Lista 1, zadanie 2

Tablicę indeksujemy od 0
Parzyste indeksy - kopiec maxowy
Nieparzyste indeksy - kopiec maxowy

Synowie wierzcholka w **`K[i]`** pamietani są (o ile istnieją) w **`K[2(i- i%2 +1) + (i%2)]`** i **`K[2(i - i%2 + 1) + (i%2) + 2]`**
Rodzic wierzchołka w **`K[i]`** pamietany jest w...
jeśli jest to prawy syn (**`i % 4 == 0 lub 1`**), to **`K[(i div 2) - 2 + i%2]`**
wpp. **`K[(i div 2) - 1 + i%2]`**

sąsiad ("lustrzany", zdefiniowany tylko dla liści) z drugiego kopca wierzchołka w **`K[i]`** pamiętany jest w **`K[i +1 - 2*(i % 2)]`**, ale tylko w sytuacji idealnej
ogólnie: jeśli "lustrzany" ma syna, to ten syn jest sąsiadem właściwym (nie może mieć dwóch! Prędzej i dostanie własnego syna, niż jego "lustrzany" sąsiad drugiego)
wpp. jeśli "lustrzany" istnieje, to on jest sąsiadem właściwym
wpp. rodzic "lustrzanego" jest sąsiadem właściwym (on już musi istnieć, z właściwości kopca)

przykład (na wierzchołkach są indeksy z tablicy, nie zawartości)

```
   0
 2  4
6

 3  5
  1
```

więzi międzykopcowe:
4-5 (lustrzanie)
5-4 (lustrzanie)
6-3 (3 jest "ojcem" lustrzanego, nieistniejącego wierzchołka o indeksie 7)
3-6 (6 jest synem lustrzanego 2)

Pamiętamy **`maxIndex`** (największy znaczący (odpowiadający wierzchołkowi) indeks w tablicy), powiedzmy że jako globalną

UWAGA: left/right odnosi się do stron po obróceniu kopca minowego do góry nogami (lub odbiciu go względem osi pionowej), czyli tak, jak na zwykłym rysunku kopca minimaksowego (w skrócie - lewy syn to syn występujący wcześniej w tablicy, dla obu kopców)

Pseudokod (i załączony kod w c) nie zawiera obsługi sytuacji skrajnych (przekroczenie pojemności tablicy, zdejmowanie z 0-elementowego kopca)

Funkcje definiujące relacje (dla klarowności późniejszego kodu)

```
function leftSon (i)
 return (2(i - i%2 + 1) + (i%2))

function rightSon (i)
  return (2(i - i%2 + 1) + (i%2) + 2)

function parent (i)
  if i==0 OR i==1 return i
  if i % 4 == 0 OR i % 4 == 1 then return (parent(i-2)
  return (i div 2) - 1 + i%2

function neighbour (i) //nigdy nie zostanie wywołane dla nie-liścia!
  mirror:= i + 1 - 2 * (i%2);
  if leftSon(mirror) <= maxIndex then return leftSon(mirror)
  if mirror <= maxIndex then return mirror
  return parent(mirror)
```

Funkcje i procedury faktyczne:

leftSon(j)>maxIndex implikuje rightSon(j)>maxIndex!

```
procedure przesuń-niżej (K[],i)
  k := i
  repeat
      j := k
      if j%2==0 AND leftSon(j) <= maxIndex AND K[leftSon(j)] > K[k]
            then k := leftSon(j
      if j%2==0 AND rightSon(j) <= maxIndex AND K[rightSon(j)] > K[k]
            then k := rightSon(j)
      if j%2==1 AND j>1 AND K[parent(j)] > K[k]
            then k := parent(j)
      if j%2==0 AND leftSon(j)>maxIndex AND K[neighbour(j)] > K[k]
            then k:=neighbour(j)
      K[j] <-> K[k]
  until j == k

procedure przesuń-wyżej (K[],i)
  k := i
  repeat
      j := k
      if (j%2==1) AND leftSon(j) <= maxIndex AND K[leftSon(j)] < K[k]
            then k:=leftSon(j)
      if (j%2==1) AND rightSon(j) <= maxIndex AND K[rightSon(j)] < K[k]
            then k:=rightSon(j)
      if (j%2==0) AND j>1 AND K[parent(j)] < K[k]
            then k:= parent(j)
      if (j%2==1) AND leftSon(j)>maxIndex AND K[neighbour(j)] < K[k]
            then k:=neighbour(j)
      K[j] <-> K[k]
  until j == k
```

W przeciwieństwie do kopca zwykłego nie możemy pominąć liści, bo może jest nieporządek na krawędziach międzykopcowych.

```
procedure buduj-kopiec (K[])
  for i:= maxIndex downTo 0
      if i%2 ==0 then przesuń-niżej(K,i)
      else przesuń-wyżej (K,i)
```

```
procedure insert (K[], what)
  maxIndex++
  K[maxIndex] := what
  if (what>parent(maxIndex) then przesuń-wyżej(K, maxIndex)
  else przesuń-niżej (K, maxIndex)

function deletemin(K)
  if maxIndex == 0 then //min jest maksem dla tablicy 1-elementowej
      maxIndex--
      return K[0]
  res := K[1]
  K[1] = K[maxIndex]
  maxIndex--
  przesuń-wyżej(K,1)
  return res

function deletemax(K)
  res := K[0]
  K[0] := K[maxIndex]
  maxIndex--
  if maxIndex >=0 then przesuń-niżej(K,0)
  return res
```