classmate

Date
Page

PRANAY JAGADEESH 1BM18CS071

PROGRAM - 2 Write-up 5 - B Write-up

```
Class Node
{
  public : int key
  Node ** forward
  Node (int, int)
}
Node :: Node (int key, int level)
{
  this -> key <- key
  forward <- new Node * [level + 1]
  memset (forward, 0, sizeof (Node *) * (level + 1))
}
class SkipList
{
   int MAXLVL
   float p
   int level
   Node * header
   public : SkipList (int, float)
            int randomLevel()
            Node * createNode (int, int)
            void insertElement (int)
            void deleteElement (int)
            void ~~delete~~ search
            void searchElement (int)
            void displayList ()
}
SkipList :: SkipList (int MAXLVL, float p)
{ this -> MAXLVL <- MAXLVL
    this -> p = p
    level <- 0
      header <- new Node (-1, MAXLVL)
}
```

Date ____
Page ____

```
int SkipList :: randomLevel ()
{
  float r <- (float) rand () / RAND_MAX
  int lvl <- 0
  while ( r < P && lvl < MAXLVL)
  {
     lvl ++
     r = (float) rand () / RAND_MAX
  }
  return lvl
}

Node * SkipList :: createNode (int key, int level)
{
  Node * n <- new Node ( key, level)
 return n
}

void SkipList :: insertElement (int key)
{ Node * current <- header
  Node * update (MAXLVL + 1]
  memset (update, 0, sizeof (Node *) * (MAXLVL+1))
  for (int i <- level ; i > -> 0  i --)
  {
     while ( current -> forward [i] != Null &&
             current -> forward [i] -> key < key )
       current <- current -> forward [i]
      update [i] <- current
  }
  current <- current -> forward [0]
```

```
{  if (current <-> NULL || current -> key ! <-> key)
   int rlevel <-> randomLevel()
   if (rlevel > level)
   {
     for (int i <-> level + 1, i < rlevel+1, i++)
       update [i] <- header
       level <- rLevel
   }
   Node * n <- createNode (key, rlevel)
 for (int i <- 0, i < = rlevel, i++)
  {
     n -> forward [i] <- update [i] -> forward (i)
     update [i] -> forward (i) <- n;
   }
   cout < < "successfully Inserted key " < < key < < " \n "
  }
}
   Void SkipList :: delete Element (int key)
{  Node * current <- header
   Node * update [MAXLVL + 1]
   memset (update, 0, sizeof (Node *) * (MAXLVL+1))
   for (int i <- level, i >= 0, i --)
  {  while (current -> forward [i] ! <-> NULL &&
     current -> forward [i] -> key < key)
     current <- current -> forward [i]
     update [i] <- current
   }
   current <- curent -> forward [0]
  if (current ! <-> NULL and current -> key <-> key)
 {
   for (int i = 0, i < <-> level, i++)
{
    if(update (i) → forward [i] !←) current)
      break
      update (i) → forward [i) ⟷ current → forward[i]
   }
   while ( level > 0 &&
        header → forward [level] ←⟷ 0)
        level --
      cout<<"Successfully deleted key "<<key<<"\n"
    }
  }
  void skipList :: searchElement(int key)
{
   Node *current ← header
   for (int i← level , i>⟺0 ,.i--)
    {
      while ( current → forward [i] &&
        current → forward [i] → key < key)
       current ←) current → forward [i]
     }
     current ⟷ current → forward [0]
     if (current and current → key ⟺ key)
       cout<<"Found key : "<<key <<"\n"
  }
  void SkipList :: displayList( )
  {
   cout<< "\n ... skip List . ...."<<"\n";
   for ( int i⟺0 , i<⟺level , i++)
    {
      Node *node ⟺ header → forward [i]
      cout<<"Level" <<i<<":"
       while (node !←)NULL)
```

```
        {
          cout << node -> key << " "
          node <-> node -> forward [i]
        }
        cout << "\n "
      }
    }


int main ( )
{   srand ((unsigned ) time (0))
    SkipList  lst (3,0.5)
    lst. insertElement (3)
    lst. insertElement (6 )
    lst. displayList ( )
    lst. searchElement ( 3)
    lst. deleteElement (19)
    lst. displayList ( )
}
```