

Structures de données linéaires

Tableaux, vecteurs, listes chaînées

Training beOI



OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

13 avril 2016

Table des matières

Tableaux et variantes

Listes chaînées

File et pile

Choisir la bonne structure

Tableau

```
1 #define MAX_N 10000
2 int tab[MAX_N];
3
4 int main() {
5     tab[1234] = 100;
6     tab[1234]; // 100
7     tab[5678]; // 0
8 }
```

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

Bitset

C++ : bitset

Java : BitSet

```
1 bitset<MAX_N> tab; // bool tab[MAX_N];
2 tab[1234] = true;
3
4 bitset<4> b1(string("1100")),
5             b2(string("0101"));
6 b1 | b2; // 1101
7 b1 & b2; // 0100
8 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

C++ : vector

Java : ArrayList<E>

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

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

Table des matières

Tableaux et variantes

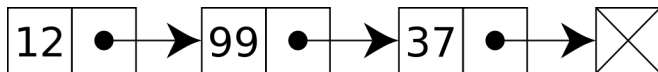
Listes chaînées

File et pile

Choisir la bonne structure

Liste chaînée : concept

Des nœuds reliés par des liens (pointeurs)

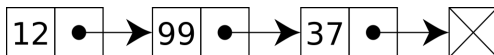


- ▶ Chaque nœud sait où est le prochain
- ▶ Les nœuds ne sont plus côte à côte

```
1 struct Node {  
2     int value;  
3     Node *next; // link (pointer)  
4 };
```


Liste chaînée : parcours

Commencer au premier nœud et suivre les liens

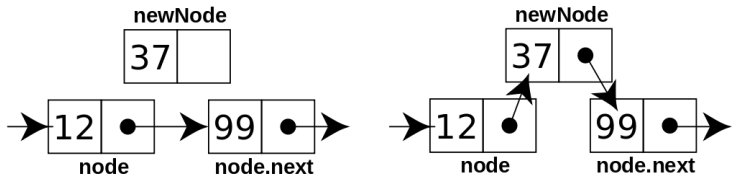


Dans le dernier nœud le lien vaut NULL :

```
1 Node *cur = start;    // always keep the first node!  
2 while (cur != NULL) {  
3     cur->value;        // access value  
4     cur = cur->next;    // switch pointer to next  
5 }
```

Liste chaînée : ajout

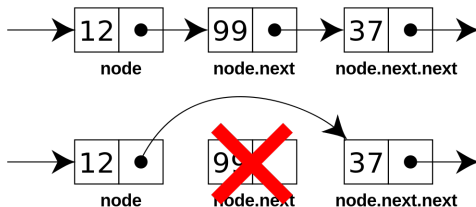
Seulement deux liens à changer



```
1 void insertAfter(Node *node, Node *new_node) {  
2     new_node->next = node->next;  
3     node->next = new_node;  
4 }
```

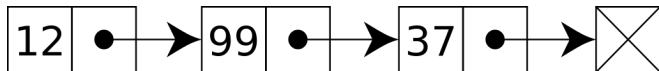
Liste chaînée : suppression

Changer le lien et supprimer



```
1 void removeAfter(Node *node) {  
2     Node *toRemove = node->next;  
3     node->next = node->next->next; // bypass  
4     free(toRemove);  
5 }
```

Liste chaînée : limitations



Avec une liste (simplement) chaînée :

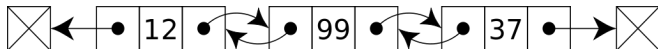
- ▶ Ajout/suppression au début : $\mathcal{O}(1)$
- ▶ Ajout/suppression à une position **donnée** : $\mathcal{O}(1)$

Si on retient la fin aussi :

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

Liste doublement chaînée

Des liens dans les deux sens !



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

```
1 struct Node {  
2     int value;  
3     Node *prev, *next; // two pointers  
4 };
```

Listes chaînée : en pratique

C++ : list

Java : LinkedList<E>

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

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

Table des matières

Tableaux et variantes

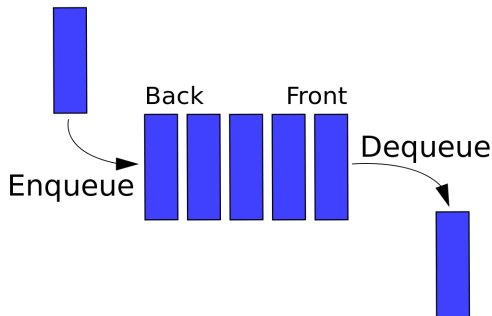
Listes chaînées

File et pile

Choisir la bonne structure

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

C++ : queue

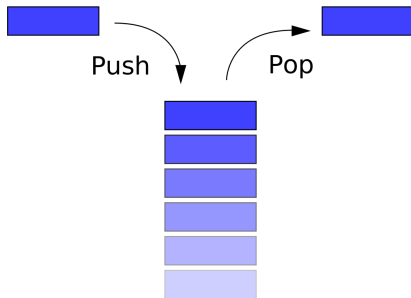
Java : Queue<E>

- ▶ Ajouter à la fin, enlever au début \Rightarrow liste chaînée
- ▶ Tout en $\mathcal{O}(1)$

```
1 queue<int> q;  
2 q.push(1);  
3 q.push(2);  
4 q.front(); // 1  
5 q.pop();  
6 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

C++ : `stack`

Java : `Stack<E>`

- ▶ Ajouter et enlever à la fin \Rightarrow liste chaînée *ou* vecteur
- ▶ Tout en $\mathcal{O}(1)$

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

Table des matières

Tableaux et variantes

Listes chaînées

File et pile

Choisir la bonne structure

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	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Vecteur	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Liste chaînée	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

- ▶ 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_stack.svg