WEEK-8

Aim: Apply Greedy method to compress the given data using Huffman encoding.

Code:

```
#include<iostream>
#include<cstdlib>
using namespace std;
#define MAX_TREE_HT 100
struct MinHeapNode
{
char data;
unsigned freq;
struct MinHeapNode *left, *right;
};
struct MinHeap
{
unsigned size;
unsigned capacity;
struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq)
{
struct MinHeapNode* temp=(struct
MinHeapNode*)malloc(sizeof(struct MinHeapNode));
```

```
temp->left = temp->right = NULL;
temp->data = data;
temp->freq = freq;
return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
struct MinHeap* minHeap=(struct
MinHeap*)malloc(sizeof(struct MinHeap));
minHeap->size = 0;
minHeap->capacity = capacity;
minHeap->array= (struct
MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
MinHeapNode*));
return minHeap;
}
void swapMinHeapNode(struct MinHeapNode** a,struct
MinHeapNode** b)
{
struct MinHeapNode* t = *a;
*a = *b;
*b = t;
}
void minHeapify(struct MinHeap* minHeap, int idx)
{
```

```
int smallest = idx;
int left = 2 * idx + 1;
int right = 2 * idx + 2;
if (left < minHeap->size && minHeap->array[left]->freq <
minHeap->array[smallest]->freq)
smallest = left;
if (right < minHeap->size && minHeap->array[right]->freq
< minHeap->array[smallest]->freq)
smallest = right;
if (smallest != idx)
{
swapMinHeapNode(&minHeap-
>array[smallest],&minHeap->array[idx]);
minHeapify(minHeap, smallest);
}
}
int isSizeOne(struct MinHeap* minHeap)
{
return (minHeap->size == 1);
}
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
struct MinHeapNode* temp = minHeap->array[0];
minHeap->array[0]=minHeap->array[minHeap->size - 1];
--minHeap->size;
```

```
minHeapify(minHeap, 0);
return temp;
}
void insertMinHeap(struct MinHeap* minHeap,struct
MinHeapNode* minHeapNode)
{
++minHeap->size;
int i = minHeap->size - 1;
while (i && minHeapNode->freq < minHeap->array[(i - 1) /
2]->freq)
{
minHeap->array[i] = minHeap->array[(i - 1) / 2];
i = (i - 1) / 2;
}
minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap* minHeap)
{
int n = minHeap->size - 1;
int i;
for (i = (n - 1) / 2; i >= 0; --i)
minHeapify(minHeap, i);
}
void printArr(int arr[], int n)
{
```

```
int i;
for (i = 0; i < n; ++i)
cout<< arr[i];
cout<<"\n";
}
int isLeaf(struct MinHeapNode* root)
{
return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char data[], int freq[],
int size)
{
struct MinHeap* minHeap = createMinHeap(size);
for (int i = 0; i < size; ++i)
minHeap->array[i] = newNode(data[i], freq[i]);
minHeap->size = size;
buildMinHeap(minHeap);
return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[], int freq[],
int size)
{
struct MinHeapNode *left, *right, *top;
struct MinHeap* minHeap = createAndBuildMinHeap(data,
freq, size);
```

```
while (!isSizeOne(minHeap))
{
left = extractMin(minHeap);
right = extractMin(minHeap);
top = newNode('$', left->freq + right->freq);
top->left = left;
top->right = right;
insertMinHeap(minHeap, top);
}
return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[], int top)
{
if (root->left)
{
arr[top] = 0;
printCodes(root->left, arr, top + 1);
}
if (root->right)
{
arr[top] = 1;
printCodes(root->right, arr, top + 1);
}
if (isLeaf(root))
{
```

```
cout<< root->data <<": ";
printArr(arr, top);
}
}
void HuffmanCodes(char data[], int freq[], int size)
{
struct MinHeapNode* root= buildHuffmanTree(data, freq,
size);
int arr[MAX_TREE_HT], top = 0;
printCodes(root, arr, top);
}
int main()
{
char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
int freq[] = { 5, 9, 12, 13, 16, 45 };
int size = sizeof(arr) / sizeof(arr[0]);
HuffmanCodes(arr, freq, size);
return 0;
}
```

Output:

```
f: 0
c: 100
d: 101
a: 1100
b: 1101 e: 111
```

WEEK-9

Aim: Implement fractional knapsack problem using Greedy Strategy

Code:

```
#include<stdio.h>
void main()
{
 int n,i,j,count=0;
 printf("Enter number of weights\n");
 scanf("%d",&n);
 float w[n],p[n],r[n],m,max_pro=0;
 printf("Enter weights\n");
 for(i=0;i<n;i++)
 {
 scanf("%f",&w[i]);
 }
 printf("Enter profits to corresponding weights\n");
 for(i=0;i<n;i++)
 {
 scanf("%f",&p[i]);
 }
 printf("Enter maximum weight\n");
 scanf("%f",&m);
 for(i=0;i<n;i++)
 r[i]=p[i]/w[i];
```

```
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1;j++)
{
if(r[j+1]>r[j])
{
r[j+1]=r[j+1]+r[j];
r[j]=r[j+1]-r[j];
r[j+1]=r[j+1]-r[j];
w[j+1]=w[j+1]+w[j];
w[j]=w[j+1]-w[j];
w[j+1]=w[j+1]-w[j];
p[j+1]=p[j+1]+p[j];
p[j]=p[j+1]-p[j];
p[j+1]=p[j+1]-p[j];
}
}
}
printf("Profit/weight array is\n");
for(i=0;i<n;i++)
{
printf("%f ",r[i]);
}
for(i=0;i<n;i++)
{
```

```
 if(m>=w[i])
 {
 max_pro=max_pro+p[i];
 m=m-w[i];
 }
 else if(m<w[i]&&count==0)
 {
 max_pro=max_pro+(m*p[i])/w[i];
 count=1;
 }
 }
 printf("\nMaximum profit is %f",max_pro);
}
```

Output:

```
Enter number of weights
4
Enter weights
2 4 6 9
Enter profits to corresponding weights
10 10 12 18
Enter maximum weight
15
Profit/weight array is
5.000000 2.500000 2.000000 2.000000
Maximum profit is 38.000000
```

Week-10

Aim: Implement minimum spanning tree using Prim's algorithm and analyse its time complexity

Code:

```
#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#include <time.h>

int n;

int minkey(int key[], bool mstset[]) {

int min_index, min = INT_MAX;

for (size_t i = 0; i < n; i++) {

if (mstset[i] == false && key[i] < min) {

min = key[i];

min_index = i;

}

}

return min_index;

}

int printmst(int parent[], int graph[n][n])

{

printf("Edge\t Weight\n");

for (int i = 1; i < n; i++) {

printf("%d - %d\t %d\n", parent[i], i, graph[i][parent[i]]);
```

```
}
}
int prims(int graph[n][n])
{
int parent[n], key[n];
bool mstset[n];
for (int i = 1; i < n; i++)
{
key[i] = INT_MAX, mstset[i] = false;
}
key[0] = 0;
parent[0] = -1;
for (int i = 0; i < n - 1; i++)
{
int u = minkey(key, mstset);
mstset[u] = true;
for (int j = 0; j < n; j++)
{
if (graph[u][j] && mstset[j] == false && graph[u][j] < key[j]) {
parent[j] = u,
key[j] = graph[u][j];
}
}
}
printmst(parent, graph);
```

```
}
int main()
{
printf("Enter the number of nodes = ");
scanf("%d", &n);
printf("Enter the Adjacency Matrix interms of costs: ");
int graph[n][n];
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
{
scanf("%d", &graph[i][j]);
}
}
clock_t start = clock();
prims(graph);
clock_t end = clock();
double t = ((double)(end-start)/CLOCKS_PER_SEC);
printf("Time taken is %lf",t);
return 0;
}
```

Output:

```
Enter the number of nodes=5
Enter the Adjacency Matrix interms of costs:
0 0 3 0 0
```

0 0 10 4 0

3 10 0 2 6

0 4 2 0 1

0 0 6 1 0

Edge Weight

3 – 1 4

0 – 2 3

2 – 3 2

3 – 4 1

Time taken is 0.004000