Linear data structures

Arrays, vectors, linked lists

beOI Training

OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

October 22, 2022

Table of contents

Arrays and variants

Linked lists

Queue and stack

Choosing the right structure

Array

```c
#define MAX_N 10000
int tab[MAX_N];

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

- Size fixed at compile time
- Accessing any element: $\mathcal{O}(1)$
- Tip: if declared outside a function, initialised to zero

Bitset

C++ : `bitset`

Java : `BitSet`

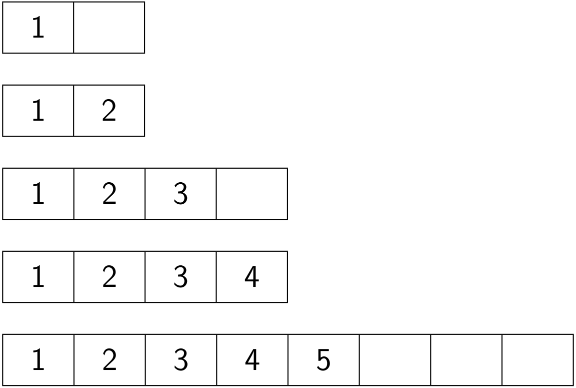
```cpp
bitset<MAX_N> tab; // bool tab[MAX_N];
tab[1234] = true;

bitset<4> b1(string("1100")),
          b2(string("0101"));
b1 | b2; // 1101
b1 & b2; // 0100
b1 >> 1; // 0110
```

- Like an array of booleans
- 8 times more compact
- Bitwise operations 64 times faster
- See manual for the list of operations

Dynamic array: logic

If out of space, double the capacity



1								Capacity $= 2$
1	2							
1	2	3						Capacity $= 4$
1	2	3	4					
1	2	3	4	5				Capacity $= 8$

Dynamic array: in practice

C++ : `vector`

Java : `ArrayList<E>`

```
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
```

- Size increases and decreases
- Accessing any element: $\mathcal{O}(1)$
- Adding/removing an element **at the end**: $\mathcal{O}(1)$
- Adding/removing elsewhere: $\mathcal{O}(n)$

Table of contents

Arrays and variants

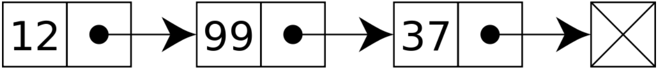
Linked lists

Queue and stack

Choosing the right structure

Linked list: concept

Nodes bound by links (pointers)

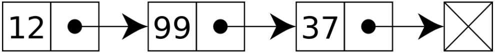


- Every node knows where is the next one
- Nodes are not necessarily next to each other in memory

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

Linked list: iterating over elements

Start at the first node and follow the links

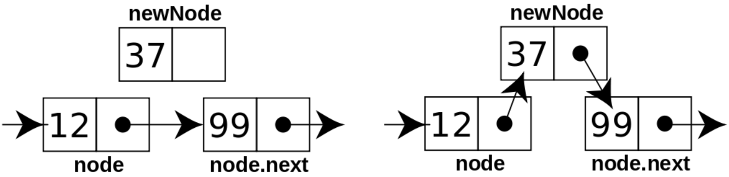


The link in the last node contains NULL:

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

Linked list: adding elements

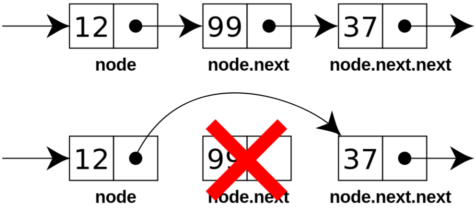
Only two links have to be changed



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

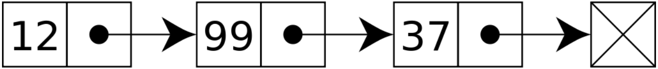
Linked list: removing elements

Change the link and remove the node



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

Linked list: limitations



With a (singly) linked list:

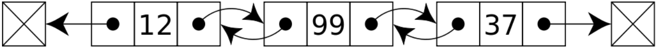
- Adding/removing at the beginning: $\mathcal{O}(1)$
- Adding/removing at a **given** location: $\mathcal{O}(1)$

Operations at the back:

- Adding at the back: $\mathcal{O}(1)$
- Removing at the back: impossible to do directly, $\mathcal{O}(n)$

Doubly linked list

Links in both ways!



- Iteration in both ways
- Removing at the back $\mathcal{O}(1)$
- A bit more memory-heavy

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

Linked lists: in practice

C++ : `list`

Java : `LinkedList<E>`

```
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
```

- `list<>` is doubly linked
- Remember positions using `iterators`
- Everything is $\mathcal{O}(1)$

Table of contents

Arrays and variants

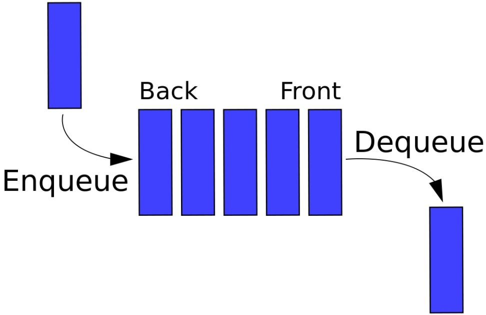
Linked lists

Queue and stack

Choosing the right structure

Queue: concept

- Queuing in a store
- We add at the back, we remove at the front
- "First In First Out"



Queue: in practice

C++ : `queue`

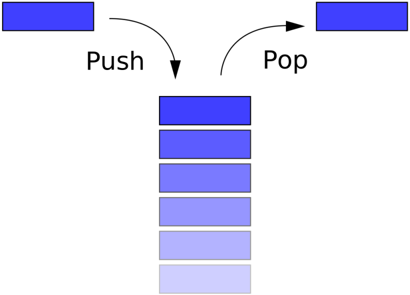
Java : `Queue<E>`

- Adding at the back, removing at the front ⇒ linked list
- Everything is $\mathcal{O}(1)$

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

Stack: concept

- Stack of pancakes
- We add at the top, we remove from the top
- "Last In First Out"



Stack: in practice

C++ : `stack`

Java : `Stack<E>`

- Adding and removing at the back ⇒ liste chaînée *ou vecteur*
- Everything is $\mathcal{O}(1)$

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

Table of contents

Arrays and variants

Linked lists

Queue and stack

Choosing the right structure

Choice: special structures

Structures for special needs:

- Adding from one side and removing from the other ⇒ **queue**
- Adding and removing from the same side ⇒ **stack**
- Booleans, bitwise operations (and, or, shift, ...) ⇒ **bitset**

Otherwise, see next slide!

Choice: arrays, vectors, linked lists

“Add” = adding or removing

Structure	Indexation	Add (back)	Add (middle)
Array	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Vector	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Linked list	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

- Adding in the middle (rare) ⇒ **list**
- Unknown maximal size ⇒ **vector**
- All other cases ⇒ **array** (faster)

Sources of the images

- 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