Lab-09

Rudramuni G.M
1BM18CS085

Bafna Gold
Date: Page:

Binomial Heap

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

Node* meargeBinomialTrees(Node *b1, Node *b2)
{ if (b1->data > b2->data)
    swap(b1, b2);
  b2->Parent = b1;
  b2->sibling = b1->child;
  b1->child = b2;
  b1->degree++;
  return b1;
}

list<Node *> unionBinomialHeap(list<Node *>l1,
                               list<Node *>l2)
{
list<Node*> _new;
list<Node *>::iterator pt = l1.begin();
list<Node *>::iterator ot = l2.begin();
while (it != l1.end() && ot != l2.end())
{ if((*pt)->degree <= (*ot)->degree)
  { _new.push_back(*it);
  } it++;
else
{ _new.push_back(*ot);
  ot++;
return _new;
```

07 Rudra

Rudramuni G.M
1BM18CS085

Lab08

```
list<Node *> insert ATree Inheap list<node *>
{  list<Node *> temp;
   temp.push_back(tree);
temp = unionBinomialHeap(_heap, temp);
 return adjust(temp);
}
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 = getmin(_heap);
    list<Node*>::iterator it;
    it = heap.begin();
 while(it != heap.end())
 {
     if (*it != temp)
    {
       new_heap.push_back(*it);
     }
   } it++;
 lo = removemin FormTree Reverse Bheap(temp);
New_heap = union BinomialHeap(new_heap, lo);
 new_heap = adjust(new_heap);
   return new_heap;
}
```

Rudra