ADS - LAB - 9

REVANTH . R
1BM18CS082
[signature]

9/12/2020

Implementation of Binomial Heap

```
// A Binomial Tree Node
struct Node
{ int data, deg;
  Node *child, *sis, *parent;
};

Node * newNode (int key)
{
    Node * temp = new Node;
    temp -> data = key;
    temp -> degree = 0;
    temp -> child = temp -> parent = temp -> sis = NULL;
    return temp;
}

// Inserting a key into the binomial Heap
list <Node *> insert (list <Node *> _head, int key)
{   Node * temp = newNode (key);
    return insertTreeHeap (_head, temp); }
```

// GetMin function return pointer of minimum value Node present in the binomial Heap

```
Node * getMin ( list <Node *> _heap )
{ list <Node*> :: iterator it = _heap.begin();
  Node *temp = *it;
  while ( it != _heap.end() )
  { if ((*it) -> data < temp -> data)
      temp = *it ; it ++ ;
  } return temp; }

list <Node *> extractMin ( list <Node *> _heap )
{ list <Node *> new_heap, lo;
  Node * temp;   // temp contains pointer
  temp = getMin (_heap);   of min. value
  list <Node *> :: iterator it;
  it = _heap.begin ();
  while ( it != _heap.end() )
{
```

```
if (*it != temp)
{
    new_heap.push_back (*it);
}
    it++; }

lo = removeMinTreeBHeap (temp);

new_heap = unionBinomialHeap (new_heap, lo);

new_heap = adjust (new_heap);

return new_heap;
}

// Inserting a Binomial Tree into Binomial Heap

list <Node*> insertTreeInHeap (list<Node*> _heap,
                               Node *tree)
{   list <Node*> temp;

    temp.push_back (tree);

    temp = unionBinomialHeap (_heap, temp);

    return adjust (temp);
}

int main ()
{
    return 0;
}
```