Prioritní fronta, halda (heap), řazení

Co je prioritní fronta?

Definována operacemi

- vlož prvek
- vyber největší (nejmenší) prvek

Proč pf ?

Rozhraní:

```
class PF {
  // ADT rozhrani
  PF();                          // vytvoření prázdné prioritní fronty
  boolean jePrazdna();           // test, je-li prázdná
  void vloz(Prvek);              // vložení prvku
  Prvek vybermax();              // výběr největšího prvku
}
```

další možná operace – nalezení, přečtení největšího prvku

Implementace ADT prioritní fronta pomocí pole

Myšlenka: pole je uspořádáno vzestupně
1. **vybermax** odebere poslední prvek :-)
2. **vlož** – větší prvky posuneme doprava o jednu pozici :-(

```
class PF {
  private int[] pf;
  private int pocet;
  final int maxN=10;

  PF() {
    pf = new int[maxN];
    pocet = 0;
  }

  boolean jePrazdna() {
    return (pocet == 0);
  }

  void vloz(int klic) {
    if (pocet == 0)
      pf[0] = klic;
    else {              // hledame index vlozeni i
      int i = pocet;
      while (i > 0 && pf[i - 1] > klic) {
        pf[i] = pf[i - 1];
        --i;
      }
      pf[i] = klic;
    }
    pocet++;
  }

  int vybermax() {
    return pf[--pocet];
  }
}
```

Jiná myšlenka: pole prvků není uspořádáno
1. **vlož** – přidej na konec pole :-)
2. **vybermax** – najdi největší :-(

```
class PF {
  private int[] pf;
  private int pocet;
  final int maxN=10;

  PF() {
    pf = new int[maxN];
    pocet = 0;
  }
  boolean jePrazdna() {
    return (pocet == 0);
  }
  void vloz(int klic) {
    pf[pocet++] = klic;
  }
  int vybermax() {
    int max = 0;                  //index max prvku
    for (int i = 1; i < pocet; i++)
      if (pf[max] < pf[i]) max =i;
    int t = pf[max];              //vymenime max a
    pf[max] = pf[pocet - 1];      //posledni
    pf[pocet - 1] = t;
    return pf[--pocet];
  }
}
```

Implementace ADT prioritní fronta pomocí spojového seznamu

Myšlenka: pomůže neuspořádaný seznam?
vlož – přidej na začátek seznamu :-)
vybermax – najdi největší :-(

```
class PF {
  private class Prvek {
    int klic;
    Prvek dalsi;
    Prvek predch;

    Prvek() {
    }

    Prvek(int klic) {
      this.klic = klic;
      this.dalsi = null;
      this.predch = null;
    }
  }

  private Prvek hlavicka;

  PF () {
    hlavicka = new Prvek();
    hlavicka.dalsi = hlavicka;
    hlavicka.predch = hlavicka;
  }

  boolean jePrazdna() {
    return (hlavicka.dalsi == hlavicka.dalsi.dalsi);
  }
```

```
  void vloz(int klic) {
    Prvek novy = new Prvek(klic);
    novy.dalsi = hlavicka.dalsi;
    novy.predch = hlavicka;
    hlavicka.dalsi.predch = novy;
    hlavicka.dalsi = novy;
  }

  int vybermax() {
    Prvek x = hlavicka.dalsi;
    for (Prvek t = x.dalsi; t != hlavicka;
                                          t = t.dalsi)
      if (x.klic < t.klic) x = t;
                    //x ukazatel na max prvek
    int max = x.klic;
    x.predch.dalsi = x.dalsi;
    x.dalsi.predch = x.predch;
    return max;
  }
}
```

// opravit eKnihu

Myšlenka: pomůže uspořádaný seznam?
vybermax – vyber prvek ze začátku seznamu :-)
vlož – vlož před první menší prvek nebo na konec :-(

Složitost:

	vlož	vyber největší prvek	najdi největší prvek
uspořádané pole	*N*	1	1
uspořádaný seznam	*N*	1	1
neuspořádané pole	1	*N*	*N*
neuspořádaný seznam	1	*N*	*N*

Jakou implementaci zvolíme, potřebujeme-li v aplikaci často zjistit největší prvek a zřídka vložit prvek?

trpělivý (lazy) přístup	vs.	netrpělivý (eager) přístup
neuděláme co nemusíme a máme práci později	vs.	uděláme hned co můžeme a později máme klid
(pouze vložíme)		*(vložíme a uspořádáme)*

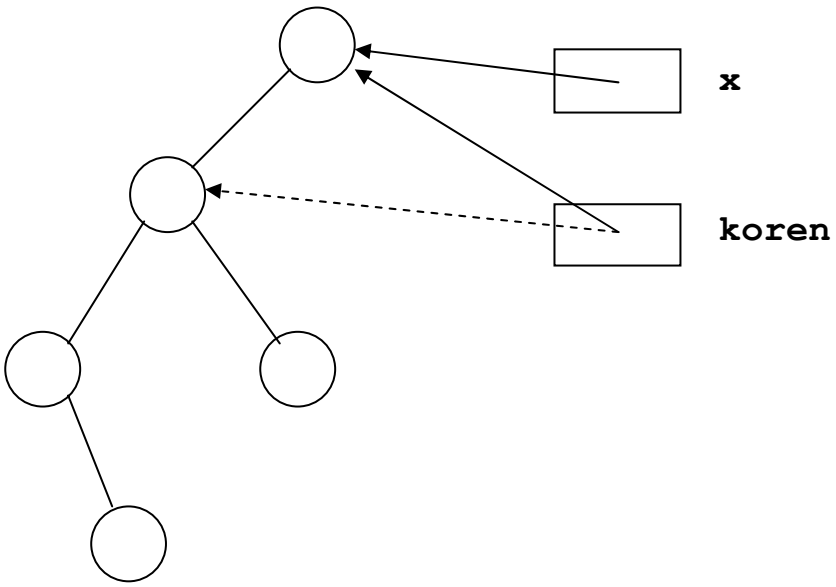
Implementace prioritní fronty pomocí BVS

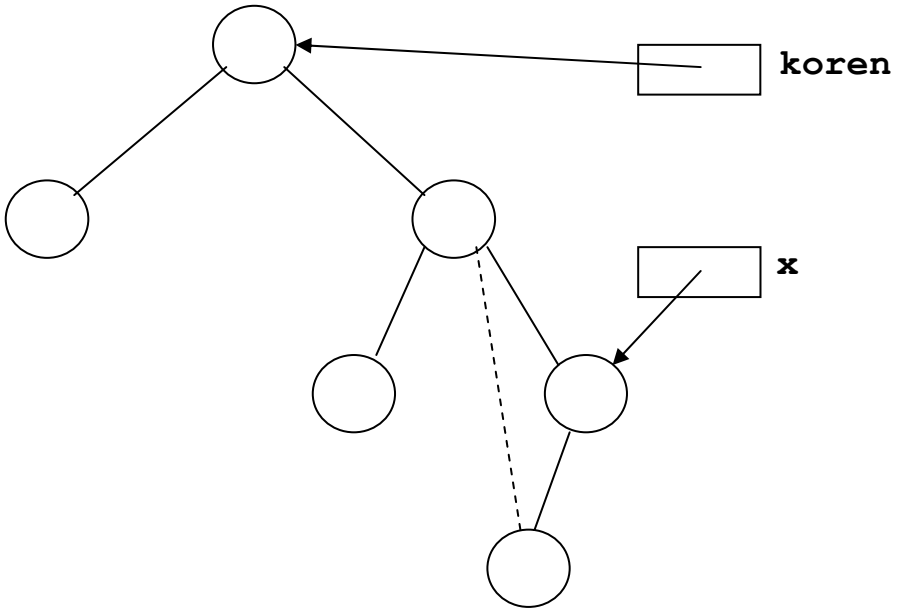
Myšlenka: operace vložení a nalezení prvku se zadaným klíčem byly ***O(h)***, kde *h* je výška stromu

vybermax:

```
int vybermax() {
  DVrchol x = koren;
  DVrchol predch = null;

  while (x.pravy != null) {
    predch = x;
    x = x.pravy;
  }
  if (x == koren)
    koren = koren.levy;
  else
    predch.pravy = x.levy;
  return x.klic;
}
```





-operace vložení, nalezení i výběru největšího prvku jsou *O(h)*
-nejhorší případ *h=N-1*
-průměrný případ $h=1,39log_2N$

Umíme zlepšit ?

Halda (heap)

Myšlenka: *N* prvků můžeme uložit v úplném binárním stromě s výškou

$$h=\Theta(\log_2 N)$$

- všechny úrovně jsou zaplněny, poslední zleva do počtu prvků

vlastnost haldy - klíč v každém vrcholu je větší nebo roven klíčům v jeho následnících, pokud je má

- kořen je největší prvek

Halda je úplný binární strom s vlastností haldy reprezentován pomocí pole.

- přesněji max-halda, obdobně min-halda

nalezení největšího prvku je ***O(1)***
- první prvek pole

vybermax:

vybereme kořen (první prvek pole) a nahradíme ho posledním prvkem

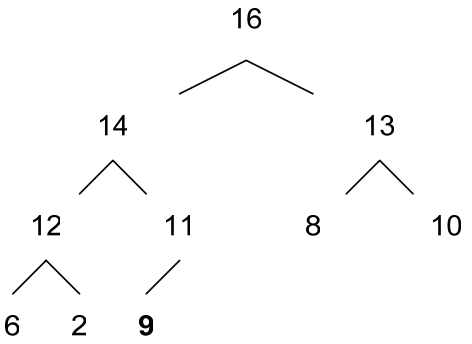
- strom zůstal úplným binárním stromem

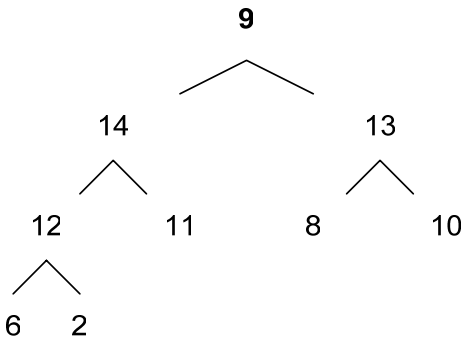
- mohla být porušena vlastnost haldy

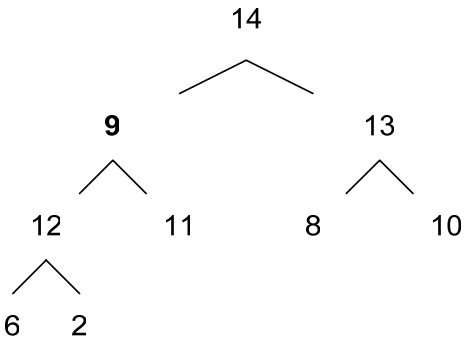
- obnovení vlastnosti haldy: nový kořen vyměníme s větším z jeho následníků, ..., dokud není obnovena vlastnost haldy

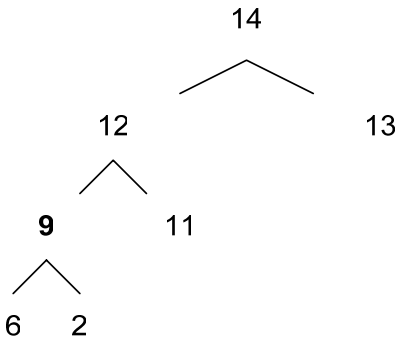
```
// kořen haldy má index 1
// poslední prvek má index pocet
// metoda dolu() začne obnovu haldy od indexu k
// pro vybermax bude k=1

private void dolu(int k, int pocet) {
  // levy následník má index 2k
  while (2*k <= pocet) {
    int j = 2*k;
  // exituje-li pravý následník j < pocet,
  // j bude index většího z obou následníků
    if (j < pocet && pf[j] < pf[j+1]) j++;
    if (pf[k] >= pf[j]) break;    // obnoveno
    vymen(k,j);
    k = j;
  }
}
```


16
14
13
12
11
8
10
6
2
9


9
14
13
12
11
8
10
6
2


14
9
13
12
11
8
10
6
2


14
12
13
9
11
6
2

vloz:

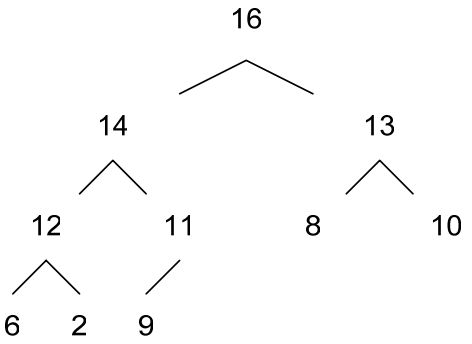
přidáme prvek na konec pole

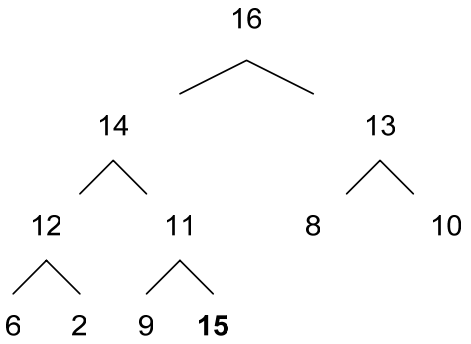
- mohla být porušena vlastnost haldy

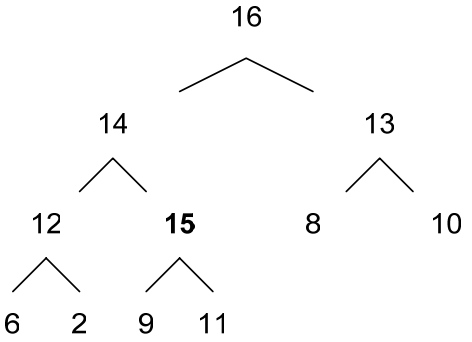
- obnovení vlastnosti haldy: vyměníme ho s předchůdcem, ..., dokud není obnovena vlastnost haldy

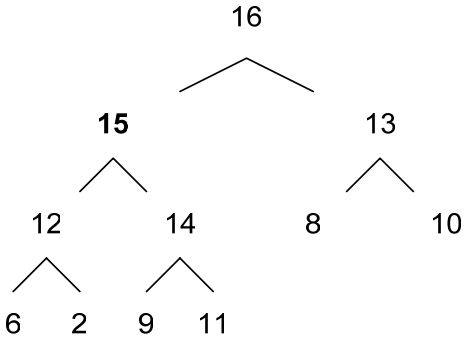
```
// metoda nahoru() začne obnovu haldy od indexu k
// pro vloz bude k=pocet


private void nahoru(int k) {
  // předchůdce má index k/2
  while (k > 1 && pf[k/2] < pf[k]) {
     vymen(k, k/2);
     k = k/2;
  }
}
```


16
14
13
12
11
8
10
6
2
9


16
14
13
12
11
8
10
6
2
9
15


16
14
13
12
15
8
10
6
2
9
11


16
15
13
12
14
8
10
6
2
9
11

```
class PF {
  private void dolu ... ;
  private void nahoru ... ;
  private void vymen ... ;
  private int[] pf;
  final int maxN = 10;
  private int pocet;

  PF() {
    pf = new int[maxN + 1];
    pocet = 0;
  }

  boolean jePrazdna() {
    return pocet == 0;
  }

  void vloz(int klic) {
    pf[++pocet] = klic;
    nahoru(pocet);
  }

  int vybermax() {
    vymen(1,pocet);
    dolu(1, pocet - 1);
    return pf[pocet--];
  }
}
```

vybermax potřebuje nejvíce $2\log_2 N$ porovnání
vlož potřebuje nejvíce $\log_2 N$ porovnání

operace jsou ***O(log N)***

vytvoření haldy s N prvky postupným vkládáním prvků operací **vlož** v nejhorším případě je:

$$\log_2 N + \ldots + \log_2 2 + \log_2 1 < N \log_2 N$$

Řazení a prioritní fronta

```
class RazeniPole {
  //ADT rozhrani
  void nactiPrvek(int)
  void tiskPole()
  void razeniPF()
}
```

Implementace metod **`nactiPrvek()`** a **`tiskPrvku()`**

```
class RazeniPole {
  private int[] pole;
  private int pocet;
  final int maxN;

  RazeniPole() {
    pole = new int[maxN];
    pocet = 0;
  }

  void nactiPrvek(int klic) {
    pole[pocet] = klic;
    pocet++;
  }

  void tiskPole() {
    for(int i = 0; i < pocet; i++)
      Sytem.out.print(pole[i]+ " ");
      System.out.println(" ");
    }
  }
}
```

Myšlenka:
1. vytvoříme prioritní frontu
2. vybíráme největší prvek a ukládáme od konce původního pole

```
void razeniPF() {
  PF pf = new PF();
  int i;
  // z pole vložíme prvky do prioritní fronty
  for(i = 0; i < pocet; i++)
    pf.vloz(pole[i]);
  // do pole je uložíme operací vybermax
  for(i = pocet-1; i >= 0; i--)
    pole[i] = pf.vybermax();
}
```

Zkuste si klienta* `RazeniPole` *s různými imlementacemi třídy* `PF` *!

- metoda `razeniPF()` obecně neřadí na místě

PF uspořádaným polem vede na metodu řazení typu řazení vkládáním

PF neuspořádaným polem vede na metodu, která odpovídá řazení výběrem

PF pomocí haldy

vytvoření $N \log_2 N$

vybíraní největšího prvku je

$$\log_2 N + ... + \log_2 2 + \log_2 1 < N \log_2 N$$

řazení je $N \log_2 N$

implementace

1. v poli vytvoříme haldu o dvou, třech, ..., **`pocet`** prvcích
2. největší prvek vyměníme s posledním a obnovíme haldu o jeden prvek menší

```
class RazeniPoleHaldou1 {
  private void vymen(int i, int j) {
    int t = pf[i];
    pf[i] = pf[j];
    pf[j] = t;
  }
  private void dolu(int k, int pocet) {
    while (2*k <= pocet) {
      int j = 2*k;
      if (j < pocet && pf[j] < pf[j+1]) j++;
      if (pf[k] >= pf[j]) break;
      vymen(k,j);
      k = j;
    }
  }
  private void nahoru(int k) {
    while (k > 1 && pf[k/2] < pf[k]) {
      vymen(k, k/2);
      k = k/2;
    }
  }
```

```
  private int[] pf;
  private int pocet;
  final int maxN=10;

  RazeniPoleHaldou1() {
    pf = new int[maxN + 1];
    pocet = 0;
  }

  void nactiPrvek(int klic) {
    pf[++pocet] = klic;
  }

  void tiskPole () {
    for(int i = 1; i <= pocet; i++)
      System.out.print(pf[i]+" ");
    System.out.println(" ");
  }

  void razeniHaldou1() {
    int i = 1;
    // postupně vytvoříme v poli pf haldu
    // velikosti i=1, ..., pocet
    while (i < pocet)
      nahoru(++i);
    // největší prvek vyměníme s posledním a
    // obnovíme haldu o 1 menší
    while (i > 1) {
      vymen(1,i);
      dolu(1,--i);
    }
  }
}
```