SOWMYA Q.V.
1RN17CS078

Date______ Page______

Binomial Heap & operations on it

```
int binomialLink (Node *h1, Node *h2)
{
    h1 -> parent = h2;
    h1 -> sibling = h2 -> child;
    h2 -> child = h1;
    h2 -> degree = h2 -> degree + 1;
}

Node * mergeBHeaps (Node *h1, Node * h2)
{   if (h1 == NULL)
        return h2;
    if (h2 == NULL)
        return h1;
    Node * res = NULL;
 if (h1 -> degree <= h2 -> degree)
    res = h1;
 else if (h1 -> degree > h2 -> degree)
    res = h2;
 while (h1 != NULL && h2 != NULL)
 {  h1
    if (h1 -> degree < h2 -> degree)
        h1 = h1 -> sibling;
  degree
   else if (h1 -> degree == h2 -> degree)
   {  Node *sib = h1 -> sibling;
        h1 -> sibling = h2;
      h1 = sib;
   }
   else { Node * sib = h2 -> sibling;
            h2 -> sibling = h1;
             h2 = Sib;
        }
 } return res;
}
Node * unionBHeaps(Node *h1, Node *h2)
{ if(h1 == NULL && h2 == NULL)
   return NULL;
  Node * res = mergeBHeaps(h1, h2);
 Node * prev = NULL, *curr = res,
        * next = curr->sibling;
 While(next != NULL)
 {
  if((curr->degree != next->degree) ||
    ((next->sibling != NULL) &&
    (next->sibling)->degree == curr->degree))
  {
    prev = curr;
    curr = next;
  }
 else {
    if(curr->val <= next->val)
    { curr->sibling = next->sibling;
      binomialLink(next, curr);
  }
 else{
     if(prev == NULL)
      res = next;
     else
     prev->sibling = next;
     binomialLink(curr, next);
     curr = next;
}}
 next = curr->sibling;
}
return res;
}
```

```
void binomialHeapInsert (int x)
{
  root = unionBHeaps (root, createNode (x));
}
void display (Node *h)
{
 while (h)
{
 cout << h->val << " ";
 display (h->child);
 h= h->sibling;
}
}
int reverseList (Node *h)
{   if (h-> sibling != NULL)
  {
   reverseList (h-> sibling );
   (h->sibling) -> sibling =h;
  }
else
  root=h;
}
Node *extractMin BHeap (Node *h)
{  if (h== NULL)
    return NULL;
   Node *min_node_prev = NULL;
   Node *min_node =h;
   int min = h->val;
   Node * curr =h;
  while (curr -> sibling)->val <min)
  { if ((curr -> sibling) -> val <min)
  { min = (curr -> sibling) -> val;
     min_node_prev = curr;
     min_node = curr -> sibling;
  }
```

Date________Page______

```
curr = curr ->sibling;
}
if (min_node - prev == NULL && min_node ->
sibling == NULL )
    h = NULL;
else if (min_node - prev == NULL)
    h= min_node ->sibling;
else
  min_node _prev ->sibling= min_node ->sibling ;
if (min_node ->child != NULL)
{ revertList (min_node -> child);
  (min_node -> child ) -> sibling = NULL;
}
return UnionBHeaps (h, root );
}

Node *findNode(Node *h, int val)
{  if(h==NULL)
     return NULL;
   if (h-> val ==val )
     return h;
  Node * res =findNode (h-> child, val );
   if (res != NULL )
     return res;
  return findNode (h-> sibling ,val );
}
void  decreaseKeyBHeap (Node* H ,int old_val,
                                 int new_val )
{
 Node * node = findNode  (H, old_val);
if( node == NULL)
  return;
```

Date_______ Page_______

```
node->val = new-val;
Node * parent = node -> parent;
while(parent != NULL && node->val < parent->
                                    val)
 {
  swap(node -> val, parent->val);
    node = parent;
    parent = parent -> parent;
}
 }
Node * binomialHeapDelete( Node *h, int val)
 {
  if (h==NULL)
   return NULL;
 decrease KeyBHeap(h, val, INT-MIN);
  return extractMinBHeap(h);
 }
```