Kopce binarne i sortowanie kopcowe (Heap-sort)

Definicja *Kopiec binarny* jest to drzwo binarne, w którego węzłach znajdują się pewne wartości i które spełnia następujące warunki:

- wszystkie poziomy drzewa oprócz (być może) najniższego są całkowicie wypełnione, a najniższy jest wypełniony bez luk od lewej strony do prawej do pewnego miejsca
- wartość w każdym węźle jest większa lub równa od wartości w każdym z jego synów

Stwierdzenie Kopiec binarny zawierający n węzłów ma wysokość $\lfloor \lg n \rfloor$.

Tablicowa reprezentacja kopca

Wartości z węzłow kopca znajdują się w tablicy na kolejnych pozycjach zaczynając od początku tablicy; wartości te są wstawione poziomami drzewa zaczynając od góry kopca, każdy poziom od lewej do prawej. Wówczas, zakładając, że pozycje w tablicy indeksujemy od 1, mamy:

- lewy syn węzła z pozycji i znajduje się na pozycji $2i$
- prawy syn węzła z pozycji i znajduje się na pozycji $2i + 1$
- ojciec węzła z pozycji i znajduje się na pozycji $\lfloor i/2 \rfloor$

Oznaczenia:

- `A.length` to rozmiar tablicy `A`
- `A.heapSize` to rozmiar kopca `A`
  (kopiec niekoniecznie wypełnia całą tablicę)

Uwaga: tablica indeksowana od 1

```
Heapify(A,i)

  l = 2*i  // lewy syn
  r = 2*i+1  // prawy syn
  if l<= A.heapSize and A[l]>A[i]
      largest = l
  else
      largest = i
  if r<= A.heapSize and A[r]>A[largest]
      largest = r
  if largest != i
      zamien A[i] z A[largest]
      Heapify(A,largest)


BuildHeap(A)
  A.heapSize = A.length
  for i = floor(A.length/2) downto 1
     Heapify(A,i)

HeapSort(A)
  BuildHeap(A)
  for i = A.length downto 2
     zamień A[A.heapSize] z A[1]
     A.heapSize = A.heapSize-1
     Heapify(A,1)
```

Stwierdzenie Pesymistyczny czas sortowania kopcowego jest $\Theta(n \, lg \, n)$, gdzie n to rozmiar sortowanej tablicy.

Kolejki priorytetowe

Definicja *Kolejka priorytetowa* jest to struktura danych zawierająca pewne elementy posiadające priorytety (priorytety to jakieś wartości, które można porównywać, typowo są to liczby całkowite) i na której są określone operacje:

- `Insert` - wstawia elment do kolejki priorytetowej
- `Maximum` - zwraca jako wynik element o maksymalnym priorytecie
- `Extract-Max` - usuwa z kolejki element o maksymalnym priorytecie, zwraca jako wynik ten usunięty element

kopcowa implementacja kolejek priorytetowych

```
Maximum(A)
   return A[1]


Extract-Max(A)
   x = A[1]
   A[1] = A[A.heapSize]
   A.heapSize = A.heapSize-1
   Heapify(A,1)
   return x


Insert(A,x)
// wstawia x do kopca A
   if A.heapSize == A.length
       błąd, kopiec pełny
   A.heapSize = A.heapSize+1
   i = A.heapSize  // i to wolna pozycja, na którą
                   // próbujemy wstawić x
   while i>1 and A[Parent(i)] < x
      A[i] = A[Parent(i)]
      i = Parent(i)
   A[i] = x
```

Złożoność pesymistyczna operacji na kolejekach priorytetowych

Porównanie różnych implementacji

	`Insert`	`Maximum`	`Extract-Max`
kopiec binarny	$\Theta(\lg\, n)$	$\Theta(1)$	$\Theta(\lg\, n)$
tablica nieuporządkowana	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
tablica uporządkowana	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$