**Student Name:- Chaudhary Hamdan**

**Student Roll No.:- 1905387**

**Algorithm Lab. Class Assignment-11**

**CSE Group 1**

**Date: - 8th October 2021**

1. **Write a program to implement the file or code compression using Huffman’s algorithm.**

**Program**

// Author: Chaudhary Hamdan

// Generated at: Fri Oct 8 12:55:20 2021

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

#define pf printf

#define pfs(x) printf("%d ", x)

#define pfn(x) printf("%d\n", x)

#define pfc(x) printf("%d, ", x)

#define FI(i,x,y,inc) for(int i = x; i < y; i += inc)

#define F(i,x,y) FI(i, x, y, 1)

#define F0(i,n) FI(i, 0, n, 1)

#define RF(i,x,y) for(int i = x; i >= y; i--)

#define sfiarr(i,a,n) int a[n]; for(int i = 0; i < n; i++) scanf("%d",&a[i])

#define sfcarr(i,a,n) char a[n]; for(int i = 0; i < n; i++){ scanf("%c",&a[i]); scanf("%c",&a[i]); }

#define pfiarr(i,a,n) for(int i = 0; i < n-1; i++) pfs(a[i]); pfn(a[n-1])

#define pfcarr(i,a,n) for(int i = 0; i < n-1; i++) printf("%c ",a[i]); printf("%c ",a[n-1])

#define MAX\_TREE\_HT 100

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

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)

printf("%d", arr[i]);

printf("\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)) {

printf("%c: ", 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() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int n;

sf(n);

sfcarr(i, arr, n)

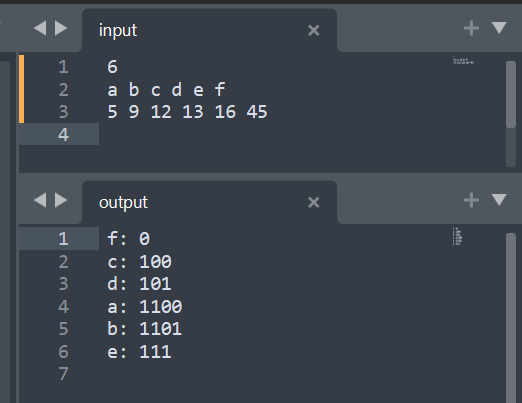
sfiarr(i, freq, n);

HuffmanCodes(arr, freq, n);

return 0;

}

**Output**

****

1. **Write a C program to implement Breadth First Search.**

**Program**

// Author: Chaudhary Hamdan

// Generated at: Fri Oct 8 14:06:41 2021

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

#define pf printf

#define pfs(x) printf("%d ", x)

#define pfn(x) printf("%d\n", x)

#define pfc(x) printf("%d, ", x)

#define FI(i,x,y,inc) for(int i = x; i < y; i += inc)

#define F(i,x,y) FI(i, x, y, 1)

#define F0(i,n) FI(i, 0, n, 1)

#define RF(i,x,y) for(int i = x; i >= y; i--)

#define pfarr(i,a,n) for(int i = 0; i < n-1; i++) pfs(a[i]); pfn(a[n-1]);

#define SIZE 40

void i\_o\_from\_file();

struct queue {

int items[SIZE];

int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph {

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

void bfs(struct Graph\* graph, int startVertex) {

struct queue\* q = createQueue();

graph->visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf("Visited %d\n", currentVertex);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp->next;

}

}

}

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

struct queue\* createQueue() {

struct queue\* q = malloc(sizeof(struct queue));

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\* q, int value) {

if (q->rear == SIZE - 1)

printf("\nQueue is Full!!");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\* q) {

int item;

if (isEmpty(q)) {

printf("Queue is empty");

item = -1;

}

else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

// printf("Resetting queue ");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\* q) {

int i = q->front;

if (isEmpty(q)) {

printf("Queue is empty");

} else {

printf("\nQueue contains \n");

for (i = q->front; i < q->rear + 1; i++) {

printf("%d ", q->items[i]);

}

}

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

}

bfs(graph, 0);

return 0;

}

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

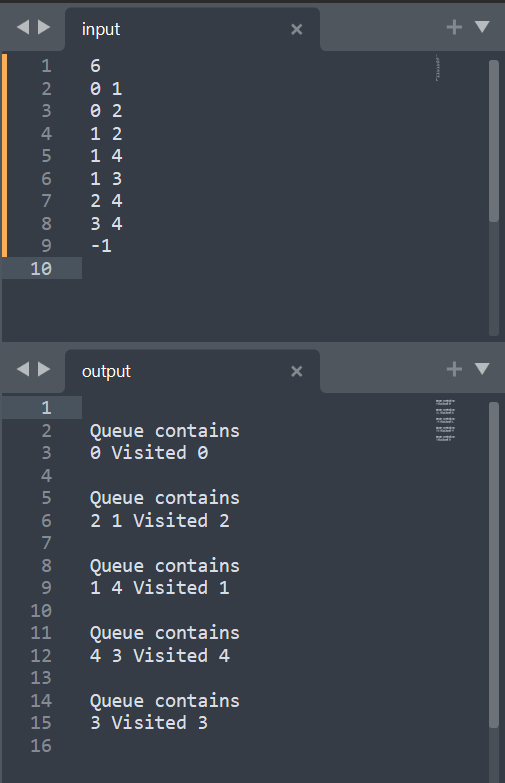
freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

**Output**

****