
* Notation O
  + Attention, c’est un ensemble, un ensemble de quoi ? un ensemble de fonctions
  + O(f(n)) = {T(n) / 0 ≤ T(n) ≤ c.f(n), n≥n0}
* Notation Θ
  + Ensemble de fonctions
  + Θ(f(n)) = {T(n) / 0 ≤ c1.f(n) ≤ T(n) ≤ c2.f(n), n≥n0}
* Invariant de boucle A
  + A est vrai avant la boucle
  + Si A et C sont vrai au début d'une itération, alors A est encore vrai à la fin de cette itération
  + m, entier positif, quantité de contrôle si
    - (m≥0) est un invariant de boucle
    - m décroit d'au moins 1 à chaque itération
* Tableaux
  + Type unique
  + Réservé en mémoire
  + Fixe
* Listes chaînées
  + Emplacements utilisés peuvent évoluer
    - On peut en détruire
    - On peut demander à en rajouter
  + Gestion dynamique de la mémoire à gérer proprement
  + Cellules
    - Une clé
    - Un successeur
    - La cellule pointe sur le successeur
    - cle[x] : la clé d'une cellule x
    - succ[x] : c'est la cellule suivante
  + Tete[L]
  + Procédures :
    - Recherche\_liste(L,k)
      * x := tete[L]   
        Tant que x ≠ NIL et cle[x] ≠ k  
         Faire x := succ[x]   
        Retourner(x)
      * Complexité
        + Taille de la liste en argument
        + Pire des cas :

Elément recherché pas dans la liste

O(n)

* + - * + Meilleur des cas :

Elément recherché en première position

Ω(1)

* + - Insérer\_tête\_liste(L,x)
    - Insérer\_queue\_liste(L,y)
      * Complexité
        + O(n)
        + Complexité toujours proportionnelle à n
    - Insérer\_trier\_liste(L,z)
      * x := tete[L] si (cle[z] < cle[x]) ou (x = NIL) alors Inserer\_tete\_liste( L, z)   
        sinon Tant que succ[x] ≠ NIL et cle[z] > cle[succ[x]]

faire x := succ[x]

succ[z] := succ[x]   
succ[x] := z

* + - * Complexité
        + O(n)
    - Pred(x)
      * si x=tete[L]  
         alors retourner(NIL)  
        sinon y:=tete[L]  
         Tant que succ[y]≠x  
         faire y:=succ[y]  
         retourner(y)
      * ne marche que si x est bel et bien dans la liste
      * Complexité
        + O(n)
* Liste chaînée double
  + Existence naturelle du pred
  + Cellules
    - Prédécesseur
    - Clé
    - Successeur
  + Tete[L] et queue[L]
  + Sentinelle
    - Nœud factice
    - Chaînage circulaire
    - Structure régulière même si la liste devient vide
    - Objectif : simplifier l'écriture des procédures précédemment vues !!!
    - On peut essayer de réécrire ces procédures en prenant cela en compte.



* + - Succ(Queue[L])=Tete[L]
    - Pred(Tete[L])=Queue[L]
* Tableau trié
  + Recherche dichotomique (récursive)
    - Dicho(X, T, g, d, res)
    - X : élément recherché
    - T : tableau déjà trié
    - Res : 0 si élément absent, sinon, position de X dans le tableau
    - g et d : gauche et droite (bornes de l'ensemble de recherche)
  + Recherche dichotomique (itérative)
    - Dicho(X, T, res)
    - Possibles invariants (à vérifier) :
      * 1≤g≤d≤n
      * ∀ i ∈ [1, g[, ∀ j ∈ [d, n[, T(i)<X≤T(j)
* Tas

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |

Appliquer Construire\_tas :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 |

* + Nombre d'éléments sur une hauteur k : au plus (récurrence)
  + Trier\_tas
    - On commence par construire un tas
    - Le plus grand élément se trouve à la racine de l'arbre (première case du tableau)