

DESIGN AND ANALYSIS OF ALGORITHMS – GREEDY ALGORITHM

Name - ANIKET SHARMA

Reg No. - RA1911003030436

Branch- CSE – I

IMPLEMENT HUFFMAN CODING USING GREEDY ALGORITHM:

```
1  #include <iostream>
2  using namespace std;
3  #define MAX_TREE_HT 50
4
5  struct MinHNode {
6      unsigned freq;
7      char item;
8      struct MinHNode* left, * right;
9  };
10 struct MinH {
11     unsigned size;
12     unsigned capacity;
13     struct MinHNode** array;
14 };
15
16 struct MinHNode* newNode(char item, unsigned freq) {
17     struct MinHNode* temp = (struct MinHNode*)malloc(sizeof(struct MinHNode));
18     temp->left = temp->right = NULL;
19     temp->item = item;
20     temp->freq = freq;
21
22     return temp;
23 }
24
25 struct MinH* createMinH(unsigned capacity) {
26     struct MinH* minHeap = (struct MinH*)malloc(sizeof(struct MinH));
27     minHeap->size = 0;
28     minHeap->capacity = capacity;
29     minHeap->array = (struct MinHNode**)malloc(minHeap->capacity * sizeof(struct MinHNode));
30
31     return minHeap;
32 }
33
34 void printArray(int arr[], int n) {
35     int i;
36     for (i = 0; i < n; ++i)
37     {
38         cout << arr[i];
39     }
40     cout << "\n";
41 }
42
43 void swapMinHNode(struct MinHNode** a, struct MinHNode** b) {
44     struct MinHNode* t = *a;
45     *a = *b;
46     *b = t;
47 }
48
49 void minHeapify(struct MinH* minHeap, int idx) {
50     int smallest = idx;
51     int left = 2 * idx + 1;
52     int right = 2 * idx + 2;
53
54     if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq) {
55         smallest = left;
56     }
57     if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq) {
58         smallest = right;
59     }
60     if (smallest != idx) {
61         swapMinHNode(&minHeap->array[smallest],
62                     &minHeap->array[idx]);
63         minHeapify(minHeap, smallest);
64     }
65 }
66
67 int checkSizeOne(struct MinH* minHeap) {
68     return (minHeap->size == 1);
69 }
70
71 struct MinHNode* extractMin(struct MinH* minHeap) {
72     struct MinHNode* temp = minHeap->array[0];
73     minHeap->array[0] = minHeap->array[minHeap->size - 1];
74     --minHeap->size;
75     minHeapify(minHeap, 0);
76
77     return temp;
78 }
79
80 void insertMinHeap(struct MinH* minHeap, struct MinHNode* minHeapNode) {
81     ++minHeap->size;
82     int i = minHeap->size - 1;
83     while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
84         minHeap->array[i] = minHeap->array[(i - 1) / 2];
85         i = (i - 1) / 2;
86     }
87     minHeap->array[i] = minHeapNode;
88 }
89
90 void buildMinHeap(struct MinH* minHeap) {
91     int n = minHeap->size - 1;
92     int i;
93     for (i = (n - 1) / 2; i >= 0; --i) {
94         minHeapify(minHeap, i);
95     }
96 }
```

```

97 int isLeaf(struct MinHNode* root) {
98     return !(root->left) && !(root->right);
99 }
100
101 struct MinH* createAndBuildMinHeap(char item[], int freq[], int size) {
102     struct MinH* minHeap = createMinH(size);
103     for (int i = 0; i < size; ++i) {
104         minHeap->array[i] = newNode(item[i], freq[i]);
105     }
106     minHeap->size = size;
107     buildMinHeap(minHeap);
108
109     return minHeap;
110 }
111
112 struct MinHNode* buildHfTree(char item[], int freq[], int size) {
113     struct MinHNode* left, * right, * top;
114     struct MinH* minHeap = createAndBuildMinHeap(item, freq, size);
115
116     while (!checkSizeOne(minHeap)) {
117         left = extractMin(minHeap);
118         right = extractMin(minHeap);
119         top = newNode('$', left->freq + right->freq);
120         top->left = left;
121         top->right = right;
122         insertMinHeap(minHeap, top);
123     }
124     return extractMin(minHeap);
125 }
126
127 void printHCodes(struct MinHNode* root, int arr[], int top) {
128     if (root->left) {
129         arr[top] = 0;
130         printHCodes(root->left, arr, top + 1);
131     }
132     if (root->right) {
133         arr[top] = 1;
134         printHCodes(root->right, arr, top + 1);
135     }
136     if (isLeaf(root)) {
137         cout << root->item << " | ";
138         printArray(arr, top);
139     }
140 }
141
142 void HuffmanCodes(char item[], int freq[], int size) {
143     struct MinHNode* root = buildHfTree(item, freq, size);
144     int arr[MAX_TREE_HT], top = 0;
145     printHCodes(root, arr, top);
146 }
147
148 int main() {
149     char arr[] = { 'A', 'B', 'C', 'D', 'E', 'F' };
150     int freq[] = { 5, 9, 12, 13, 16, 45 };
151     int size = sizeof(arr) / sizeof(arr[0]);
152     cout << "Char | Huffman code ";
153     HuffmanCodes(arr, freq, size);
154     return 0;
155 }

```

OUTPUT:

```

Microsoft Visual Studio Debug Console
Char | Huffman code
-----
F | 0
C | 100
D | 101
A | 1100
B | 1101
E | 111

C:\Users\devan\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe (process 3700) exited with code 0.
Press any key to close this window . . .

```

IMPLEMENT FRACTIONAL KNAPSACK USING GREEDY ALGORITHM:

```

1  #include <iostream>
2  #include<math.h>
3  #include<algorithm>
4  using namespace std;
5  class item
6  {
7  public: int value, weight; item(int value, int weight) { this->value=value; this->weight=weight; }
8  };
9  bool cmp(item a,item b)
10 {
11     double r1 = (double)a.value / (double)a.weight;
12     double r2 = (double)b.value / (double)b.weight;
13     return r1 > r2;
14 }
15 double fractionalKnapsack(int w,item arr[], int n)
16 {
17     sort(arr, arr + n, cmp);
18     int curweight = 0;
19     double finalvalue = 0.0;
20     for (int i = 0; i < n; i++)
21     {
22         if (curweight + arr[i].weight <= w)
23         {
24             curweight += arr[i].weight;
25             finalvalue += arr[i].value;
26         }
27         else
28         {
29             int remain = w - curweight;
30             finalvalue += arr[i].value * ((double)remain / (double)arr[i].weight);
31             break;
32         }
33     }
34     return finalvalue;
35 }
36 int main()
37 {
38     int w = 50;
39     item arr[] = { {60,10}, {100,20}, {120,30} };
40     int n = sizeof(arr) / sizeof(arr[0]);
41     cout << "Maximum value we can obtain = "<< fractionalKnapsack(w,arr,n);
42     return 0;
}

```

OUTPUT:

```

Microsoft Visual Studio Debug Console
Maximum value we can obtain = 240
C:\Users\devan\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe (process 5740) exited with code 0.
Press any key to close this window . . .

```