

# Structures de données linéaires

Tableaux, vecteurs, listes chaînées

Training beOI



OLYMPIADE BELGE D'INFORMATIQUE  
BELGISCHE INFORMATICA-OLYMPIADE

12 février 2016

# Table des matières

Tableaux et variantes

Listes chaînées

File et pile

Comparaison

# Tableau

```
#define MAXN 10000
int tab[MAXN];

int main()
{
    tab[1234] = 100;
    tab[1234]; // 100
    tab[5678]; // 0
}
```

- ▶ Taille fixée à la compilation
- ▶ Accès à un élément arbitraire :  $O(1)$
- ▶ Truc : en-dehors d'une fonction, initialisé à zéro

# Bitset

```
bitset<MAXN> tab; // bool tab[MAXN];  
tab[1234] = true;  
  
bitset<4> b1(string("1100")),  
          b2(string("0101"));  
b1 | b2; // 1101  
b1 & b2; // 0100  
b1 >> 1; // 0110
```

- ▶ Comme un tableau de booléens
- ▶ 8x plus compact
- ▶ Opérations bit-à-bit 64x plus rapides
- ▶ Voir manuel pour la liste des opérations

# Tableau dynamique : fonctionnement

Si plus de place, multiplier par 2

1	
---	--

Capacité = 2

1	2
---	---

1	2	3	
---	---	---	--

Capacité = 4

1	2	3	4
---	---	---	---

1	2	3	4	5			
---	---	---	---	---	--	--	--

Capacité = 8

# Tableau dynamique : en pratique

```
vector<int> vec(8, -1); // initialize to -1
vec[5] += vec[2];      // -2
vec.push_back(5);
vec.push_back(19);
vec.pop_back();
vec.back();            // 5
```

- ▶ Taille augmente et diminue
- ▶ Accès à un élément arbitraire :  $O(1)$
- ▶ Ajout/suppression d'un élément **à la fin** :  $O(1)$
- ▶ Ajout/suppression autre part :  $O(n)$

# Table des matières

Tableaux et variantes

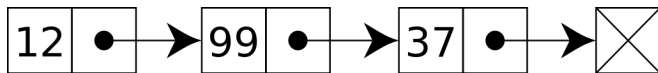
Listes chaînées

File et pile

Comparaison

# Liste chaînée : concept

Des nœuds reliés par des liens (pointeurs)



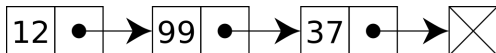
- ▶ Chaque nœud sait où est le prochain
- ▶ Les nœuds ne sont plus côte à côte

```
struct Node
{
    int value;
    Node *next; // link (pointer)
};
```



# Liste chaînée : parcours

Commencer au premier nœud et suivre les liens

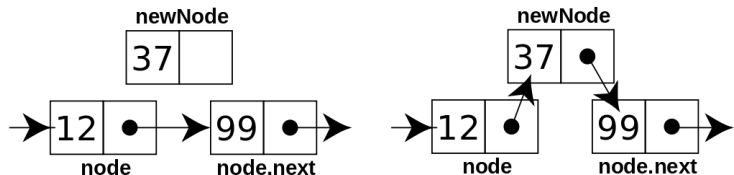


Dans le dernier nœud le lien vaut NULL :

```
Node *cur = start;    // always keep the first node!  
while (cur != NULL)  
{  
    cur->value;        // access value  
    cur = cur->next;   // switch pointer to next  
}
```

# Liste chaînée : ajout

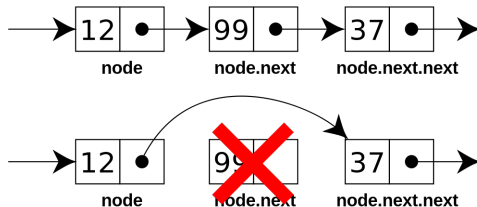
Seulement deux liens à changer



```
void insertAfter(Node *node, Node *new_node)
{
    new_node->next = node->next;
    node->next = new_node;
}
```

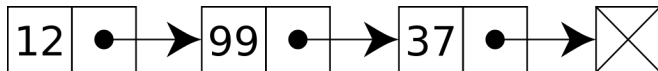
# Liste chaînée : suppression

Changer le lien et supprimer



```
void removeAfter(Node *node)
{
    Node *toRemove = node->next;
    node->next = node->next->next; // bypass
    free(toRemove);
}
```

## Liste chaînée : limitations



Avec une liste (simplement) chaînée :

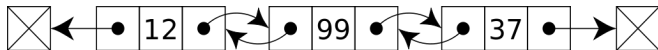
- ▶ Ajout/suppression au début :  $O(1)$
- ▶ Ajout/suppression à une position **donnée** :  $O(1)$

Si on retient la fin aussi :

- ▶ Ajout à la fin :  $O(1)$
- ▶ Suppression à la fin : pas possible,  $O(n)$

# Liste doublement chaînée

Des liens dans les deux sens !



- ▶ Parcours dans les deux sens
- ▶ Suppression à la fin en  $O(1)$
- ▶ Un peu plus lourd

```
struct Node
{
    int value;
    Node *prev, *next; // two pointers
};
```

# Listes chaînée : en pratique

```
list<int> l;  
list<int>::iterator it;  
  
l.push_back(3);    // 3  
it = l.begin();   // ^ points to 3  
l.push_back(4);    // 3 4  
l.push_front(1);   // 1 3 4  
l.insert(it, 2);   // 1 2 3 4 (inserts before 3)  
l.pop_front();     // 2 3 4  
l.pop_back();      // 2 3
```

- ▶ Les `list<>` sont doublement chaînées
- ▶ Retenir les positions avec des `iterator`
- ▶ Tout en  $O(1)$

# Table des matières

Tableaux et variantes

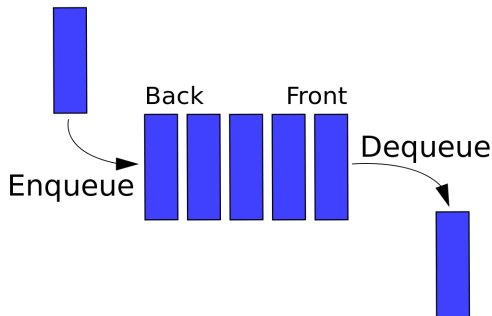
Listes chaînées

File et pile

Comparaison

# File : concept

- ▶ Faire la file dans un magasin
- ▶ On ajoute à la fin, on enlève au début
- ▶ Premier arrivé premier servi (First In First Out)





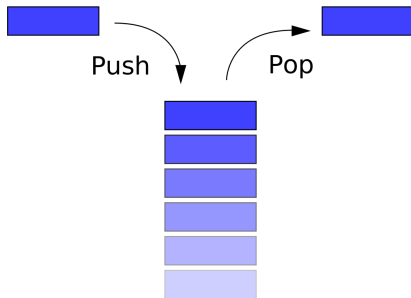
## File : en pratique

- ▶ Ajouter à la fin, enlever au début  $\Rightarrow$  liste chaînée
- ▶ Tout en  $O(1)$
- ▶ On utilise `queue<>` (fait pour ça)

```
queue<int> q;  
q.push(1);  
q.push(2);  
q.front(); // 1  
q.pop();  
q.front(); // 2
```

# Pile : concept

- ▶ Pile de crêpes
- ▶ On ajoute au-dessus, on enlève au-dessus
- ▶ Dernière cuite première mangée (Last In First Out)



# Pile : en pratique

- ▶ Ajouter et enlever à la fin  $\Rightarrow$  liste chaînée *ou* vecteur
- ▶ Tout en  $O(1)$
- ▶ On utilise `stack<>` (fait pour ça)

```
stack<int> q;  
q.push(1);  
q.push(2);  
q.top(); // 2  
q.pop();  
q.top(); // 1
```

# Table des matières

Tableaux et variantes

Listes chaînées

File et pile

Comparaison

# Choix : structures spéciales

Structures pour besoins spéciaux :

- ▶ Ajoute d'un côté et on enlève de l'autre  $\Rightarrow$  **file**
- ▶ Enlève et ajoute d'un même côté  $\Rightarrow$  **pile**
- ▶ Booléens, opérations spéciales (et, ou, shift, ...)  $\Rightarrow$  **bitset**

Sinon, voir slide suivante !

# Choix : tableaux, vecteurs, listes chaînées

“Ajout” = ajout ou suppression

Structure	Indexation	Ajout fin	Ajout milieu
Tableau	$O(1)$	$O(n)$	$O(n)$
Vecteur	$O(1)$	$O(1)$	$O(n)$
Liste chaînée	$O(n)$	$O(1)$	$O(1)$

- ▶ Ajout au milieu nécessaire (rare)  $\Rightarrow$  **liste chaînée**
- ▶ Taille maximale inconnue  $\Rightarrow$  **vecteur**
- ▶ Tous les autres cas  $\Rightarrow$  **tableau** (plus rapide)

## Source des figures

- ▶ <https://commons.wikimedia.org/wiki/File:Singly-linked-list.svg>
- ▶ <https://commons.wikimedia.org/wiki/File:CPT-LinkedLists-addingnode.svg>
- ▶ <https://en.wikipedia.org/wiki/File:CPT-LinkedLists-deletingnode.svg>
- ▶ <https://en.wikipedia.org/wiki/File:Doubly-linked-list.svg>
- ▶ [https://en.wikipedia.org/wiki/File:Data\\_Queue.svg](https://en.wikipedia.org/wiki/File:Data_Queue.svg)
- ▶ [https://en.wikipedia.org/wiki/File:Data\\_stack.svg](https://en.wikipedia.org/wiki/File:Data_stack.svg)