Principe de la recherche

Clé de recherche

- utilisation de l'ordinateur comme support de mémorisation d'informations
- retrouver une information
  - identification = clé de recherche
    - unicité : informations différentes ont des clés différentes
    - extrait des informations elles-mêmes (nom, prénom, n° SS...)
    - imposée par l'utilisateur à l'informaticien: il y a souvent une sémantique externe attachée à cette identification
  - structuration des informations = ensemble
    - adjonction, suppression, recherche

Le type ensemble

type ensemble
avec la_clé : élément → clé
opérations

vide	: → ensemble	**préconditions**
ajouter	: élément × ensemble /→ ensemble	¬ (la_clé(e) E)
supprimer	: clé × ensemble /→ ensemble	c E
recherche	: clé × ensemble /→ élément	c E
_ _	: clé × ensemble → booléen	

sémantique

c E = **si** E = vide **alor**s faux
sinsi E = ajouter(e, E') **alors** c = la_clé(e) c E'
sinon soit E = supprimer(c', E'); c ≠ c' c E'**fsi**

supprimer(c, ajouter(e, E)) = **si** c = la_clé (e) **alors** E
sinon ajouter(e, supprimer(c, E)) **fsi**

recherche(c, ajouter(e, E)) = **si** c = la_clé(e) **alors** e **sinon** recherche(c, E) **fsi**

Représentation par une liste

- liste quelconque

  c E si et seulement s'il existe k tel que c = la_clé(ième(`l` , k))

  ajouter (e, E) = insérer (`l` , k, e)

  supprimer (c, E) = supprimer (`l` , recherche (c, `l`))

  recherche (c, `l`) = **si** estvide (`l`) **alors** 1
  sinsi c = la_clé (premier(`l`)) **alor**s 1
  sinon 1 + recherche (c, fin(`l`)) **fsi**

- adjonction au plus en Θ(n), suivant choix de k
- recherche et suppression en Θ(n)
- variantes: autoadaptatifs

Liste ordonnée

- ordre sur les clés => ordre des éléments dans la liste

  ajouter (e, E) = insérer (l , recherche (c, l), e)

  recherche (c, l, d, f) =
  si d = f **alors** d
  sinon soit m = (d+f)/2;
  si la_clé (ième(l, m)) < c **alor**s recherche (c, l, m +1, f)
  sinon recherche (c, l,d, m) **fsi**

  recherche (c, l) = **si** longueur(l) = 0 **then** 1
  sinsi la_clé(ième(l , longueur(l)) < c **then** longueur(l)+1
  sinon recherche (c, l , 1, longueur(l)) **fsi**

- recherche en Θ(logn)
- adjonction et suppression en Θ(n)

Le tas ou file d'attente à priorité

type tas
avec _ ≤ _ : élément × élément → booléen
opérations

vide	: → tas	**préconditions**
min	: tas /→ élément	¬ (E = vide)
ajouter	: élément × tas /→ tas	¬ (e E)
supprimer-min	: tas /→ tas	¬ (E = vide)
_ _	: élément × tas → booléen	

axiomes

1 e E = **si** E = vide **alor**s faux
sinsi E = ajouter (e1, E1) **alors** e = e1 e E1
sinon soit E = supprimer-min (E1); e ≠ min (E1) e E1 **fsi**

2 min (E) E

3 e E min (E) ≤ e

Implantation du tas (1)

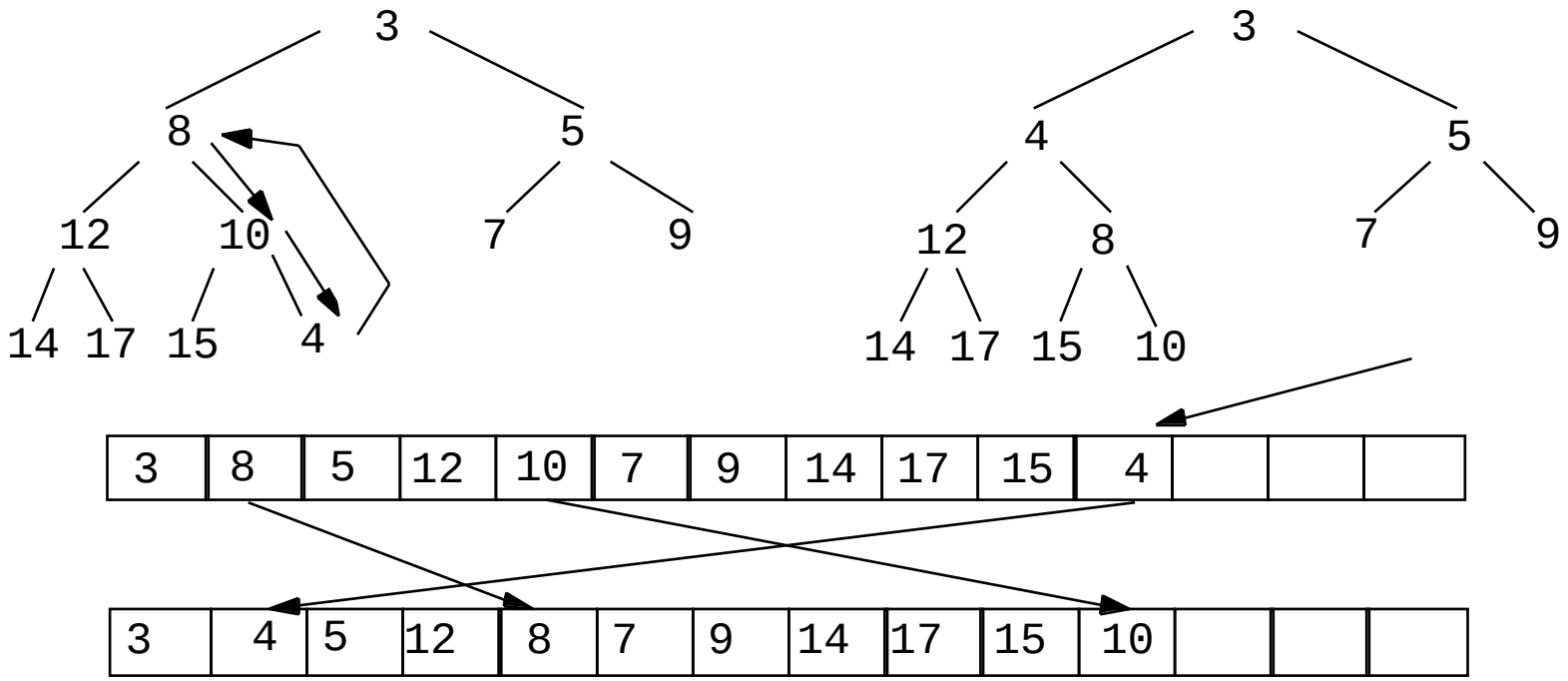
- représentation du tas par un arbre parfait partiellement ordonné
  - rappel arbre parfait:
    - toutes les feuilles sont sur les deux derniers niveaux,
    - l'avant dernier niveau est complet
    - les feuilles du dernier niveau sont le plus à gauche possible
  - arbre parfait s'implante linéairement par niveau
  - partiellement ordonné:
    - tout nœud est plus petit que ses deux fils
- implantation par extension des listes contigues

Implantation du tas (2)

- min : la racine, donc le premier de la liste
- ajouter:
  - ajouter la feuille, donc dernier de la liste
  - réajuster sur le chemin vers la racine
- supprimer-min:
  - mettre la dernière feuille à la racine, donc le dernier en tête
  - réordonner depuis la racine
- toutes opérations en $\Theta(\log n)$

Implantation du tas (3)

- Exemple d'adjonction



Implantation du tas en Ada


```
generic
    with function "<"(U, V : Element ) return Boolean is <>;
package Listes_Contigues.En_Tas is
    procedure Ordonner_Le_Tas (La_Liste: in out Liste; Jusque : in Positive);
        -- Ordonner_Le_Tas transforme le debut de la liste en tas
    function Min_Du_Tas (La_Liste: in Liste) return Element;
    procedure Ajouter_Au_Tas (La_Liste: in out Liste; Val: in Element);
    procedure Supprimer_Du_Tas (La_Liste: in out Liste);
    procedure Ajuster_Le_Tas (La_Liste: in out Liste; En, Jusque : in Positive);
        -- Ajuster_Le_Tas deplace l'element En pour obtenir un tas
end Listes_Contigues.En_Tas;
```

- Remarques:
  - la liste reste organisée en tas si on ne se sert que de cela
  - l'utilisateur a sa part de responsabilité