main.c

Run

```c
#include<stdio.h>
int main(){
int a[2][2],b[2][2],c[2][2],i,j;
int m1,m2,m3,m4,m5,m6,m7;
printf("Enter the 4 elements of first matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&a[i][j]);
printf("Enter the 4 elements of second matrix: ");
for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
printf("\nThe first matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",a[i][j]);
}
printf("\nThe second matrix is\n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",b[i][j]);
}
m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
m2= (a[1][0]+a[1][1])*b[0][0];
m3= a[0][0]*(b[0][1]-b[1][1]);
m4= a[1][1]*(b[1][0]-b[0][0]);
m5= (a[0][0]+a[0][1])*b[1][1];
m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
c[0][0]=m1+m4-m5+m7;
c[0][1]=m3+m5;
c[1][0]=m2+m4;
c[1][1]=m1-m2+m3+m6;
printf("\nAfter multiplication \n");
for(i=0;i<2;i++){
printf("\n");
for(j=0;j<2;j++)
printf("%d\t",c[i][j]);
}
return 0;
}
```

Output Clear

```
/tmp/46rEWP1COg.o
Enter the 4 elements of first matrix: 2
0
1
9
Enter the 4 elements of second matrix: 3
9
4
7
The first matrix is

2   0
1   9
The second matrix is

3   9
4   7
After multiplication

6   18
39  72
```

main.c

Run

```c
#include <limits.h>
#include <stdio.h>
void recursiveMinMax(int arr[], int N,
int* minE, int* maxE)
{
if (N < 0) {
return;
}
if (arr[N] < *minE) {
*minE = arr[N];
}
if (arr[N] > *maxE) {
*maxE = arr[N];
}
recursiveMinMax(arr, N - 1, minE, maxE);
}
void findMinimumMaximum(int arr[], int N)
{
int i;
int minE = INT_MAX, maxE = INT_MIN;
recursiveMinMax(arr, N - 1, &minE, &maxE);
printf("The minimum element is %d", minE);
printf("\n");
printf("The maximum element is %d", maxE);
return;
}
int main()
```

Output Clear

```
/tmp/46rEWP1C0g.o
The minimum element is -1
The maximum element is 4
```

main.c

Run

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
typedef struct point
{
double x;
double y;
}POINT,VECTOR;
POINT b[1000];
VECTOR normal;
int n;
int upper_lower(int i, VECTOR ab, double c) {
double x, y,result;
y = b[i].y;
x = normal.x*b[i].x;
result = -(x + c) / normal.y;
if (y>result) return 1;
if (y == result) return 0;
else
return -1;
}
int ccw(VECTOR v,VECTOR v2)
{
double cp;
cp = v2.x*v.y - v2.y*v.x;
if (cp == abs(cp)) return 1;
else
return -1;
}
double vector_length(VECTOR v)
{
return sqrt(pow(v.x, 2) + pow(v.y, 2));
}
int cmp_points(const void *p1, const void *p2)
{
const POINT *pt1 = p1;
const POINT *pt2 = p2;
if (pt1->x > pt2->x)
return 1;
if (pt1->x < pt2->x)
return -1;
if (pt1->y > pt2->y)
return 1;
```

```
45  if (pt1->y < pt2->y)
46  return -1;
47  return 0;
48  }
49  int main()
50  {
51  int i,poloha,upper[1000],lower[1000],h=0,d=0;
52  scanf("%d", &n);
53  if (n <= 0 && n > 1000) return 0;
54  for (i = 0; i < n; i++)
55  {
56  scanf("%lf %lf", &b[i].x, &b[i].y);
57  }
58  qsort(b, n, sizeof(POINT), cmp_points);
59  VECTOR ab;
60  double c;
61  ab.x = b[n - 1].x - b[0].x;
62  ab.y = b[n - 1].y - b[0].y;
63  normal.x = -ab.y;
64  normal.y = ab.x;
65  c = -normal.x*b[0].x - (normal.y*b[0].y);
66  for (i = 0; i < n; i++)
67  {
68  poloha = upper_lower(i,ab,c);
69  if (poloha == 1) upper[h++] = i;
70  if (poloha == -1) lower[d++]=i;
71  if (poloha == 0)
72  {
73  upper[h++] = i;
74  lower[d++] = i;
75  }
76  }
77  int j = 0;
78  double v, length = 0;
79  VECTOR v1, v2, v3,v4;
80  v3.x = 0; v3.y = 0;
81  for (i = 0; ; i++)
82  {
83  int in = 0;
84  if (lower[i + 2] < 0)
```

```
{
v1.x = b[lower[i + 1]].x - b[lower[0]].x;
v1.y = b[lower[i + 1]].y - b[lower[0]].y;
v2.x = b[lower[i]].x - b[lower[i + 1]].x;
v2.y = b[lower[i]].y - b[lower[i + 1]].y;
length += vector_length(v1);
length += vector_length(v2);
break;
}
v1.x = b[lower[i + 1]].x - b[lower[i]].x;
v1.y = b[lower[i + 1]].y - b[lower[i]].y;
v2.x = b[lower[i + 2]].x - b[lower[i]].x;
v2.y = b[lower[i + 2]].y - b[lower[i]].y;
in = ccw(v1, v2);
if (in == 1)
{
length += vector_length(v1);
v3 = v2;
v4 = v1;
}
if (in == -1)
{
length -= vector_length(v4);
if (v3.x != 0 && v3.y != 0)
{
length += vector_length(v3);
v3.x = 0; v3.y = 0;
}
else
{
length += vector_length(v2);
}
}
}
printf("%.3lf", length);
return 0;
}
```

Output Clear

```
/tmp/icrv10mxww.o
4
0 3
4 4
1 0
0 0
```

main.c

Run

```c
#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;
```

```
45 if (left < minHeap->size
46 && minHeap->array[left]->freq
47 < minHeap->array[smallest]->freq)
48 smallest = left;
49 if (right < minHeap->size
50 && minHeap->array[right]->freq
51 < minHeap->array[smallest]->freq)
52 smallest = right;
53 if (smallest != idx) {
54 swapMinHeapNode(&minHeap->array[smallest],
55 &minHeap->array[idx]);
56 minHeapify(minHeap, smallest);
57 }
58 }
59 int isSizeOne(struct MinHeap* minHeap)
60 {
61 return (minHeap->size == 1);
62 }
63 struct MinHeapNode* extractMin(struct MinHeap* minHeap)
64 {
65 struct MinHeapNode* temp = minHeap->array[0];
66 minHeap->array[0] = minHeap->array[minHeap->size - 1];
67 --minHeap->size;
68 minHeapify(minHeap, 0);
69 return temp;
70 }
71 void insertMinHeap(struct MinHeap* minHeap,
72 struct MinHeapNode* minHeapNode)
73 {
74 ++minHeap->size;
75 int i = minHeap->size - 1;
76 while (i
77 && minHeapNode->freq
78 < minHeap->array[(i - 1) / 2]->freq) {
79 minHeap->array[i] = minHeap->array[(i - 1) / 2];
80 i = (i - 1) / 2;
81 }
82 minHeap->array[i] = minHeapNode;
83 }
84 void buildMinHeap(struct MinHeap* minHeap)
85 {
86 int n = minHeap->size - 1;
87 int i;
88 for (i = (n - 1) / 2; i >= 0; --i)
89 minHeapify(minHeap, i);
90 }
```

```
91  void printArr(int arr[], int n)
92  {
93  int i;
94  for (i = 0; i < n; ++i)
95  printf("%d", arr[i]);
96  printf("\n");
97  }
98  int isLeaf(struct MinHeapNode* root)
99  {
100 return !(root->left) && !(root->right);
101 }
102 struct MinHeap* createAndBuildMinHeap(char data[],
103 int freq[], int size)
104 {
105 struct MinHeap* minHeap = createMinHeap(size);
106 for (int i = 0; i < size; ++i)
107 minHeap->array[i] = newNode(data[i], freq[i]);
108 minHeap->size = size;
109 buildMinHeap(minHeap);
110 return minHeap;
111 }
112 struct MinHeapNode* buildHuffmanTree(char data[],
113 int freq[], int size)
114 {
115 struct MinHeapNode *left, *right, *top;
116 struct MinHeap* minHeap
117 = createAndBuildMinHeap(data, freq, size);
118 while (!isSizeOne(minHeap)) {
119 left = extractMin(minHeap);
120 right = extractMin(minHeap);
121 top = newNode('$', left->freq + right->freq);
122 top->left = left;
123 top->right = right;
124 insertMinHeap(minHeap, top);
125 }
126 return extractMin(minHeap);
127 }
128 void printCodes(struct MinHeapNode* root, int arr[],
129 int top)
130 {
131 if (root->left) {
132 arr[top] = 0;
133 printCodes(root->left, arr, top + 1);
134 }
```

```
135 if (root->right) {
136 arr[top] = 1;
137 printCodes(root->right, arr, top + 1);
138 }
139 if (isLeaf(root)) {
140 printf("%c: ", root->data);
141 printArr(arr, top);
142 }
143 }
144 void HuffmanCodes(char data[], int freq[], int size)
145 {
146 struct MinHeapNode* root
147 = buildHuffmanTree(data, freq, size);
148 int arr[MAX_TREE_HT], top = 0;
149 printCodes(root, arr, top);
150 }
151 int main()
152 {
153 char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
154 int freq[] = { 5, 9, 12, 13, 16, 45 };
155 int size = sizeof(arr) / sizeof(arr[0]);
156 HuffmanCodes(arr, freq, size);
157 return 0;
158 }
```

Output Clear

```
/tmp/46rEWP1COg.o
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

main.c

Run

```c
#include<stdio.h>
int main()
{
float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
int n,i,j;
printf("Enter the number of items :");
scanf("%d",&n);
for (i = 0; i < n; i++)
{
printf("Enter Weight and Profit for item[%d] :\n",i);
scanf("%f %f", &weight[i], &profit[i]);
}
printf("Enter the capacity of knapsack :\n");
scanf("%f",&capacity);
for(i=0;i<n;i++)
ratio[i]=profit[i]/weight[i];
for (i = 0; i < n; i++)
for (j = i + 1; j < n; j++)
if (ratio[i] < ratio[j])
{
temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp = weight[j];
weight[j] = weight[i];
weight[i] = temp;
temp = profit[j];
profit[j] = profit[i];
profit[i] = temp;
}
printf("Knapsack problems using Greedy Algorithm:\n");
for (i = 0; i < n; i++)
{
if (weight[i] > capacity)
break;
else
{
Totalvalue = Totalvalue + profit[i];
capacity = capacity - weight[i];
}
}
if (i < n)
Totalvalue = Totalvalue + (ratio[i]*capacity);
printf("\nThe maximum value is :%f\n",Totalvalue);
return 0;
}
```

Output Clear

```
/tmp/V9PDOjsmyN.o
Enter the number of items :1 2
Enter Weight and Profit for item[0] :
3 4
Enter the capacity of knapsack :
Knapsack problems using Greedy Algorithm:

The maximum value is :3.000000
```

main.c

Run

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node* left;
struct node* right;
};
struct node* newNode(int data)
{
struct node* node
= (struct node*)malloc(sizeof(struct node));
node->data = data;
node->left = NULL;
node->right = NULL;
return (node);
}
void printPostorder(struct node* node)
{
if (node == NULL)
return;
printPostorder(node->left);
printPostorder(node->right);
printf("%d ", node->data);
}
void printInorder(struct node* node)
{
if (node == NULL)
return;
printInorder(node->left);
printf("%d ", node->data);
printInorder(node->right);
}
void printPreorder(struct node* node)
{
if (node == NULL)
return;
printf("%d ", node->data);
printPreorder(node->left);
printPreorder(node->right);
}
int main(){
struct node* root = newNode(1);
root->left = newNode(2);
root->right = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);
printf("\nPreorder traversal of binary tree is \n");
printPreorder(root);
printf("\nInorder traversal of binary tree is \n");
printInorder(root);
printf("\nPostorder traversal of binary tree is \n");
printPostorder(root);
getchar();
return 0;}
```

Output Clear

```
/tmp/V9PDOjsmyN.o
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```

main.c

Run

```c
#include <stdio.h>
#include <stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
printf("\n\tImplementation of Kruskal's Algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{if(cost[i][j] < min){
min=cost[i][j];
a=u=i;
b=v=j;}}}
u=find(u);
v=find(v);
if(uni(u,v)){
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i){
while(parent[i])
i=parent[i];
return i;
}
int uni(int i,int j)
{
if(i!=j){
parent[j]=i;
return 1;
}
return 0;
}
```

Output Clear

```
/tmp/V9PDOjsmyN.o
Implementation of Kruskal's Algorithm

Enter the no. of vertices:3
Enter the cost adjacency matrix:
1 2 3
3 4 4
1 2 4
The edges of Minimum Cost Spanning Tree are
1 edge (3,1) =1
2 edge (1,2) =2

    Minimum cost = 3
```