# Algoritmy a Dátové Štruktúry

Jana Katreniaková
katreniakova@dcs.fmph.uniba.sk

# Abstraktné dátové štrukutúry

## Načo sú dátové štruktúry?

- Ukladanie informácií o množine/postupnosti/inom (zatiaľ číselnej)
  - vnútorná organizácia dát
- Potrebné robiť efektívne vzhľadom k operáciám
  - implementácia operácií

# Zoznam

## Abstraktná dátová štruktúra zoznam

- Postupnosť objektov - u nás čísel
- Operácie hľadanie prvku, vloženie prvku, vymazanie prvku, prvý/posledný prvok, predchodca/nasledovník prvku, ...

## Možnosti implementácie

- Kompaktné množiny - polia, polia s dyn. veľkosťou
- Spájané zoznamy

## Kedy použiť ktoré?

- Kompaktné množiny - dopredu známy pribl.počet prvkov (mierne lepšie sú dynamické polia), potrebný rýchly prístup cez index
- Spájané zoznamy - veľa pridávania a vymazávania prvkov

# Zoznam

## Abstraktná dátová štruktúra zoznam

- Postupnosť objektov - u nás čísel
- Operácie hľadanie prvku, vloženie prvku, vymazanie prvku, prvý/posledný prvok, predchodca/nasledovník prvku, ...

## Možnosti implementácie

- Kompaktné množiny - polia, polia s dyn. veľkosťou
- Spájané zoznamy

## Kedy použiť ktoré?

- Kompaktné množiny - dopredu známy pribl.počet prvkov (mierne lepšie sú dynamické polia), potrebný rýchly prístup cez index
- Spájané zoznamy - veľa pridávania a vymazávania prvkov

# Zoznam

## Abstraktná dátová štruktúra zoznam

- Postupnosť objektov - u nás čísel
- Operácie hľadanie prvku, vloženie prvku, vymazanie prvku, prvý/posledný prvok, predchodca/nasledovník prvku, ...

## Možnosti implementácie

- Kompaktné množiny - polia, polia s dyn. veľkosťou
- Spájané zoznamy

## Kedy použiť ktoré?

- Kompaktné množiny - dopredu známy pribl.počet prvkov (mierne lepšie sú dynamické polia), potrebný rýchly prístup cez index
- Spájané zoznamy - veľa pridávania a vymazávania prvkov

# Zoznam pomocou kompaktných množín

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: potrebné alokovať dopredu – ake veľké?
  - malé = nezmestí sa
  - veľké = nehospodárnosť
  - Riešenie – realokácia ak sa minie resp. je voľné keď je vhodne nastavené, tak amortizovane dobré
- Vector (resp. ekvivalent v inom jazyku) - robí realokáciu za nás

## Operácie

- Vkladanie: na koniec - jednoduché O(1), inde nutný presun O(N)
- Vyberanie: pozor aby nezvnikli diery O(N)
- Min, Max, Member: prejde celé pole O(N)

# Zoznam pomocou kompaktných množín

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: potrebné alokovať dopredu – ake veľké?
  - malé = nezmestí sa
  - veľké = nehospodárnosť
  - Riešenie – realokácia ak sa minie resp. je voľné keď je vhodne nastavené, tak amortizovane dobré
- Vector (resp. ekvivalent v inom jazyku) - robí realokáciu za nás

## Operácie

- Vkladanie: na koniec - jednoduché O(1), inde nutný presun O(N)
- Vyberanie: pozor aby nezvnikli diery O(N)
- Min, Max, Member: prejde celé pole O(N)

# Zoznam pomocou kompaktných množín

```
ARRAYINSERTPOS(x,pos,&A)
1  N ← length(A); length(A) + +;
2  for i←N downto pos
3  do A[i+1]←A[i];
4  A[pos]←x;
```

```
ARRAYMEMBER(x,&A)
1  N ← length(A);
2  for i← 0 to N-1
3  do if x = A[i]
4        then return i;
5  return −1;
```

```
ARRAYDELETE(x,&A)
1  pos ← Member(x, A); N ← length(A);
2  for i← pos to N-1
3  do A[i] ← A[i + 1];
4  length(A) − −;
```

# Zoznam pomocou spájaných zoznamov

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Implementácia pomocou pointrov
  - struct node  int data; node *next
- Alokácia vždy jedného prvku
  - newnode = (node *) malloc(sizeof(node));
- Explicitne uložený pointer na začiatok
  - node *head;

## Operácie

- Vkladanie/Vyberanie na miesto určené pointrom: O(1)
- Vyberanie hodnotou: potrebujeme nájsť O(N)
- Min, Max, Member: prejde celý zoznam O(N)
- Nalsedovník O(1) ale predchodca O(N)

# Zoznam pomocou spájaných zoznamov

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Implementácia pomocou pointrov
  - struct node  int data; node *next
- Alokácia vždy jedného prvku
  - newnode = (node *) malloc(sizeof(node));
- Explicitne uložený pointer na začiatok
  - node *head;

## Operácie

- Vkladanie/Vyberanie na miesto určené pointrom: O(1)
- Vyberanie hodnotou: potrebujeme nájsť O(N)
- Min, Max, Member: prejde celý zoznam O(N)
- Nalsedovník O(1) ale predchodca O(N)

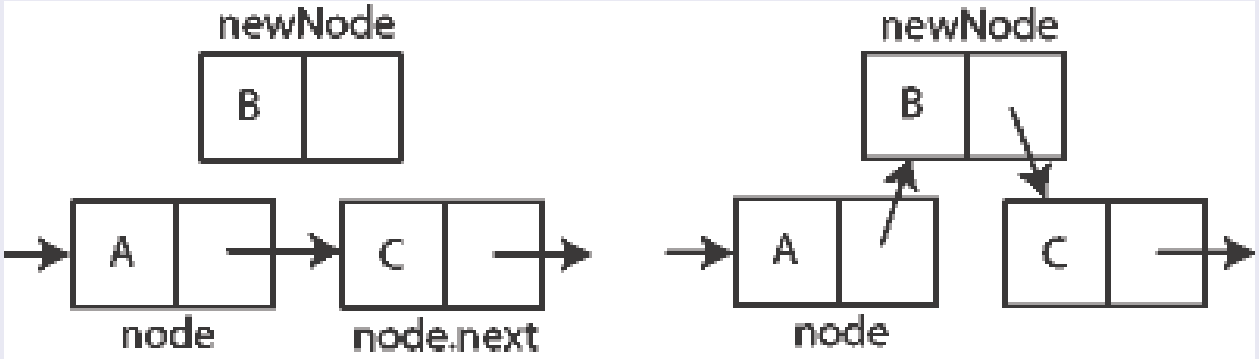
# Zoznam pomocou spájaných zoznamov

```
LISTMEMBER(x,&A)
1  node cur = head;
2  while (cur->next != NULL)
3  do if (cur->data == x)
4        then return cur;
5     cur = cur->next;
6  return -1;
```

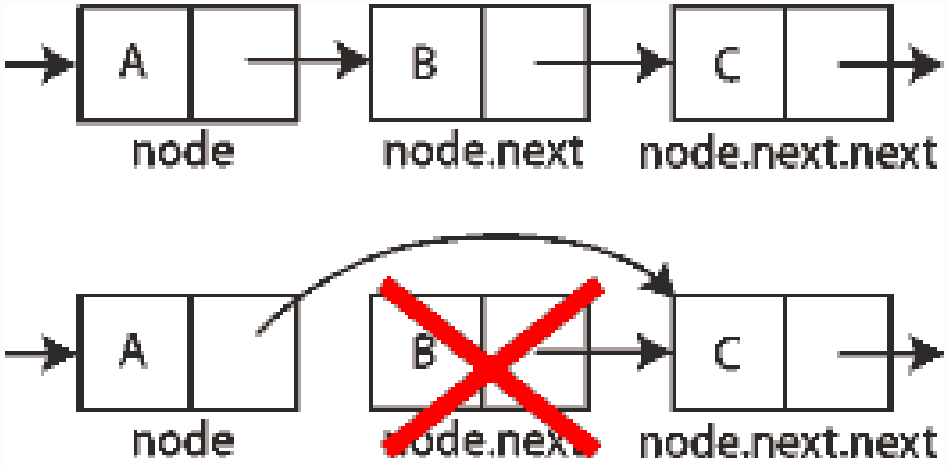
```
INSERTHEAD(x,&A)
1  node *newnode;
2  newnode = (node*)malloc(sizeof(node));
3  newnode->data = x;
4  newnode->next = Head;
5  Head = newnode;
6  length(A)++;
```

# Zoznam pomocou spájaných zoznamov

## Insert

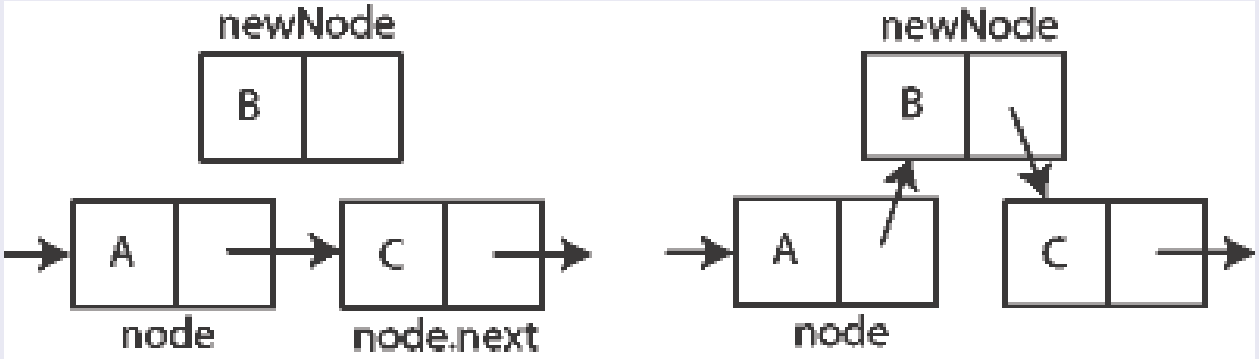



## Delete

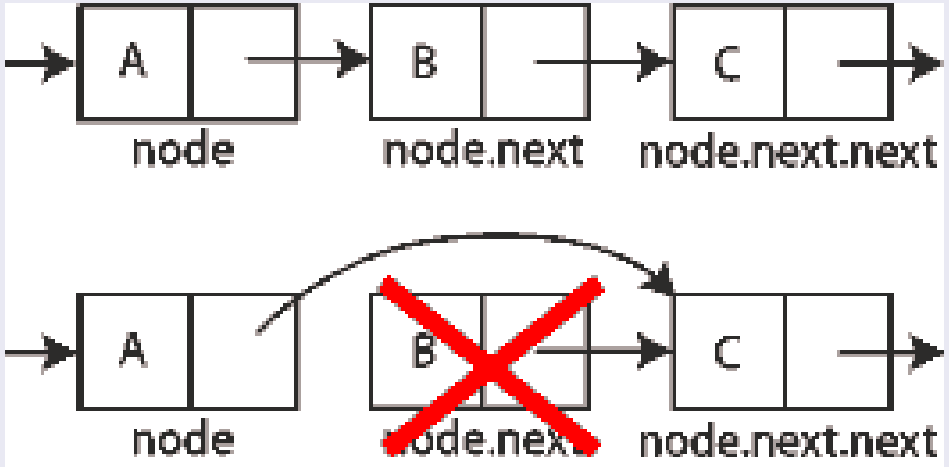


# Zoznam pomocou spájaných zoznamov

## Insert




## Delete



# Zásobník - stack

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: opäť problém s alokáciou
- Vector resp. v niektorých jazykoch priamo Stack
- Spájané zoznamy

## Operácie

- Push: vloženie prvku na vrch zásobníka
- Pop: vyberanie prvku z vrchu zásobníka
- Empty?

# Zásobník - stack

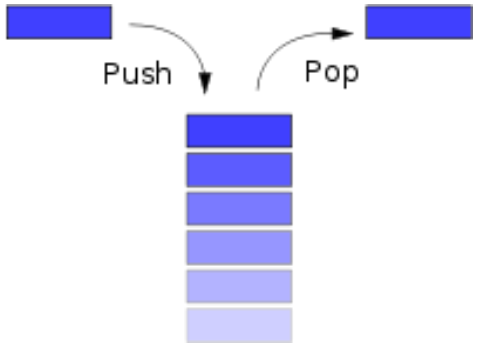
## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: opäť problém s alokáciou
- Vector resp. v niektorých jazykoch priamo Stack
- Spájané zoznamy

## Operácie

- Push: vloženie prvku na vrch zásobníka
- Pop: vyberanie prvku z vrchu zásobníka
- Empty?

# Zásobník pomocou poľa




PUSH(x,&A)
1 $Top++;\ A[Top] \leftarrow x;$

POP(&A)
1 **if** $(Top = 0)$
2 **then return** $Err;$
3 $Top--;$
4 **return** $A[Top + 1];$

# Fronta

## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: opäť problém s alokáciou
- Vector resp. v niektorých jazykoch priamo Queue
- Spájané zoznamy

## Operácie

- Enque: vloženie prvku na koniec fronty
- Dequeue: vyberanie prvku zo začiatku fronty
- Empty?

# Fronta

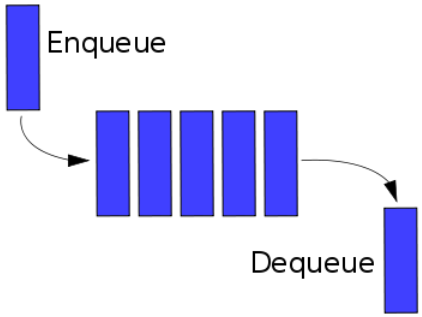
## Kam ukladáme?

- Veľkosť: explicitne uložená!
- Pole: opäť problém s alokáciou
- Vector resp. v niektorých jazykoch priamo Queue
- Spájané zoznamy

## Operácie

- Enque: vloženie prvku na koniec fronty
- Dequeue: vyberanie prvku zo začiatku fronty
- Empty?

# Fronta pomocou poľa




PUSH(x,&A)
1 $A[End] \leftarrow x$;
2 $End \leftarrow (End + 1) mod N$;
3 **if** $(Front == End)$
4 **then return** $Err$;

DEQUEUE(&A)
1 **if** $(Front == End)$
2 **then return** $Err$;
3 **return** $A[Front]$;
4 $Front \leftarrow (Front + 1) mod N$;

# Fronta a Stack pomocou spájaných zoznamov

- Keď už viete spraviť zoznam iste zvláadnete aj jednoduchšie operácie
- V jednom prípade úplne triviálne
- V druhom sa dá, ale hodilo by sa mať smerník nie iba na nasledovníka (na cvičení alebo prednáške uvidíme riešenie)