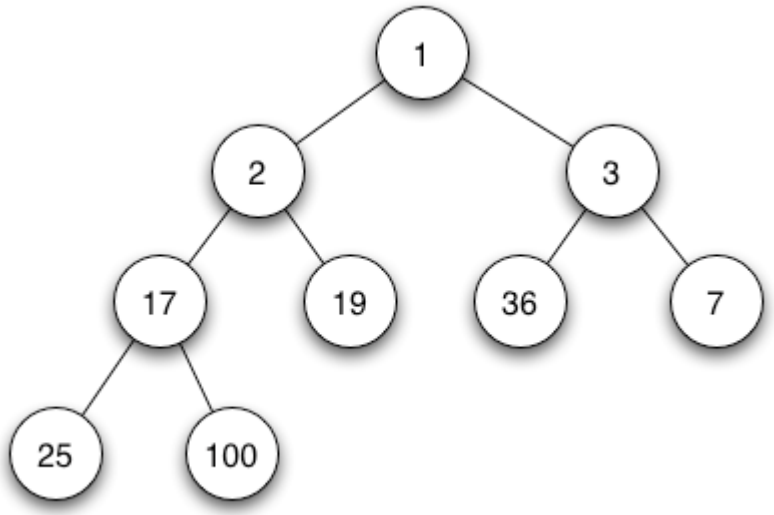
Algorytmy i złożoności

Wykład 4. Kopiec (ang. *heap*)

Sortowanie kopcowe

Definicja
Kopiec to drzewo binarne, w którym:

- dla dowolnych węzłów u, v, jeśli u jest następnikiem v to spełniony jest *warunek kopca*, czyli **u ≤ v**
  (w korzeniu jest element największy w kopcu)
- wszystkie poziomy drzewa, oprócz być może ostatniego, są wypełnione; ostatni jest wyrównany do lewej. Przykładowy kopiec:



Dodatkowe założenie: kopiec przechowywany jest w tablicy t[1..n], (indeksowanej od 1 !) kolejno poziomami, tj.:

- t[1] = korzeń,
- t[2], t[3] = poziom (głębokość) 1, itd.

Zależności pomiędzy numerami węzłów w kopcu::

- poprzednikiem `t[i]` jest `t[i/2]` (dzielenie całkowitoliczbowe),
- lewym następnikiem `t[i]` jest `t[2*i]`,
- prawym następnikiem `t[i]` jest `t[2*i+1]`.

Wysokość kopca o *n* wierzchołkach wynosi dokładnie
$h(n) = \lfloor \log n \rfloor$

Operacje przesiewania: w dół i w górę

Operacja 1 (Przesiewanie w górę)
Zakładając, że wartość `t[k]` zwiększyła się przywróć warunek kopca na ścieżce od `t[k]` do korzenia. W tym celu zamieniaj kolejne pary (poprzednik, następnik), aż trafisz na poprawny porządek w parze, albo dojdziesz do korzenia.

```
void upheap (Item t[], int k)
{
     while (k>1 && t[k/2] < t[k])
     {    exch (t[k], t[k/2]);
          k = k/2;
     }
}
```

Pesymistyczna liczba porównań: około *log k* (dokładnie: $\lfloor log\ k \rfloor$)

Operacja 2 (Przesiewanie w dół)
Zakładając, że wartość `t[k]` zmniejszyła się przywróć warunek kopca w poddrzewie o korzeniu `t[k]`.

```
void downheap (Item t[], int k, int n)
{
     int j = 2*k ;
     while (j <= n)
     {    if (j < n && t[j] < t[j+1]) j++;
          if (t[k] >= t[j]) break;
          exch (t[k], t[j]);
          k = j; j = 2*k;
     }
}
```

Pesymistycznie: na każdym poziomie dwa porównania: wybór większego z następników oraz porównanie z poprzednikiem.

Zatem liczba porównań: około 2 * różnica poziomów węzła k i liści.
Dokładnie liczba porównań: 2*$\lfloor$log (n/k)$\rfloor$ ==> złożoność O(log n).

Operacja 3 (Realizacja kolejki priorytetowej)
Kopiec jest strukturą, która może być wykorzystana do realizacji kolejki priorytetowej.

Przypomnijmy: *kolejka priorytetowa* to struktura danych, dla której są określone następujące operacje:
- *insert (Q, e)* wstaw element e do kolejki Q
- *max (Q)* podaj wartość największego elementu w kolejce Q
- *deletemax (Q)* usuń największy element z kolejki Q

Kopiec jest w tablicy `t[]`, zmienna *n* zawiera liczbę elementów kopca.

```
void insert (Item e)
{
     t[++n] = e ;
     upheap (t, n) ;
}
```
Złożoność: O(log n)

```
Item max()
{
     return t[1] ;
}
```
Oczywiste: O(1)

```
void deletemax();
{
     t[1] = t[n];
     n--;
     downheap(a, 1, n);
}
```

Złożoność: O(log n)
Inny wariant: zamiast `t[1]=t[n]` wykonaj `exch(t[1], t[n]);`
Wtedy dotychczasowy największy jest na końcu tablicy, poza zakresem kopca (stosuje się to w algorytmie *heapsort*)

Operacja 4 (Sortowanie przez kopcowanie (*heapsort*))
Dysponując kolejką priorytetową można sortować:
1. kolejno dla i=1, ..., n wykonuj: insert (Q, t[i]) ;
2. kolejno dla i=n, n-1, ... , 1 wykonuj:
     t[i] = max (Q) ; deletemax (Q) ;
Elementy opuszczają kolejkę w kolejności malejącej.

Jeśli kolejka realizowana za pomocą kopca to otrzymujemy sortowanie o złożoności O(*n*log*n*)
Wada: kopiec to dodatkowa tablica, oprócz tablicy `t[]`.
Wady tej nie ma sortowanie bezpośrednio wykorzystujące operację *downheap* w tablicy t[ ] – *heapsort*.

```
void heapsort (Item t[], int n)
// tablica t[] jest n+1 – elementowa: t[n+1]

for (k = n/2 ; k >= 1 ; k--)
     downheap (a, k, n) ;
while (n > 1)
{
     exch (t[1], t[n]) ;
     n-- ;
     downheap (a, 1, n) ;
}
```