Algorithmique

Listes

Alexandre PIN

01/10/2010

Définition :
liste d'objets. C'est une suite d'objets O. C'est donné par un entier (n) la longueur de la liste) et une suite e1_En dans O
exemples

- [3,5] liste d'entier
- [a,e,r] liste de lettre
- [3,a] n'est pas une liste (plusieurs types
- [[3][3,4]] liste de liste d'entiers.
- [] ➔ Il s'agit d'une liste légale ‼
- [[]] liste de liste. (ne contenant qu'un singleton : la liste vide

Représentations

Tableau

T[MAX] d'objets
n.

1	7	9	1	5

N=4 => on s'en fou du la dernière case.

Liste en CAML. [3,4,7,9,10,12] x et l dans x ::l.

C : typedef struct tmp_liste{
int element;
struct tmp_liste suite;}

Rec_liste

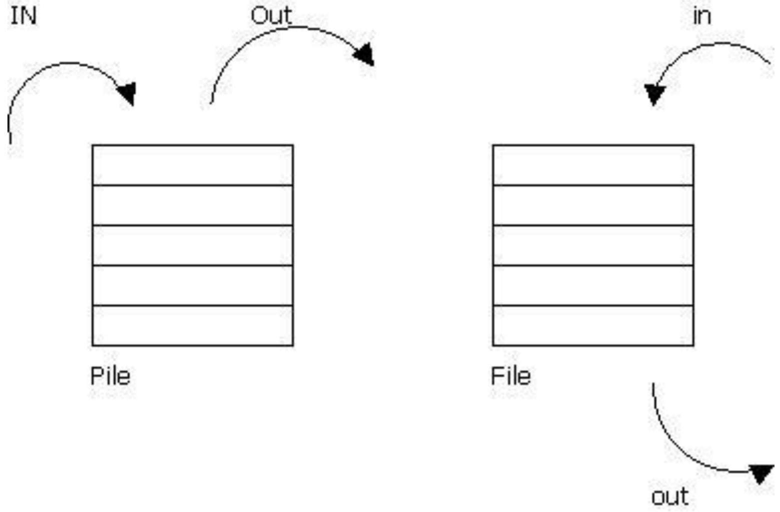
typeDef struct rec_liste * liste

08/10/2010

La Taille d'une liste ?
en général

Souvent, on aura, ou on fera e0…ei oon a des objets de taille Θ(i).
dans ce cas, on pourra prendre |[ei – en]|
Attention ! taille ne peut pas être pour par exemple : liste de graphe, liste de lise. [[1,2,3] est de taille 1.

La différence entre la pile et la file est que dans la pile, les élément entrent et sortent du même coté.



FIFO : first in first out (File)

FILO : First in last out (Pile)

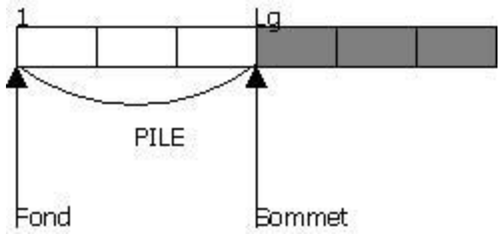
Représentations:

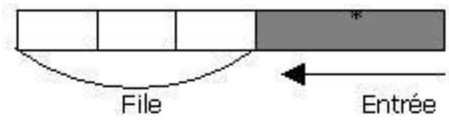
Entier + tableau [1..MAX]

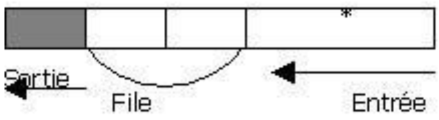
Longueur	[3,4,5]
T[…]	longueur = 3.

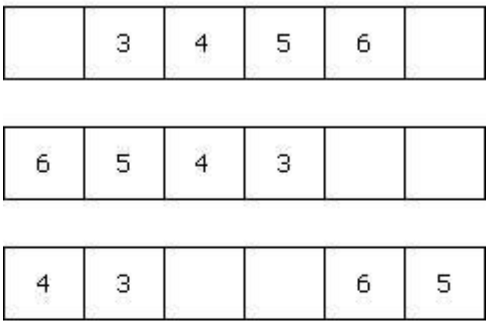
Pile longueur

T[ ]





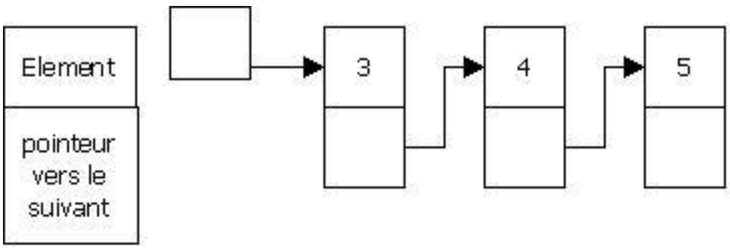




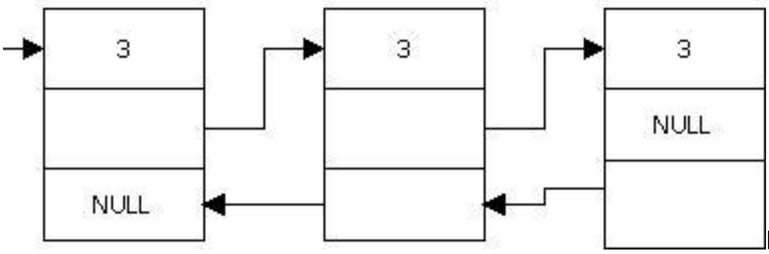
<u>Liste chainées</u>

Un objet structure

Un objet pointeurs vers la structure. LISTE sera le pointeur

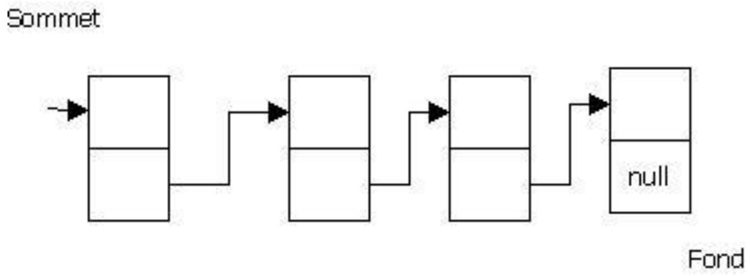


Variante :

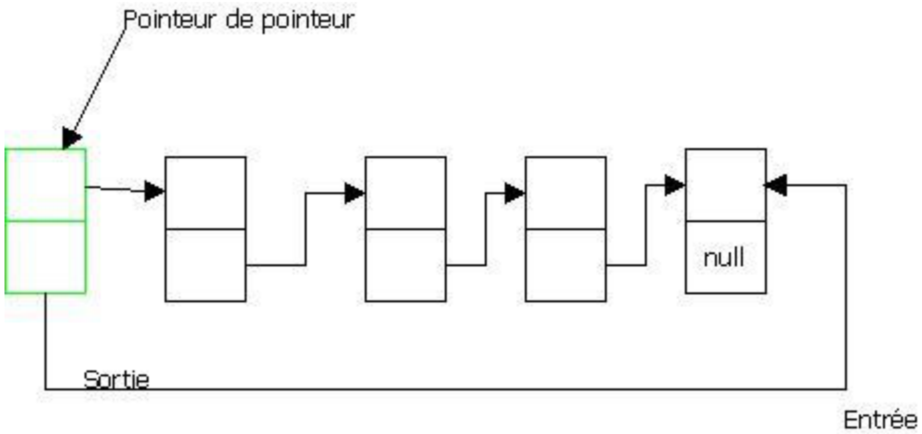


Ici, On peut revennir en arrière.

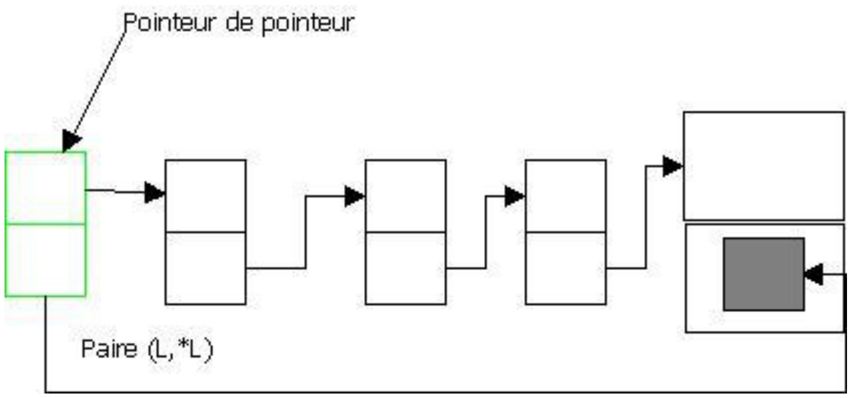
PILE



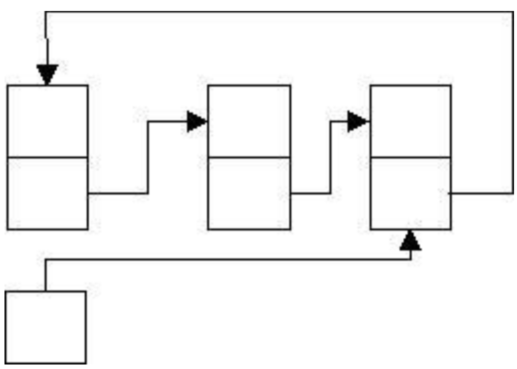
File :



Variantes



Variante 2



Listes Piles

Liste avec les opérations suivantes :

- Initialisation à vide
  InitVide(L)
  L=[]
- Tester le vide
  if L=[]
  estVide(L) :
- Premier(L) le premier élément de la liste ne peut être appelé que si L ≠ []
- Ajoute (X,L)
  Fonction qui rend une liste
  aj(X, [a1-an]) ↦ [Xo1, Xa1-an]
  L n'est pas modifié par cet appel
- Empile (X,L)
  procédure qui ajoute x à L entête
  L doît être une variable, L est in out
- Suite (L)
  [a2..An] (on exclu le premier élément).
  Il faut une liste non vide !
- Dépile
  Procédure qui enlève le 1^{er} élément
  L est une variable, L est INOUT
- PointeurSuite(L)
  Fonction qui rend la suite de L sous forme d'une variable de sorte que si on modifie pointeurSuite(L), cela se répercute sur L.
  Ex : Depile(pointeurSuite(L))
  →enleve le second éléments
- Longueur

```
Longeur(L) : int
	si estVide(L)
		rendre 0 ;
	sinon
		rendre 1 + longCarueur(suite(L))
```

<u>Diviser pour régner</u>

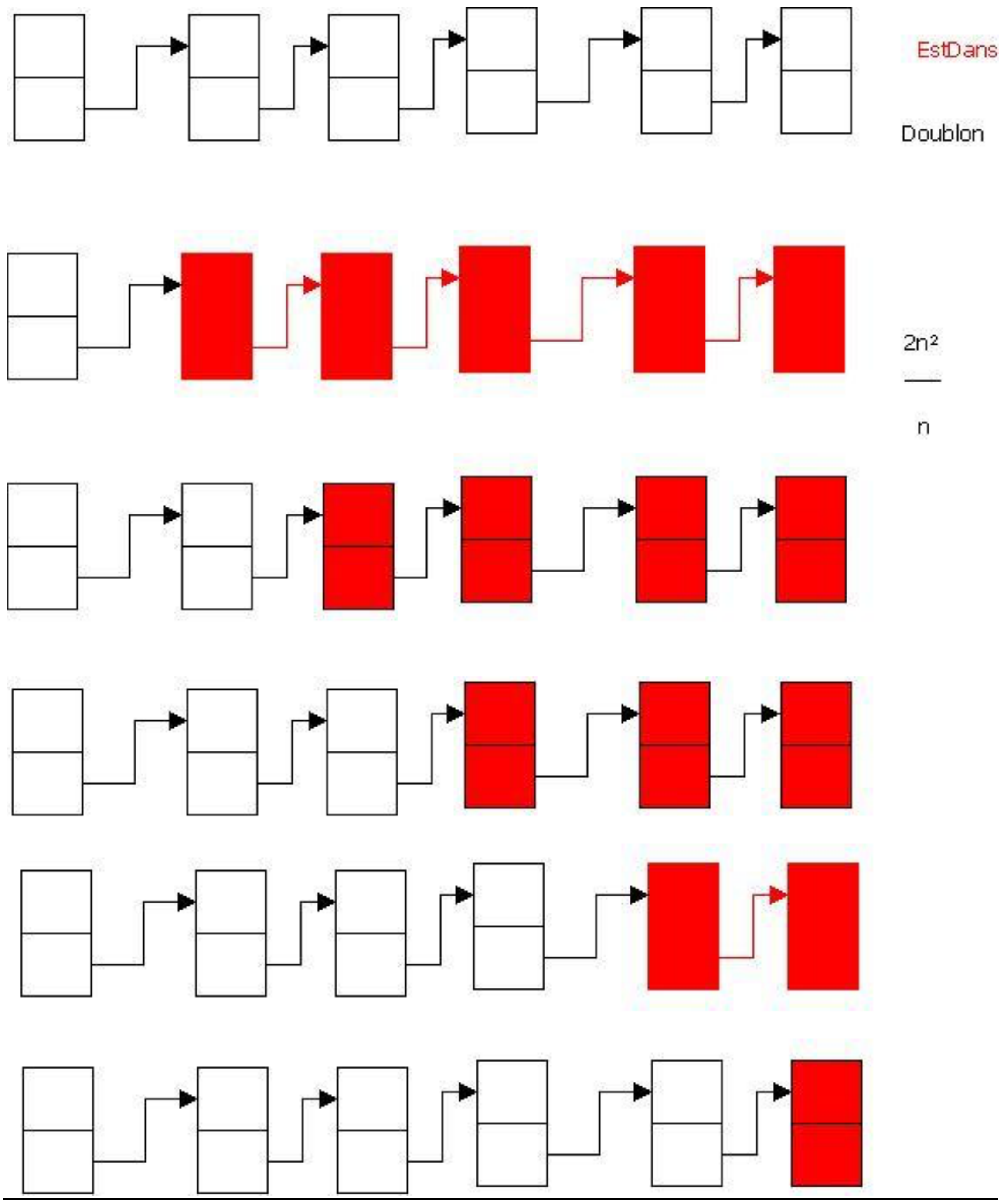
Casser le problème et sous traiter.
cela inclus le récursif.

```
Doublon(L) : booleen
	/* rend vrai si il exite deux termes de la suite qui sont égaux */
	Si estVide(L) alors
		Rendre faux
```

```
	Sinon
		Rendre estDans(premier(L),suite(L))ou doublon(suite(L))
```

```
EstDans(L) : bool
	Si estVide(L)alors
		rendre faux
	sinon
		rendre X= premier(L) ou estDans(x,Suite(L))
```

<u>Complexité de doublons</u>



[... manque une demi heure du cours du 15 octobre ...]

```
Concat (L1 :Liste de Truc, L2 : Liste de Truc) return Liste de Truc
      Si L1 = [] alors
            Rendre L2
      Sinon
            Rendre ajoute(premier(L1)) concat((L1) , L2)
```

[...]

Tableaux

Rechercher le max

de T[1..n] ($n \geq 1$)

```
Posmax (T[1..n] : tableau d'entier) : entier ∈ [1..N]
      Pos ← 1
      Pour k de 2 à N
            si T[k]>T[pos]            // si il y a plusieurs occurrence de max,
                  Alors pos←k         //c'est la dernière qui est stockée.
            Fsi
      fpour
      rendre pos
```

complexité : n-1 comparaisons. **Peut on faire mieux** ?

Pour trouver le max, il faut regarder toutes les cases. Un comparaisons regarde 2 cases.

- ⇨ Tout algo de recherche de maximum fera au moins N/2 coups.
  Donc avec n-1, l'ordre de grandeur est optimal. Et la constante ?

<u>Autre algo : tournoi</u>

Tant que y a $\geq$2 joueurs
On fait des paires, eventuellement il reste un joueur solitaire (si le nombre de ajoueur impair).
On fait joueur un match entre élement de chaque paire, on élimine les perdants.
n-1 comparaisons.

Si $n \neq 2^p$?
si n =23.

1er phase :	11 match,	12 survivants
2e phase	6	6
	3	3
	2	2
	1	1

---

22 ➔ N-1

Au depart, il y N joueurs. A la fin il y a 1 joueur. Chaque match élimine un joueur (ou 0 dans un autre algo, si on fait un match inutil.) Il y a donc evidement n-1 match dans le tournoi.

Tout Algo de recherche de max fera au moins n-1 comparaison.

Recherche dichotomique

T[1..n] est trié de façon croissante.

On cherche x.
on veut rendre

- la position tous T[pos] = x
- ou -1 si x ∉ T[]

Principe

Je regarde au milieu, puis à ¼ ou ¾

A chauqe fois au milieu de l'espace de recherche. Le nouvel espace de recherche sera le coté gauche ou le droit.

```
RechercheDichotomique(x, T[1..n])
      Rendre RD (x, T[], 1, n)
```

```
RD (x, T[], d, f) entier.
      Cherche x ∈ T[d..f]
      Rend d tq T[p]=x si ça existe
            -1 sinon
                  /* Faut il traiter un tableau de longueur nulle ? => non  */
      Si d=f alors
            Si T[d] =x alors
                  Rendre d
            Sinon rendre -1
      Sinon :
            Milieu ← (d+f)                /* correctif ?  OUI (d+f+1)     */
            Si x< T[milieu] alors         /* < ou ≤ ?   <.                */
                  RD(X, T[],d, milieu)          /* ou milieu -1 ?
      */
            Sinon
                  Rendre RD(x, T[], milieu, f)    /* ou milieu +1 ?         */
```

Que doit on rendre si x apparait plusieurs fois dans le même tableau ?

On demande au client ce qu'il veut. Le client répond « position de la dernière occurrence »

Il faut regrouper les 2 cas. On restera « X<T[milieu] »

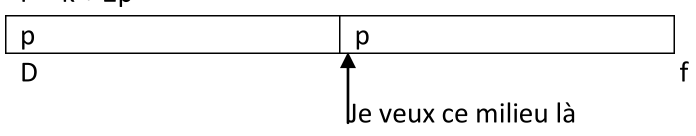
T[Milieu] est il inclus dans le côté gauche ou le côté droit ? Milieu est à droite car si il ya égalité, va vers la droite.

On veut bien couper.

- Si la longueur est paire

  D = k +1

  F = k + 2p



Il faut que milieu = k + p + 1

Division entière → (d+f)/2 =k +p

(D+f +1) /2

(d+f)/2 +1) d + f +2

- Si longueur impaire

On veut

P	p+1

D f

Ou

P+1	p

D f

On veut milieu) k +p + 1 ou k+p +2

<u>Itératif</u>

```
RechercheDichotomique (x, T[])
      d ← 1
      s ← N
      tant que d<f
            milieu ← ———
            si x ∈ T[milieu] alors
                  f← milieu -1
            sinon
                  d ← milieu
            fsi
      ftq
      si T[d] = x alors
            rendre d
      sinon
            rendre -1
```

Complexité en nombre de comparaison

La longueur de l'espace de recherche :

- Au début : N
- A la fin : 1

Chaque comparaison divise par 2 la longueur de l'espace de recherche.

⇨