**ASSIGNMENT-10**

**(ALGO LAB)**

**ACTIVITY SELECTION PROBLEM**

#include<stdio.h>

#include<stdlib.h>

struct Table

{

int st;

int ft;

int index;

};

typedef struct Table table;

swap(table \*a,table \*b)

{

table temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void sort(table a[],int n)

{

int min,i,j;

for(i=0;i<n-1;i++)

{

min=i;

for(j=i+1;j<n;j++)

{

if(a[j].ft<a[i].ft)

min=j;

}

if(i!=min)

swap(&a[i],&a[min]);

}

}

activityselection(int n,table a[])

{

sort(a,n);

int current=0;

int j,i;

int ans[n];

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

ans[i]=0;

ans[a[0].index]=1;

current=0;

for(j=1;j<n;j++)

{

if(a[j].st>a[current].ft)

{

ans[a[j].index]=1;

current=j;

}

}

printf("\nselected activity:\n");

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

{

if(ans[i]==1)

printf("A%d |",i);

}

}

int main()

{

int n,i;

printf("enter number of activity");

scanf("%d",&n);

table a[n];

printf("enter starting time and finishing time");

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

{

scanf("%d",&a[i].st);

scanf("%d",&a[i].ft);

a[i].index=i;

}

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

printf("%d | ",a[i].index);

printf("\n");

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

printf("%d | ",a[i].st);

printf("\n");

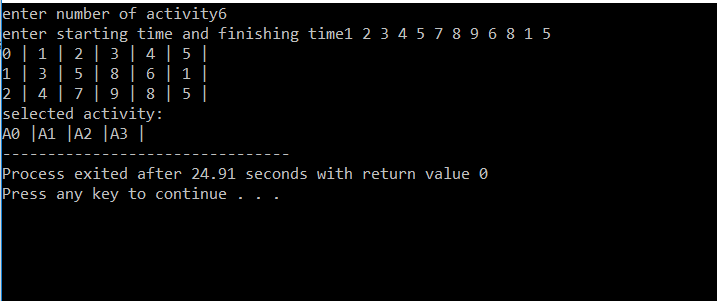
for(i=0;i<n;i++)

printf("%d | ",a[i].ft);

activityselection(n,a);

}

// 1 2 3 4 5 7 8 9 6 8 1 5

****

**HUFFMAN ENCODING**

#include <stdio.h>

#include <stdlib.h>

#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)

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()

{

char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };

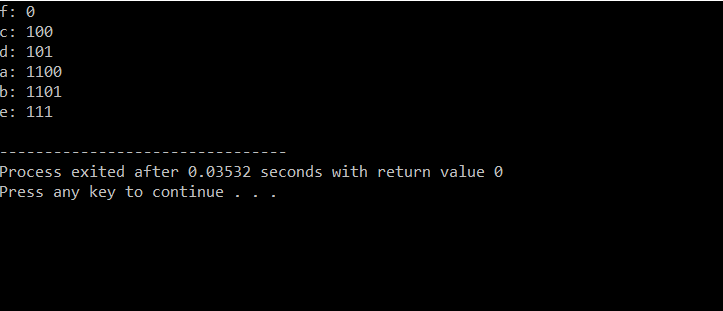
int freq[] = { 5, 9, 12, 13, 16, 45 };

int size = sizeof(arr) / sizeof(arr[0]);

HuffmanCodes(arr, freq, size);

return 0;

}



**KNAPSACK PROBLEM**

//3 21 2 16 4 36 5 50 6 36 1 5

#include<stdio.h>

#include<stdlib.h>

struct knapsack

{

int index;

double w;

double p;

double pd;

};

typedef struct knapsack ks;

swap(ks \*a,ks \*b)

{

ks temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void sort(ks a[],int n)

{

int max,i,j;

for(i=0;i<n-1;i++)

{

max=i;

for(j=i+1;j<n;j++)

{

if(a[j].pd>a[i].pd)

max=j;

}

if(i!=max)

swap(&a[i],&a[max]);

}

}

void fks(int limit,ks knap[],int n)

{

sort(knap,n);

int i;

float fraction[n];

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

fraction[i]=0.000000;

float temp=0;

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

{

if((temp+knap[i].w)<=limit)

{

fraction[knap[i].index]=1.000000;

temp=temp+knap[i].w;

}

else

{

float f=(limit-temp)/knap[i].w;

fraction[knap[i].index]=f;

break;

}

}

printf("\n");

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

printf("%d---> %lf \n",i,fraction[i]);

}

int main()

{

int n,i,limit;

printf("enter number of items");

scanf("%d",&n);

ks knap[n];

printf("eneter weight and profit of objects");

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

{

scanf("%lf",&knap[i].w);

scanf("%lf",&knap[i].p);

knap[i].index=i;

knap[i].pd=knap[i].p/knap[i].w;

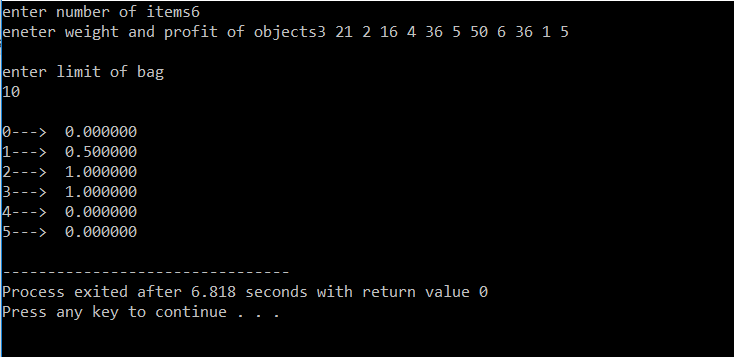
}

printf("\nenter limit of bag\n");

scanf("%d",&limit);

fks(limit,knap,n);

}



**ALL PERMUTATION PROBLEM**

#include<stdio.h>

#include<string.h>

void ap(char s[],int lb,int ub)

{

int i;

char c;

if(lb==ub)

{

printf("%s\n",s);

return;

}

for(i=lb;i<=ub;i++)

{

c=s[i];

s[i]=s[lb];

s[lb]=c;

ap(s,lb+1,ub);

c=s[i];

s[i]=s[lb];

s[lb]=c;

}

}

int main()

{

char s[20];

int n;

printf("Enter the string\n");

scanf("%s",s);

n=strlen(s);

printf("All permutations:\n");

ap(s,0,n-1);

}

