Lineaire data structuren

Tabellen, vectors en gelinkte lijsten

beOI Training

13 april 2016

Inhoudstafel

Tabel

```
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 }
```

- Grootte van de tabel vooraf bepaald bij de compilatie
- Toegang tot een willekeurig element: $\mathcal{O}(1)$
- *Buiten functies* worden elementen op 0 geïnitialiseerd

Bitset

C++ : `bitset`

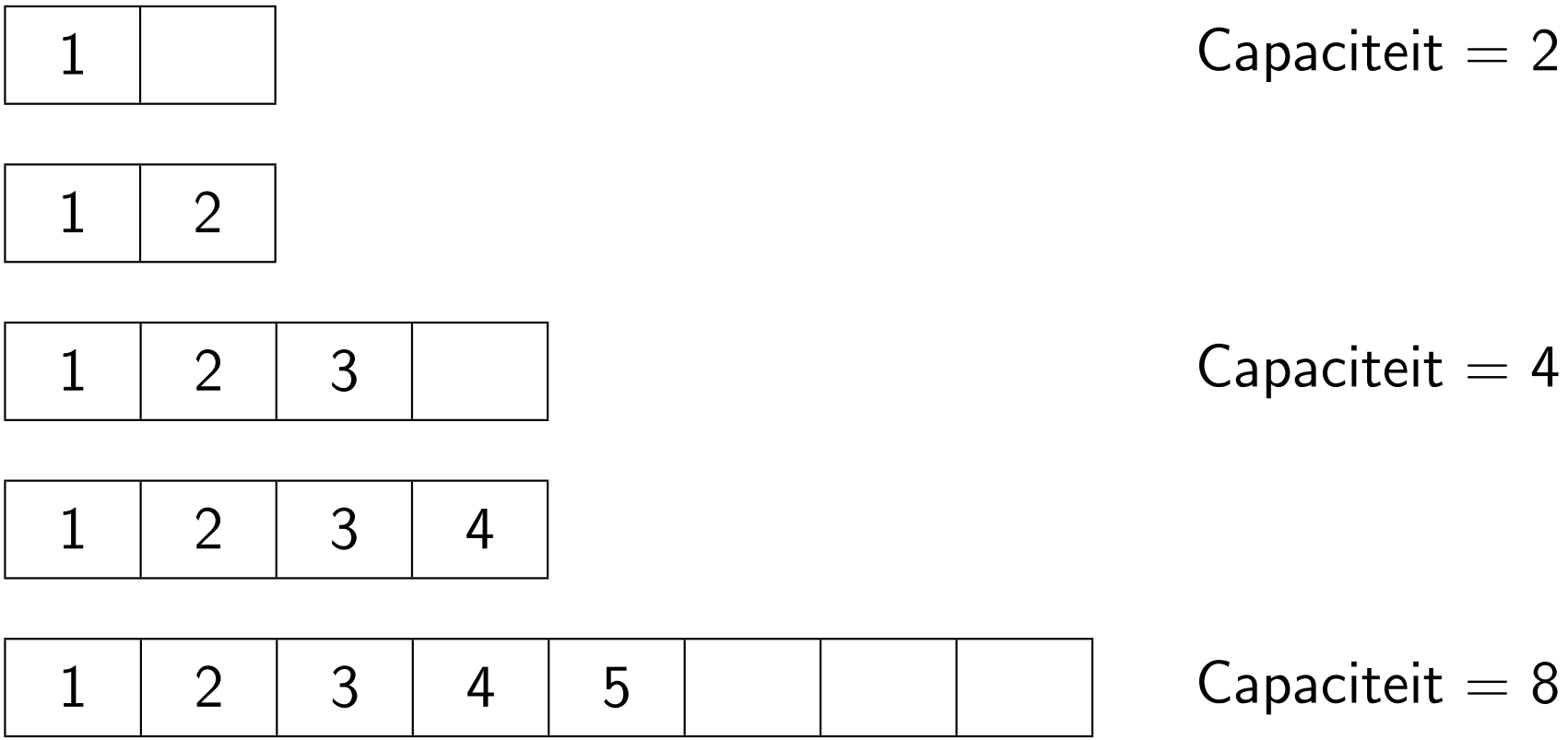
Java : `BitSet`

```
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
```

- 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

C++ : `vector`

Java : `ArrayList<E>`

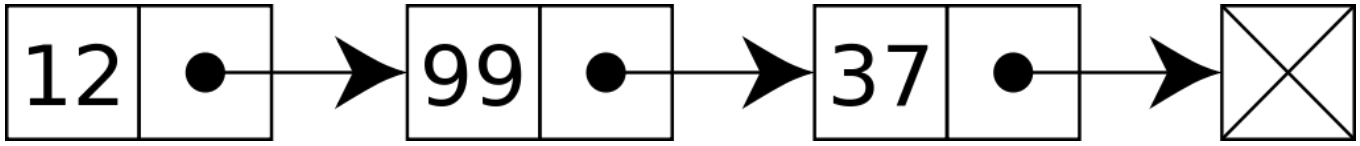
```
1 vector<int> vec(8, -1); // initialize to -1
2 vec[5] += vec[2];       // -2
3 vec.push_back(5);
4 vec.push_back(19);
5 vec.pop_back();
6 vec.back();             // 5
```

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

Inhoudstafel

Gelinkte lijst: concept

Knopen worden gelinkt met wijzers (pointers)

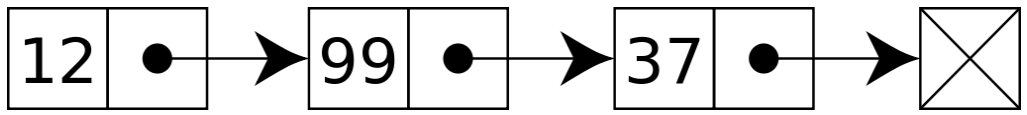


- Elke knoop weet waar de volgende is

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

Gelinkte lijst: doorlopen

Beginnen bij de eerste knoop en de wijzers volgen

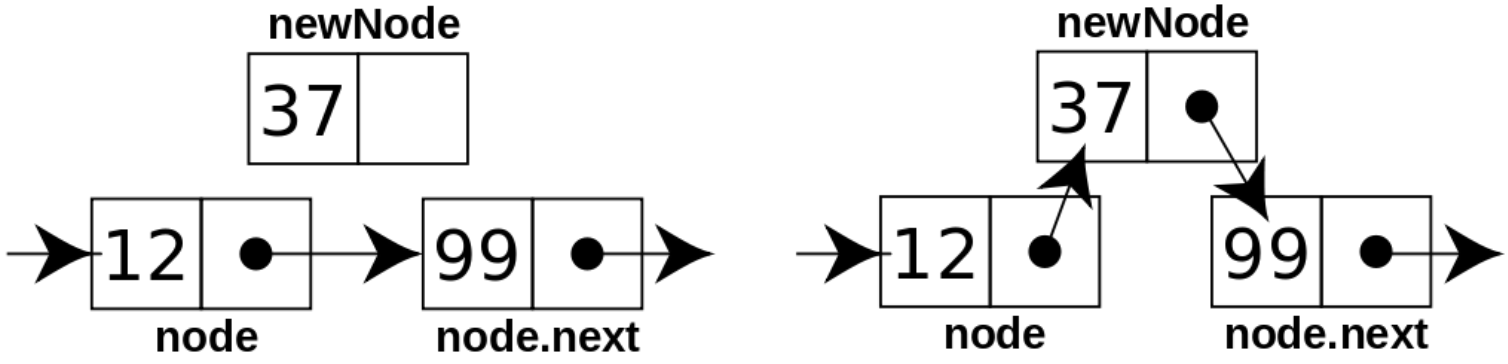


In de laatste knoop heeft de wijzer de waarde NULL:

```
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 }
```

Gelinkte lijst: toevoegen

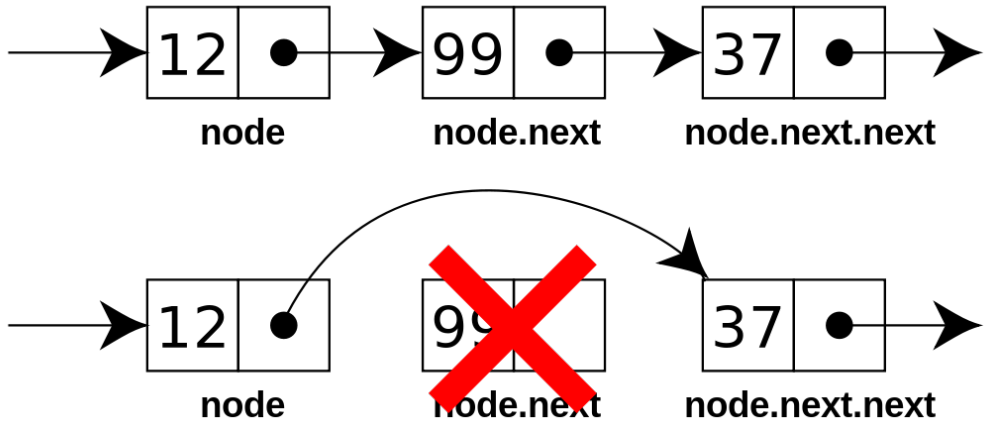
Je moet maar twee wijzers veranderen




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

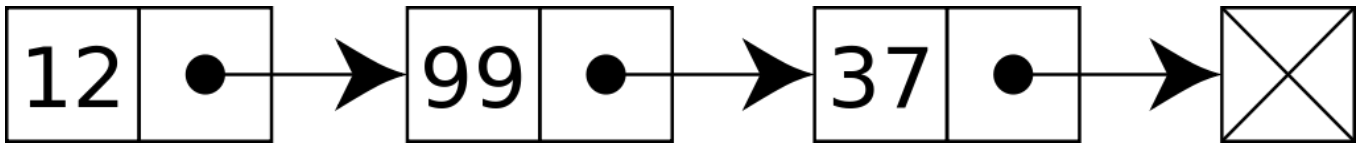
Gelinkte lijst: verwijderen

Verander de wijzer en geef geheugen vrij van de verwijderde knoop




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

Gelinkte lijst: beperkingen



Met een enkel gelinkte lijst:

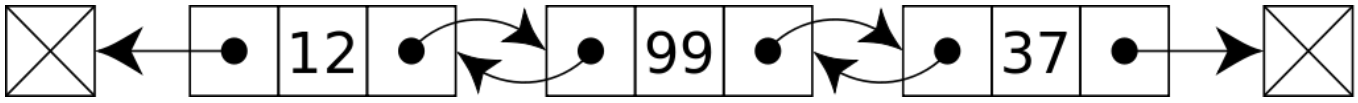
- Toevoegen/verwijderen in het begin van de lijst: $\mathcal{O}(1)$
- Toevoegen/verwijderen op een **gegeven** positie : $\mathcal{O}(1)$

Als we ook het einde onthouden:

- Toevoegen op het einde: $\mathcal{O}(1)$
- Verwijderen op het einde : $\mathcal{O}(n)$

Dubbel gelinkte lijsten

Wijzers in de twee richtingen!



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

```
1 struct Node
2 {
3     int value;
4     Node *prev, *next; // two pointers
5 };
```

Gelinkte lijst: in de praktijk

C++ : `list`

Java : `LinkedList<E>`

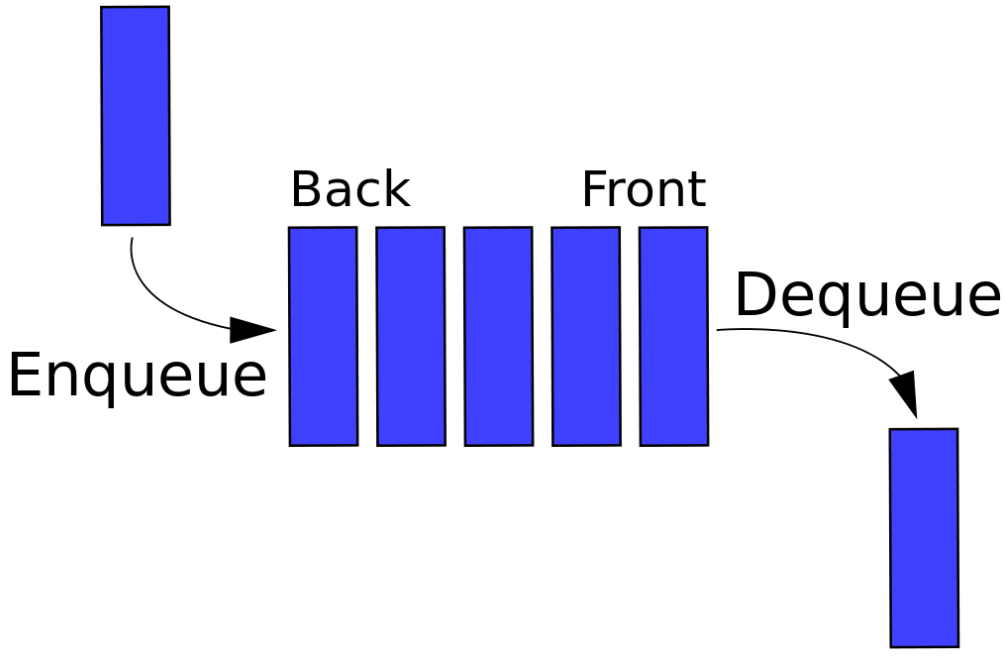
```
1  list<int> l;
2  list<int>::iterator it;
3
4  l.push_back(3);  // 3
5  it = l.begin();  // ^ points to 3
6  l.push_back(4);  // 3 4
7  l.push_front(1); // 1 3 4
8  l.insert(it, 2); // 1 2 3 4 (inserts before 3)
9  l.pop_front();   // 2 3 4
10 l.pop_back();    // 2 3
```

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

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

C++ : `queue`

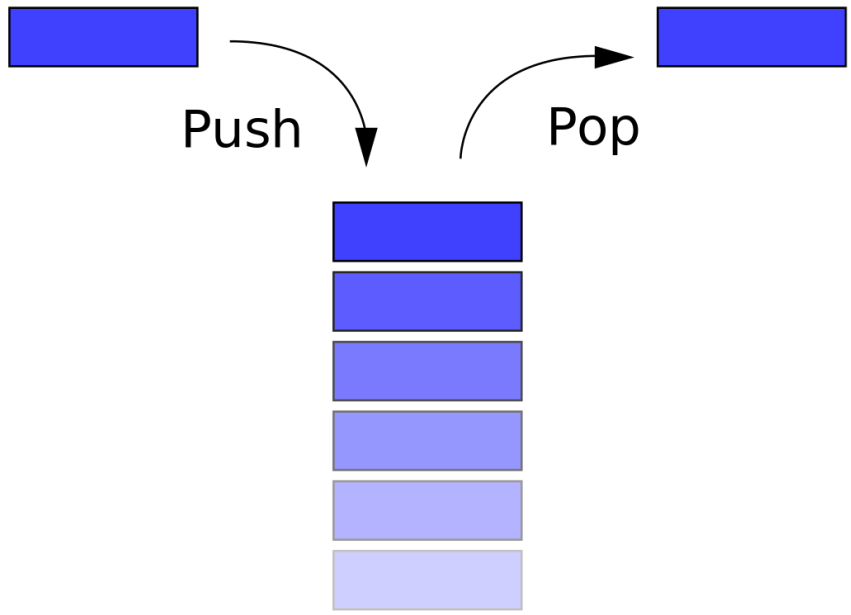
Java : `Queue<E>`

- Toevoegen aan het einde, weghalen bij het begin $\Rightarrow$ gelinkte lijst
- Allemaal in $\mathcal{O}(1)$

```
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

C++ : `stack`
Java : `Stack<E>`

- Toevoegen en verwijderen op het einde $\Rightarrow$ gelinkte lijst *of* *vector*
- Allemaal in $\mathcal{O}(1)$

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

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	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Vector	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Gelinkte lijst	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

- 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