PROGRAM 12: ACTIVITY SELECTION AND HUFFMAN CODING IMPLEMENTATION

Aim: To write a program in C++ for implementing Activity Selection and Huffman Coding.

Description:

Problem statement:

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Example 1 : Consider the following 3 activities sorted by by finish time.

```
start[]  = {10, 12, 20};
finish[] =  {20, 25, 30};
```

A person can perform at most two activities. The maximum set of activities that can be executed is {0, 2} [These are indexes in start[] and finish[]]

Example 2 : Consider the following 6 activities sorted by by finish time.

```
start[]  =  {1, 3, 0, 5, 8, 5};
finish[] =  {2, 4, 6, 7, 9, 9};
```

A person can perform at most **four** activities. The maximum set of activities that can be executed is {0, 1, 3, 4} [These are indexes in start[] and finish[]]

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of the previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

Algorithm:

Step 1: Sort the activities according to their finishing time

Step 2: Select the first activity from the sorted array and print it.

Step 3: Do the following for the remaining activities in the sorted array.

…….a) If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it.

Program:

```
// C++ program for activity selection problem.
// The following implementation assumes that the activities
// are already sorted according to their finish time
#include <bits/stdc++.h>
using namespace std;

// Prints a maximum set of activities that can be done by a single
// person, one at a time.
//  n   -->  Total number of activities
//  s[] -->  An array that contains start time of all activities
//  f[] -->  An array that contains finish time of all activities
```

```
void printMaxActivities(int s[], int f[], int n)
{
    int i, j;
    cout <<"Following activities are selected "<< endl;
     // The first activity always gets selected
    i = 0;
    cout <<" "<< i;
     // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
      // If this activity has start time greater than or
      // equal to the finish time of previously selected
      // activity, then select it
      if (s[j] >= f[i])
      {
          cout <<" " << j;
          i = j;
      }
    }
}

// driver program to test above function
int main()
{
    int s[] =  {1, 3, 0, 5, 8, 5};
    int f[] =  {2, 4, 6, 7, 9, 9};
    int n = sizeof(s)/sizeof(s[0]);
    printMaxActivities(s, f, n);
    return 0;
}
```

Output:

Following activities are selected n 0 1 3 4

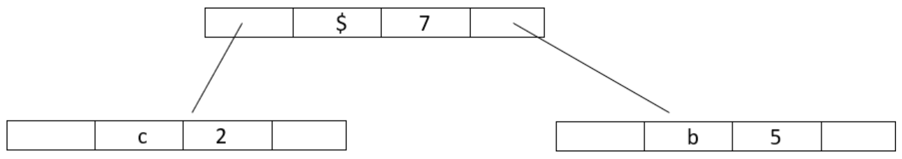
Description:

Character :: Frequency

a	::	10
b	::	5
c	::	2
d	::	14
e	::	15

Step 1: Build a min heap containing 5 nodes.

Step 2 : Extract two minimum frequency nodes from min heap. Add a new internal node 1 with frequency equal to 5+2 = 7

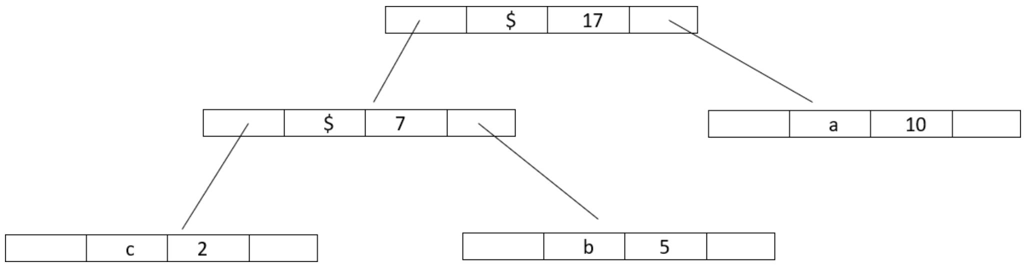


Now minheap contains 4 nodes:

Character	::	Frequency
a	::	10
Internalnode1	::	7
d	::	14

e	::	15

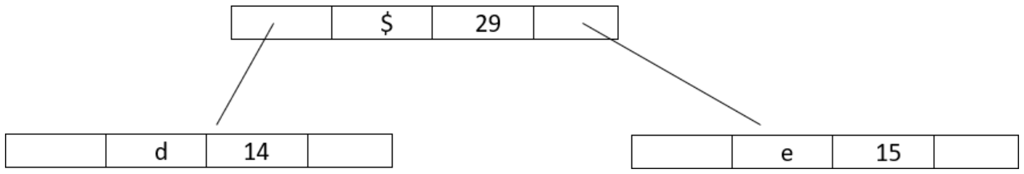
Step 3 : Again, Extract two minimum frequency nodes from min heap and add a new internal node 2 with frequency equal to 7+10 = 17



Now minheap contains 3 nodes:

Character	::	Frequency
Internalnode2	::	17
d	::	14
e	::	15

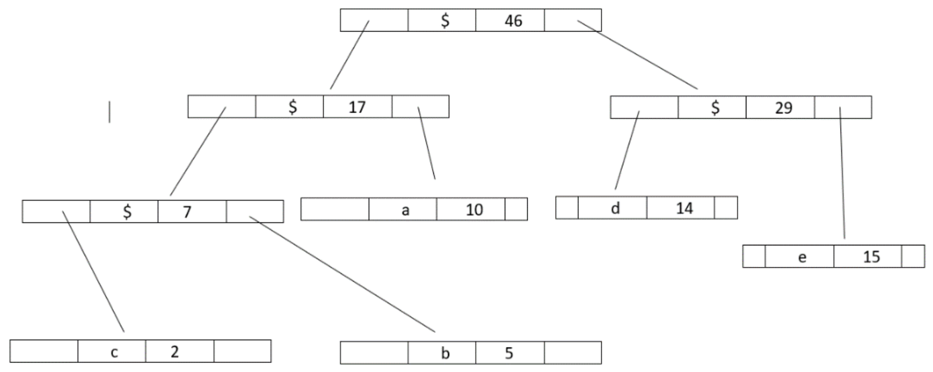
Step 4: Extract two minimum frequency nodes from min heap and add a new internal node 3 with frequency equal to 15+14 = 29.



Now minheap contains 2 nodes:

Character	::	Frequency
Internalnode2	::	17
Internalnode3	::	29

Step 5 : Extract two minimum frequency nodes from min heap and add a new internal node 4 with frequency equal to 17+29 = 46.



Now minheap contains 1 node:

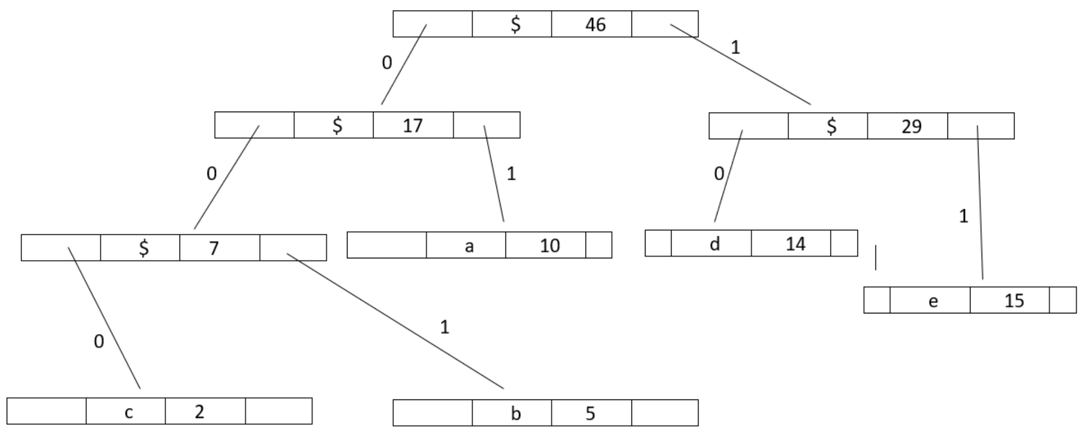
Character	::	Frequency
Internalnode4	::	46

Since the heap contains only one node so, the algorithm stops here. Thus, the result is a Huffman Tree.

Print codes from Huffman Tree:

The steps to Print codes from Huffman Tree

1. Traverse the tree formed starting from the root.
2. Maintain a string.
3. While moving to the left child write '0' to the string.
4. While moving to the right child write '1' to the string.
5. Print the string when the leaf node is encountered.



Algorithm:

1. Create a leaf node for every character in the input. Build a Minimum Heap of all leaf nodes.
2. For the Minimum Heap, get the top two nodes (say N1 and N2) with minimum frequency.
3. Create a new internal node N3 with frequency equal to the sum of frequency of nodes N1 and N2. Make N1 as the left child of N3 and N2 as the right child of N3. Add this new node N3 to the Minimum Heap.
4. Repeat steps 2 and 3 till the point, the Minimum Heap has only one node.

Program:

```
#include<bits/stdc++.h>
  using namespace std;
//Huffman tree node
  struct MinHeapNode{
      char data;
      int freq;
      MinHeapNode *left,*right;
      MinHeapNode(char data,int freq)
      {
          left=right=NULL;
          this->data=data;
          this->freq=freq;
      }
};

//For comparison of two nodes.
struct compare{
    bool operator()(MinHeapNode *l,MinHeapNode *r)
    {
        return (l->freq>r->freq);
    }
};

//Print Huffman Codes
void printCodes(struct MinHeapNode* root,string str){
   //If root is Null then return.
   if(!root){
       return;
    }
```

```
//If the node's data is not '$' that means it's not an internal node and
print the string.
   if(root->data!='$') {
       cout<<root->data<<": "<<str<<endl;
   }
   printCodes(root->left,str+"0");
   printCodes(root->right,str+"1");
}

//Build Huffman Tree
void HuffmanCodes(char data[],int freq[],int size){
struct MinHeapNode *left,*right,*top,*tmp;
//create a min heap.
priority_queue<MinHeapNode*,vector<MinHeapNode*>,compare>
minheap;
// For each character create a leaf node and insert each leaf node in
the heap.
for(int i=0;i<size;i++)
{
    minheap.push(new MinHeapNode(data[i],freq[i]));
 }
 //Iterate while size of min heap doesn't become 1
 while(minheap.size()!=1) {
     //Extract two nodes from the heap.
     left = minheap.top();
     minheap.pop();
     right = minheap.top();
     minheap.pop();

 //Create a new internal node having frequency equal to the sum of
two extracted nodes. Assign '$' to this node and make the two
extracted node as left and right children of this new node. Add this
node to the heap.

     tmp = new MinHeapNode('$',left->freq+right->freq);
     tmp->left = left;
     tmp->right = right;
     minheap.push(tmp);
 }
 //Calling function to print the codes.
 printCodes(minheap.top()," ");
}

int main()
{
  char arr[] = {'a','b','c','d','e'};
  int freq[] = {10,5,2,14,15};
  int size=5;
 HuffmanCodes(arr,freq,size);
 return 0;
}
```

Output:

```
c: 000
b: 001
a: 01
d: 10
e: 11
```

Result:

Thus a C++ program has been written and executed for implementing Activity Selection Problem and Huffman coding.