Solutions to Heapsort with pointers

Ferenc Pittler

15. April 2017

We implement the heap sort algorithm using a balanced binary tree. Every heap node has three pointers:

1. `parent`: points to the node's parent
2. `left`: points to it's right child
3. `right`: points to its left child

The definition of the heap is actually quite simple (the realization is not as simple as it sounds): it is a balanced binary tree with the a relationship between a child and it's parent. For example if the nodes contains some numerical value, the relationship can be, that the value from the parents is always larger as the value from its child.

Actually this implementation of the binary tree is quite similar to that of the double linked list. Therefore here we also use a specific typ, that points to the first and the last element of our binary tree. To summarize the types we are using are the followings in `c`:

```
typedef struct heapnode {
   int element;
   struct heapnode *left;
   struct heapnode *right;
   struct heapnode *parent;
} heapnode;
typedef struct binaryheap {
   struct heapnode *last;
   struct heapnode *first;
} binaryheap;
```

First we have to create an empty tree. For this we have to allocate memoryspace for a binaryheap. Then set its first and last pointers to `NULL`. This will be actually an empty binary tree.

```
binaryheap *createleertree(void){
   //erzeugt ein neues binaere Baum
   binaryheap *ret=(binaryheap *)malloc(sizeof(binaryheap));
   if (ret == NULL){
     printf("Memory reservierung fuer leer binary heap war nicht erfolgreich\n");
     exit(1);
   }
   //Unsere leere Baum
   ret->last=NULL;
```

```
    ret->first=NULL;
    return ret;
}
```

Having created a binary tree, we can start to fill it with nodes. The way we fill it actually is using a two dimensional array of heapnodes as helpers. We have now seen explicitely how a heap can be implemented by arrays, so we put the elements in arrays and set the connection between them (the parent, left, right) pointers with the help of the array index. For example the left of the node with arrayindex `i` has to be the node with arrayindex `2i+1`. We have to be careful, that for even number of elements the last parent does not have a right child!

```
void createheapnode(binaryheap **tree, heapnode **root, int *list, int n){
    int i;
//Zeiger array fuer die heap elements
    heapnode **array=(heapnode **)malloc(sizeof(heapnode*)*n);
    if (array == NULL){
      printf("Memory reservierung fuer Zeigers auf heap elements war nicht
     erfolgreich\n");
      exit(1);
    }
    for (i=0; i<n; ++i){
      array[i]=(heapnode *)malloc(sizeof(heapnode));
      if (array[i] == NULL){
        printf("Nicht genug speicher fuer die %d element im heap\n", i);
        exit(1);
      }
    }
//erfuellt alle elementen mit den Werten
    for (i=0; i<n; ++i){
      array[i]->element=list[i];
      array[i]->left=NULL;
      array[i]->right=NULL;
      array[i]->parent=NULL;
    }
//machen die eltern kind beziehungen
//linkes Kind fuer i wird 2*i+1 sein
//rechtes kind fuer i wird 2*i+2 sein
//achten wenn n ist gerade wir haben am
//ende ein linkes Kind
    for (i=0; i<n/2;++i){
      array[i]->left=array[2*i+1];
      array[2*i+1]->parent=array[i];
      if (2*i+2 != n){
        array[i]->right=array[2*i+2];
        array[2*i+2]->parent=array[i];
      }
    }
    *root=array[0];
    (*tree)->first=array[0];
    (*tree)->last=array[n-1];
    free(array);
}
```

In this way we only have filled it with random elements, so our we are not able to call our binary tree as a binary heap. We must restore the heap property.

```c
void versickern(binaryheap **b, heapnode **q){
   int temp1;
   int temp2;
   int orig;
//wenn sie kein Kind hat wir sind fertig
   if ((*q)->left == NULL){
      return;
   }
//wenn sie ein Kind hat es muss link sein
//uberprufen wir die heapnode eigenschaft
//wenn es notig, tauschen wir die Wert
//mit dem linken Kind
   if ((*q)->right == NULL){
      if ( (*q)->element < (*q)->left->element ){
         swap(b,q,*q, (*q)->left);
      }
      return;
   }
//wenn es zwei Kinder hat
//wir mussen auch das heapnode eigenschaft uberpruefen
   temp1=(*q)->left->element;
   temp2=(*q)->right->element;
   orig=(*q)->element;
//wenn das root ist die groesste wir sind fertig
   if ((orig > temp1 ) && (orig > temp2))
     return;
//wenn nicht wir suchen fuer die groesste
//von rechten und linken Kind und tauschen
//aber wir sind nicht fertig, wir muessen
//auch fuer die rechtes beziehungseise
//der linkes Kind auch uberpruefen
   if ( temp1 > temp2) {
     swap(b,q,*q, (*q)->left);
     versickern(b,&((*q)->left));
     return;
   }
   swap(b,q,*q, (*q)->right);
   versickern(b,&((*q)->right));
   return;
}
const int MAX=100;
void heapify(binaryheap **b,heapnode **q){
   if ((*q)->left == NULL)
     return;
   else
     heapify(b,&((*q)->left));
   if ((*q)->right != NULL){
     heapify(b,&((*q)->right));
   }
   versickern(b,q);
}

void heapify(binaryheap **b,heapnode **q){
   if ((*q)->left == NULL)
```

```
      return;
    else
      heapify(b,&((*q)->left));
    if ((*q)->right != NULL){
      heapify(b,&((*q)->right));
    }
    versickern(b,q);
}
```

Here the most important routine is the swap, which actually swap a parent with its child in the tree. We have to set the following pointers (Figure FIXME).

```
void swap(binaryheap **b,heapnode **root, heapnode * const parent, heapnode * const child) {
  heapnode *p = parent;
  heapnode *c = child;
  if (child == parent->left) {
     c->parent=p->parent;
     p->parent=c;
     p->left=c->left;
     c->left=p;
     if (p->right!=NULL){
      p->right->parent=c;
     }
     if (c->right!=NULL){
      c->right->parent=p;
     }
     swap_elem(&(p->right),&(c->right));
  }
  else if (child == parent->right) {
     c->parent=p->parent;
     p->parent=c;
     p->right=c->right;
     c->right=p;
     if (p->left!=NULL){
      p->left->parent=c;
     }
     if (c->left!=NULL){
      c->left->parent=p;
     }
     swap_elem(&(p->left),&(c->left));
  }
  else {
    printf("Cannot swap elements which are not direct reletives\n");
    abort();
  }
  if (parent == *root) {
    *root = child;
  }
  if ((*b)->first == parent){
    (*b)->first=child;
  }
  if ((*b)->last == child ){
    (*b)->last = parent;
  }
}
```

1 Remove the first from the tree

Here we free the first pointer and change the pointers in such a way that the last element will be a new root of the tree. Of course, this will ruin the heap property, which we have to restore with `versickern`. In this part we need a code to find actually the new last element of the tree. In order to get the new last element we implement the following trick:

1. If the original last element is a right child then it is easy. The left child of its parent will be the new last element.

2. If it is a left child we make the following trick. We go up and store the actual path. In order to go to the new last element we follow exactly the opposite path in the opposite order from the root till the left child of the actual node is not equal to zero.

```
void removelast(binaryheap **b,heapnode **q){
   heapnode *lastelem=(*b)->last;
   heapnode *firstelem=(*b)->first;
   if ( (*b)->first == NULL ){
     printf("Fehler es gibt kein element im Baum\n");
     exit(1);
   }
   if (firstelem == lastelem ){
    //nur ein element
    printf("%d\n", lastelem->element);
    //wir geben es frei
    free(lastelem);
    (*q)=NULL;
    (*b)->last = NULL;
    (*b)->first= NULL;
    return;
   }
   if (firstelem->left == lastelem){
    //gubt es nur zwei element
    printf("%d\n", firstelem->element);
    free(firstelem);
    //last elem parent zeiger to NULL
    lastelem->parent=NULL;
    (*b)->first=lastelem;
    (*q)=lastelem;
    return;
   }
   if (firstelem->right == lastelem){
    //nur drei element
    //nachdem tauschen wir ueberpruefen
    //das heap eigenschaft
    printf("%d\n", firstelem->element);
    lastelem->left=firstelem->left;
    free(firstelem);
    lastelem->parent=NULL;
```

```c
    (*b)->first=lastelem;
    (*b)->last=lastelem->left;
    lastelem->left->parent=lastelem;
    (*q)=lastelem;
    versickern(b,q);
    return;
   }
   if (lastelem->parent->left == lastelem){
      //spaicher das Weg zum root
      int *path=(int *)malloc(sizeof(int)*100);
      heapnode *temp2=lastelem->parent;
      printf("%d\n",firstelem->element);
      heapnode *temp=lastelem;
      heapnode *temp3;
      int i=0,j=0;
      if (path == NULL){
        printf("Memory allocation error in path\n");
        exit(1);
      }
      for (i=0; i<100; ++i)
        path[i]=0;
      i=0;
      while(temp->parent != NULL){
        temp3=temp->parent;
        if (temp3->right == temp){
          path[i]=1;
        }
        else{
          path[i]=0;
        }
        temp=temp3;
        i++;
      }
      //wir machen das gegenteil
      //vom Weg in der andere richtungs
      for(j=0; j<i;++j){
        if (path[i-j-1]==0){
          temp3=temp->right;
          if (temp3 == NULL)
            break;
          temp=temp3;
          }
         else{
          temp3=temp->left;
          if (temp3 == NULL)
           break;
          temp=temp3;
         }
      }
      lastelem->right=firstelem->right;
      lastelem->left=firstelem->left;
      firstelem->left->parent=lastelem;
      firstelem->right->parent=lastelem;
      free(firstelem);
      temp2->left=NULL;
      lastelem->parent=NULL;
      (*q)=lastelem;
```

```
        (*b)->first=lastelem;
        (*b)->last=temp;
        //wir muessen das heap eigenschaft ueberpruefen
        versickern(b,q);
        return;
    }
    else{
        printf("%d\n",firstelem->element);
        heapnode *temp2=lastelem->parent;
        heapnode *temp=lastelem->parent->left;
        lastelem->right=firstelem->right;
        lastelem->left=firstelem->left;
        firstelem->left->parent=lastelem;
        firstelem->right->parent=lastelem;
        free(firstelem);
        temp2->right=NULL;
        lastelem->parent=NULL;
        (*q)=lastelem;
        (*b)->first=lastelem;
        (*b)->last=temp;
        versickern(b,q);
    }

}
```

2 main

```
int main(int argc, char *argv[]){
    int list[]={2,3,4,8,5,6,1,7};
    heapnode *Q;
    binaryheap *B;
    B=createleertree();
    createheapnode(&B,&Q,list,8);

    heapify(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
    removelast(&B,&Q);
```