Lineaire data structuren

Tabellen, vectors en gelinkte lijsten

beOI Training

8 mars 2016

Inhoudstafel

Tabel

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

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

- Grootte van de tabel vooraf bepaald bij de compilatie
- Toegang tot een willekeurig element : $O(1)$
- Elementen worden op 0 geïnitialiseerd

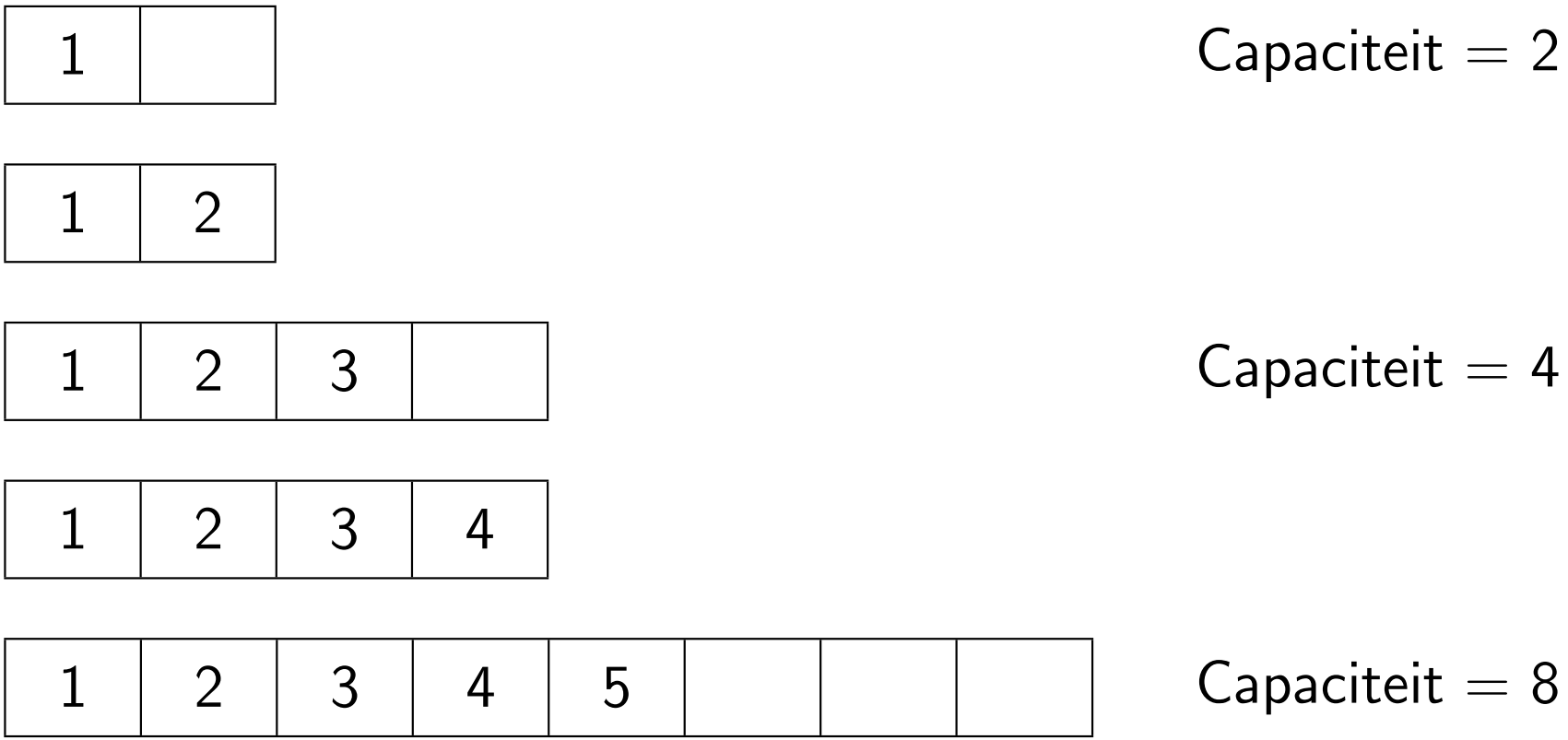
Bitset

```
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
```

- Zoals een tabel van booleans
- 8 keer compacter
- Bit operaties tot 64 keer sneller
- Zie internet voor een volledige lijst van operaties

Dynamische tabel : werking

Als er onvoldoende plaats is, wordt de lengte verdubbeld



Dynamische tabel : in de praktijk

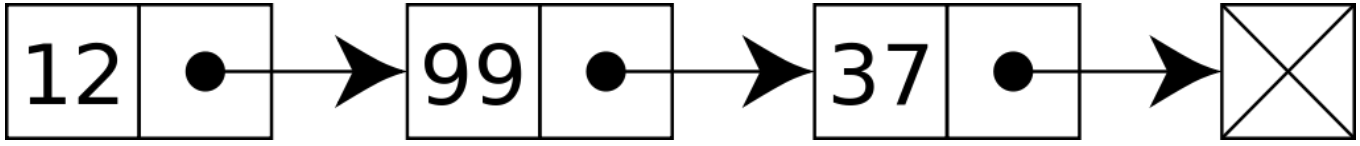
```
vector<int> vec(8, -1); // initialize to -1
vec[5] += vec[2];       // -2
vec.push_back(5);
vec.push_back(19);
vec.pop_back();
vec.back();             // 5
```

- Omvang kan toenemen en afnemen
- Toegang tot een willekeurig element : $O(1)$
- Toevoegen/verwijderen van een element **op het einde** : $O(1)$
- Ergens anders toevoegen/verwijderen : $O(n)$

Inhoudstafel

Gelinkte lijst : concept

Knopen worden gelinkt met wijzers (pointers)

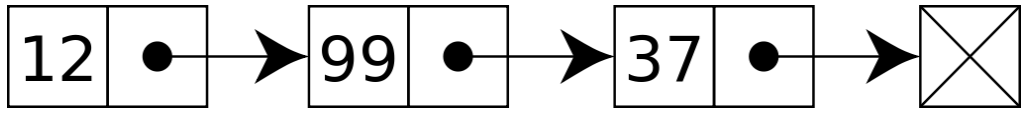


- Elke knoop weet waar de volgende is

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

Gelinkte lijst : doorlopen

Beginnen bij de eerste knoop en de wijzers volgen

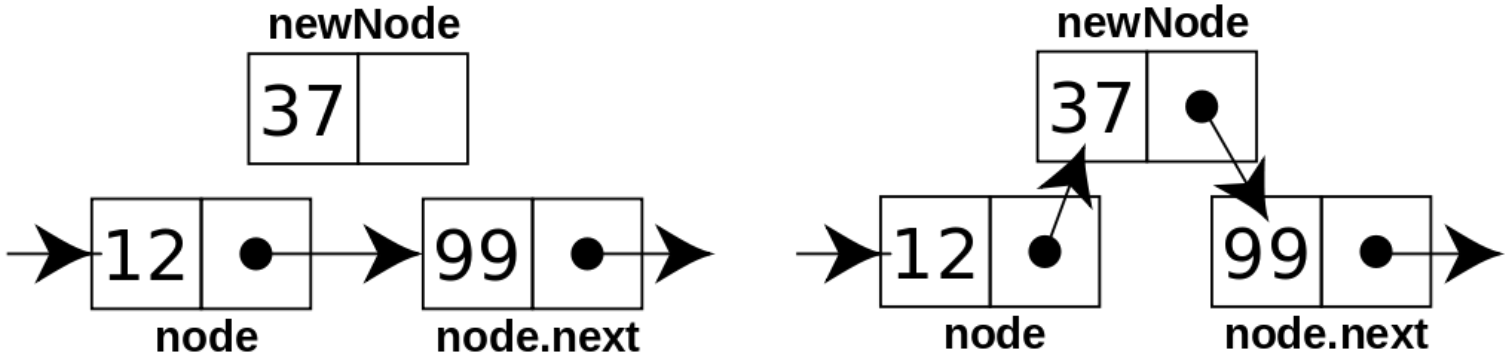


In de laatste knoop heeft de wijzer de waarde NULL :

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

Gelinkte lijst : toevoegen

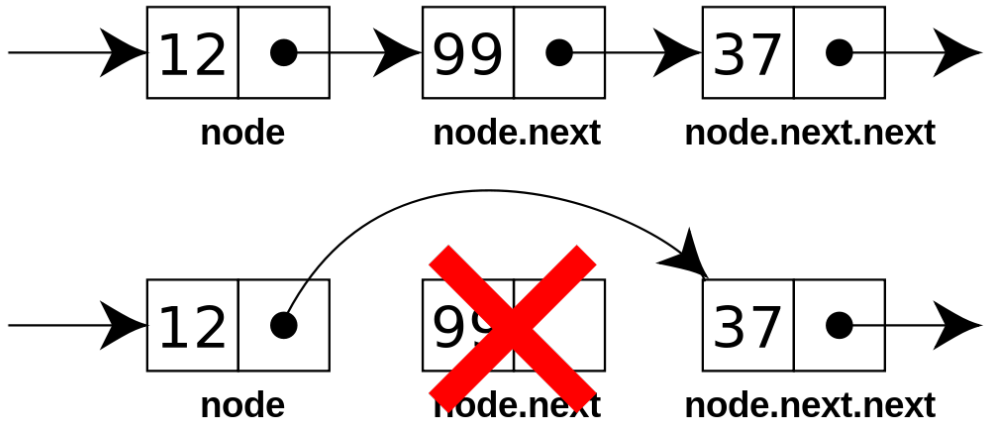
Je moet maar twee wijzers veranderen




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

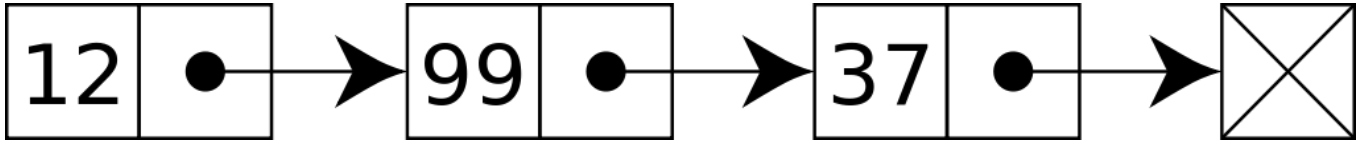
Gelinkte lijst : verwijderen

Verander de wijzer en geef geheugen vrij van de verwijderde knoop




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

Gelinkte lijst : beperkingen



Met een enkel gelinkte lijst :

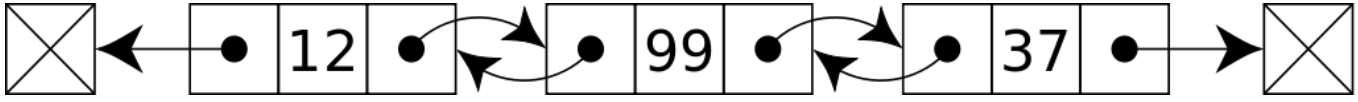
- Toevoegen/verwijderen in het begin van de lijst : $O(1)$
- Toevoegen/verwijderen op een **gegeven** positie : $O(1)$

Als we ook het einde onthouden :

- Toevoegen op het einde : $O(1)$
- Verwijderen op het einde : $O(n)$

Dubbel gelinkte lijsten

Wijzers in de twee richtingen !



- Doorlopen in de twee richtingen
- Verwijderen op het einde van de lijst in $O(1)$
- Iets meer geheugengebruik

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

Gelinkte lijst : in de praktijk

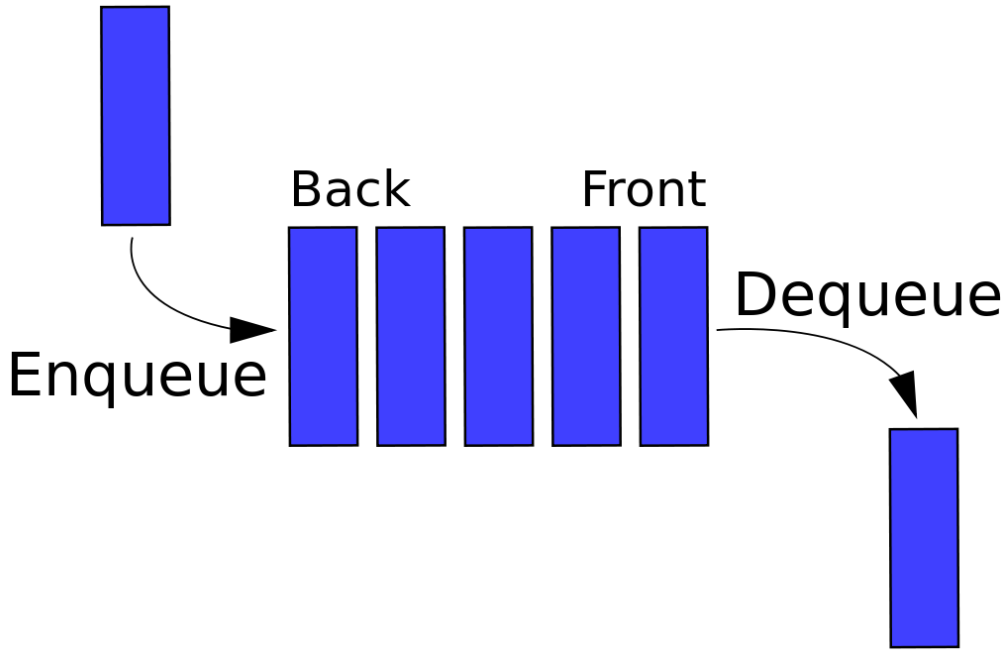
```
list<int> l;
list<int>::iterator it;

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
```

- De data structuur `list<>` is een dubbel gelinkte lijst
- Posities onthouden met `iterators`
- Allemaal in $O(1)$

Inhoudstafel

Rij : concept

- Zoals een rij in een winkel
- We voegen aan het einde dingen toe en halen bij het begin dingen weg
- First In First Out



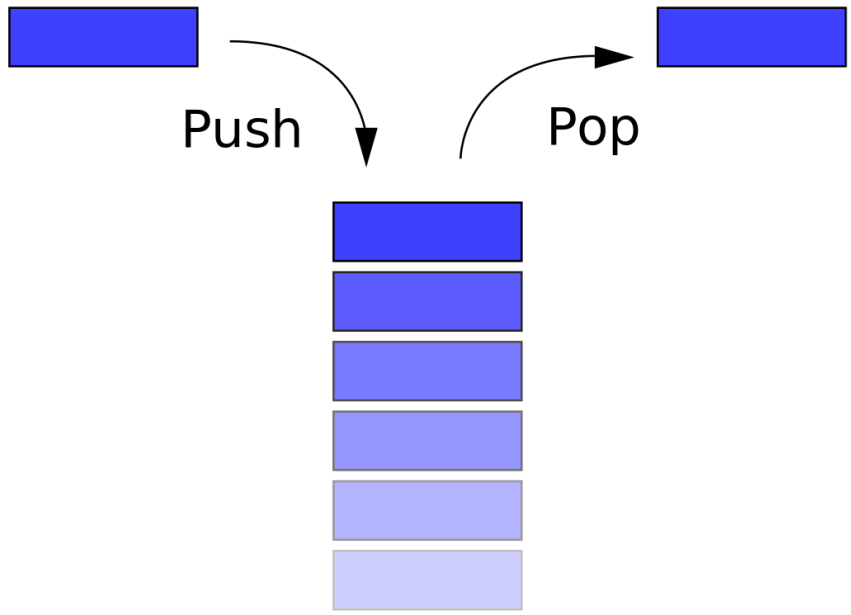
Rij : in de praktijk

- Toevoegen aan het einde, weghalen bij het begin $\Rightarrow$ gelinkte lijst
- Allemaal in $O(1)$
- We gebruiken hiervoor queue<>

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

Stapel : concept

- Zoals een stapel pannekoeken
- We voegen bovenaan toe en halen bovenaan weg
- Laatst gebakken wordt eerst opgegeten (Last In First Out)



Stapel : in de praktijk

- Toevoegen en verwijderen op het einde $\Rightarrow$ gelinkte lijst *of* *vector*
- Allemaal in $O(1)$
- We gebruiken hiervoor `stack<>`

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

Inhoudstafel

Keuze : speciale data structuren

Data structuren voor specifieke noden :

- Aan de ene kant toevoegen en aan de andere kant weghalen $\Rightarrow$ **rij**
- Toevoegen en weghalen aan dezelfde kant $\Rightarrow$ **stapel**
- Booleans, speciale operaties (en, of, shift...) $\Rightarrow$ **bitset**

In het andere geval, zie volgende slide !

Keuze : tabellen, vectors en gelinkte lijsten

"Toevoegen" = toevoegen of verwijderen

Data structuur	Indexatie	Toevoegen einde	Toevoegen midden
Tabel	$O(1)$	$O(n)$	$O(n)$
Vector	$O(1)$	$O(1)$	$O(n)$
Gelinkte lijst	$O(n)$	$O(1)$	$O(1)$

- Toevoegen in het midden nodig (zeldzaam) $\Rightarrow$ **gelinkte lijst**
- Maximale grootte onbekend $\Rightarrow$ **vector**
- Alle andere gevallen $\Rightarrow$ **tabel** (sneller)

Bronnen van de figuren

- 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