**Section (I): Tracing Questions (Total: 8 marks)**

**(Q1) (3 mark)**

* Draw the binary expression tree for the following prefix expression.
* Also, write the corresponding postfix, and infix expression.
* **+ - A \* - C M - / / / E Z F G H B**

**(Q2) (5 mark)**

**For exercises (i - iii) use the following values, and draw the hash table for each of them:**

**30 , 33 , 47 , 24 , 37 , 21 , 40**

1. Store the values in a hash table with size 8 cells.
2. Store the values in a hash table with 4 buckets, each bucket contains 2 cells.
3. Store the values in a hash chain table with function (Key % 8).
4. Fill in the following table, showing the number of comparisons needed to find each value:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **(i)** | **(ii)** | **(iii)** |
| **21** |  |  |  |
| **47** |  |  |  |
| **40** |  |  |  |

**Section (II): Algorithm Questions (Total: 8 marks)**

**Algorithm 1: (8 marks)**

You are asked to write a function called **Remove\_Most\_Two\_ Largest\_Nodes** **( )**

* Your function will remove the most 2 largest nodes. **!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT**

|  |  |
| --- | --- |
| 865  860  399  395  450  400  70  90  80  50  112  120  **115**  160  **600**  110  **500**  122  155  130  220  270  260  250  200  390 | **940**  850  73  77  75  70  155  170  160  805  **950**  150  810  200  800  700  835  830  840 |

void Remove\_Most\_Two\_Largest\_Nodes()

{

CTNode\* pT = pRoot,\*pB=NULL,\*ptt=NULL,\*pBt=NULL;

int check = 0;

while (check < 2)

{

pT = pRoot, pB = NULL, ptt = NULL, pBt = NULL;

while (pT->pRight != NULL)

{

pB = pT;

pT = pT->pRight;

}

if (pT->pLeft == NULL && pT->pRight == NULL)

{

if (pB == NULL)

{

pRoot = NULL;

}

else

{

if (pB->pLeft == pT) //not related to Q requirement but its for remove process generally

{

pB->pLeft = NULL;

}

else

{

pB->pRight = NULL;

}

}

delete pT;

}

else

{

if (pT->pLeft != NULL && pT->pRight != NULL)

{

ptt = pT->pLeft;

while (ptt->pRight != NULL)

{

pBt = ptt;

ptt=ptt->pRight;

}

pT->info = ptt->info;

if (pBt != pT)

{

pBt->pRight = ptt->pLeft;

}

else

{

pBt->pLeft = ptt->pLeft;

}

}

else

{

if (pB == NULL) //the first node

{

if (pT->pLeft != NULL)

{

pRoot = pT->pLeft;

}

else

{

pRoot = pT->pRight;

}

}

if (pB->pLeft != NULL)

{

if (pT->pLeft != NULL)

{

pB->pLeft = pT->pLeft;

}

else

{

pB->pLeft = pT->pRight;

}

}

else

{

if (pT->pLeft != NULL)

{

pB->pRight = pT->pLeft;

}

else

{

pB->pRight = pT->pRight;

}

}

}

}

check++;

}

}

**Section (III): Problem Solving (Total: 24 marks)**

**Problem 1: (5 marks)**

*In the this question use the following definition for the nodes in the List:*

**class CListNode**

**{**

**public:**

info

pNext

**int info;**

**CListNode \*pNext;**

**};**

**Note:**

* in this problem you have to write just single function

(no permission to write more than 1 function).

* no permission to use loop inside the function.

Write a **recursive** function to do the following:

The function will creates a third list **(L3)**, that contains the length of gaps in **(L1)**, based on the values from **(L2)**.

**Assume** that, the values in **(L2)** , will be exist in **(L1)** and in the same order of **(L2)**.

**e.g.**

Len = 2

Len = 0

Len= 3

L3

Head

L1

Head

L2

Head

void GapLength(CNode\* pT1, CNode\*& pB, CNode\*& pT2, int& check, int& len, CList& L)

{

if (pT1 == NULL || pT2 == NULL)

return;

if (pT1->info == pB->info || pT1->info == pT2->info)

{

if (pT1->info == pB->info) //to start count

{

check = 1;

}

else

{ //to end count

if (len > 0)

{

len--;

}

CNode\* pnn = new CNode;

pnn->info = len;

check = 0;

len=0;

pB = pB->pNext;

pt2 = pT2->pNext;

L.Attach(pnn);

}

}

if (check == 1)

{

len++;

}

GapLength(pT1->pNext, pB, pT2, check, len, L);

}

**Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a binary search tree from the user. **(T)**
2. **For the Left sub-tree:**

* Calculate the total of the nodes of the (1st half) of the height. **(TotLeftUp)**.
* Calculate the total of the nodes of the (2nd half) of the height. **(TotLeftDown)**

1. **For the Right sub-tree:**

* Calculate the total of the nodes of the (1st half) of the height. **(TotRightUp)**.
* Calculate the total of the nodes of the (2nd half) of the height.**(TotRightDown)**

**Assume that** : the height of the left sub-tree will be even , and

the height of the right sub-tree will be even

2450

84

80

83

86

210

90

200

10

900

910

800

980

1050

1000

2400

2000

950

1060

890

1100

2500

500

Root

89

88

Height = 3

**TotLeftDown** = 510

Height = 3

**TotLeftUp** = 510

Height = 4

**TotRightUp** = 11890

Height = 4

**TotRightDown** = 11890

4790

1. Create a List according to the following protocol:

* Each node in the list points to the nodes in the 1st level of each sub-tree.

**Note:** declare your CListNode.

88

84

80

83

86

10

2500

1060

980

1050

1000

2400

2000

1100

950

900

890

210

90

200

910

800

500

Root

4790

2450

89

Head

1. Using your generated List:

* Display each right branch.

**e.g.**

200, 210

86, 88, 89

800 , 910 , 200, 4790

950 , 1000, 1050, 1060

2000 , 2400 , 2450

class CNode

{

public:

CNode\* pNext;

CTNode\* pUpT;

};

void CountLevels(CTNode\* pT, int level, int&maxHeight)

{

if (pT == NULL)

{

return;

}

CountLevels(pT->pLeft, level + 1, maxHeight);

if (level > maxHeight)

{

maxHeight = level;

}

CountLevels(pT->pRight, level + 1, maxHeight);

}

void CalcTotal(CTNode\* pT, int level, int start, int end, int&tot)

{

if (pT == NULL)

{

return;

}

CalcTotal(pT->pLeft, level + 1, start, end, tot);

if (level >= start && level <= end)

{

tot += pT->info;

}

CalcTotal(pT->pRight, level + 1, start, end, tot);

}

void FirstNode(CTNode\* pT, int level, int first, CList&L)

{

if (pT == NULL)

return;

FirstNode(pT->pLeft, level + 1, first, L);

if (level == first)

{

CNode\* pnn = new CNode;

pnn->pUpT = pT;

pnn->pNext = NULL;

L.Attach(pnn);

}

FirstNode(pT->pRight, level + 1, first, L);

}

void main()

{

CBST T;

CList L;

int N;

cin >> N;

for (int i = 0; i < N; i++)

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

pnT->pRight = pnT->pLeft = NULL;

T.Insert(pnT);

}

int level = 1, maxHeightL = -9999, tot = 0;

CountLevels(T.pRoot->pLeft, level, maxHeightL);

CalcTotal(T.pRoot->pLeft, level, 1, maxHeightL / 2, tot);

int ToLeftUp = tot;

tot = 0;

CalcTotal(T.pRoot->pLeft, level, (maxHeightL / 2) + 1, maxHeightL, tot);

int ToLeftDown = tot;

int maxHeightR = -9999;

CountLevels(T.pRoot->pRight, level, maxHeightR);

CalcTotal(T.pRoot->pRight, level, 1, maxHeightR / 2, tot);

int ToRightUp = tot;

tot = 0;

CalcTotal(T.pRoot->pRight, level, (maxHeightR / 2) + 1, maxHeightR, tot);

int ToRightDown = tot;

FirstNode(T.pRoot->pLeft, level, 1, L);

FirstNode(T.pRoot->pLeft, level, (maxHeightL / 2) + 1, L);

FirstNode(T.pRoot->pRight, level, 1, L);

FirstNode(T.pRoot->pLeft, level, (maxHeightR / 2) + 1, L);

CNode\* pTrav = L.pHead;

while (pTrav != NULL)

{

CNode\* pT = pTrav->pUpT;

while (pT != NULL)

{

cout << pT->info;

pT = pT->pDownL;

}

pTrav = pTrav->pNext;

}

}