Daa lab programs

1. Implementation of Merge Sort using divide and conquer

Code->

#include<stdio.h>

void mergesort(int [],int ,int);

void merge(int [],int,int,int);

int main() {

int a[10],n,i;

printf("Enter the number of elements");

scanf("%d",&n);

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

{

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

}

mergesort(a,0,n-1);

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

{

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

}

return 0;

}

void mergesort(int a[10],int l,int r)

{

if(l<r)

{

int mid=l+(r-l)/2;

mergesort(a,l,mid);

mergesort(a,mid+1,r);

merge(a,l,mid,r);

}

}

void merge(int a[10],int l,int m,int r)

{

int n1,n2,i,j,x[10],y[10],k;

n1=m-l+1;

n2=r-m;

for(i=0;i<n1;i++)

{

x[i]=a[l+i];

}

for(j=0;j<n2;j++)

{

y[j]=a[m+j+1];

}

i=0;j=0;k=l;

while(i<n1 && j<n2)

{

if(x[i]<=y[j])

{

a[k]=x[i];

i++;k++;

}

else{

a[k]=y[j];

j++;k++;

}

}

while(i<n1)

{

a[k++]=x[i++];

}

while(j<n2)

{

a[k++]=y[j++];

}

}

Output->

8

65 90 25 35 78 93 55 12

12 25 35 55 65 78 90 93

2.Implementation of Quick Sort using divide and conquer

Code->

#include<stdio.h>

void quicksort(int [],int ,int);

int main() {

int a[10],n,i;

printf("Enter the number of elements");

scanf("%d",&n);

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

{

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

}

quicksort(a,0,n-1);

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

{

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

}

return 0;

}

void quicksort(int a[10],int f,int l)

{ int pivot,i,j,temp;

if(f<l)

{

pivot=f;

i=f;

j=l;

while(i<j){

while(a[j]>a[pivot]){j--;}

while(a[i]<=a[pivot] && i<l){i++;}

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

temp=a[pivot];

a[pivot]=a[j];

a[j]=temp;

quicksort(a,f,j-1);

quicksort(a,j+1,l);}

}

Output->

8

65 90 25 30 78 93 55 12

1. 5 30 55 65 78 90 93

3.Implementation of Heap Sort.

Code->

#include<stdio.h>

void create(int[]);

void down\_adjust(int[],int);

int main()

{

int heap[20],last,temp,i,n;

printf("Enter the number of elements\n");

scanf("%d",&n);

heap[0]=n;

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

{

scanf("%d",&heap[i]);

}

create(heap);

while(heap[0]>1)

{

last=heap[0];

temp=heap[1];

heap[1]=heap[last];

heap[last]=temp;

heap[0]--;

down\_adjust(heap,1);

}

printf("After sorting\n");

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

{

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

}

}

void create(int heap[20])

{

int i,n;

n=heap[0];

for(i=n/2;i>=1;i--)

{

down\_adjust(heap,i);

}

}

void down\_adjust(int heap[20],int i)

{

int flag=1,j,temp,n;

n=heap[0];

while(2\*i<=n && flag==1)

{

j=2\*i;

if(j+1<=n && heap[j+1]>heap[j])

{

j=j+1;

}

if(heap[i]>heap[j])

{

flag=0;

}

else{

temp=heap[i];

heap[i]=heap[j];

heap[j]=temp;

i=j;

}

}

}

Input->8

65 90 25 35 78 93 55 12

Output->

After sorting

12 25 35 55 65 78 90 93

4. Implementation of Binary Search using divide and conquer.

Code->

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,low,high,d,mid,i,key,a[10],flag;

printf("Enter number of elements");

scanf("%d",&n);

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

{

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

}

printf("Enter the number u want to search->");

scanf("%d",&key);

low=0;

high=n-1;

while(low<=high)

{

mid=low+(high-low)/2;

if(key==a[mid])

{

flag=1;

break;

}

if(key<a[mid])

{

high=mid-1;

}

else{

low=mid+1;

}

}

if(flag==0)

{

printf("Not found");

}

if(flag==1)

{

printf("%d is found at %d",key,mid+1);

}

return 0;

}

Output->

5

1 2 3 4 5

3

3 is found at 3

5. Implementation of Finding Min Max using divide and conquer

Code->

#include<stdio.h>

void minmax(int,int);

int a[20],n,max,min;

int main()

{

int i;

printf("Enter number of elements");

scanf("%d",&n);

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

{

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

}

minmax(0,n-1);

printf("Maximum element %d\n",max);

printf("Minimum element %d\n",min);

return 0;

}

void minmax(int i,int j)

{

int mid,max1,min1;

if(i==j)

{

max=a[i];min=a[i];

}

else{

if(i==j-1)

{

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

{

max=a[i];min=a[j];

}

else{

max=a[j];min=a[i];

}

}

else{

mid=(i+j)/2;

minmax(i,mid);

max1=max;min1=min;

minmax(mid+1,j);

if(max<max1){max=max1;}

if(min>min1){min=min1;}

}}

}

Output->

5

4 2 1 5 3

Maximum element 5

Minimum element 1

6. Implementation of BFS Traversal using adjacency matrix

Code->

#include<stdio.h>  
//int insert(int);  
void add(int);  
int delete();  
void bfs(int);  
//void display();  
int front=-1,rear=-1;  
int a[10][10],n,que[10],visited[10];  
void main()  
{  
    int i,j;  
    printf("Enter the number of nodes->");  
    scanf("%d",&n);  
    for(i=0;i<n;i++)  
    {  
        for(j=0;j<n;j++)  
        {  
            scanf("%d",&a[i][j]);  
        }  
    }  
    for(i=0;i<=n;i++)  
    {  
        visited[i]=0;  
    }  
    bfs(1);  
    //display();  
}

void bfs(int s)  
{  
int p,i;  
add(s);  
visited[s]=1;  
p=delete();  
if(p!=0)  
printf(" %d",p);  
while(p!=0)  
{  
for(i=1;i<=n;i++)  
if((a[p][i]!=0)&&(visited[i]==0))  
{  
add(i);  
visited[i]=1;  
}  
p=delete();  
if(p!=0)  
printf(" %d ",p);  
}  
for(i=1;i<=n;i++)  
if(visited[i]==0)  
bfs(i);  
}  
void add(int item)  
{  
if(rear==19)  
printf("QUEUE FULL");  
else  
{  
if(rear==-1)  
{  
que[++rear]=item;  
front++;  
}  
else  
que[++rear]=item;  
}  
}  
int delete()  
{  
int k;  
if((front>rear)||(front==-1))  
return(0);  
else  
{  
k=que[front++];  
return(k);  
}  
}

output->

Enter the number of nodes->4

0 1 1 0

1 0 1 1

1 1 0 1

0 1 1 0

 1 2  3  4

7. Implementation of BFS Traversal using adjacency list

Code->

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*next;

}Node;

int visited[30],queue[20],front=-1,rear=-1;

void add(Node \*\*head,int j)

{

Node \*newNode=(Node \*)malloc(sizeof(Node));

newNode->data=j;

newNode->next=\*head;

\*head=newNode;

}

void enQueue(int x)

{

queue[++rear]=x;

if(front==-1)

{

front++;

}

}

int deQueue()

{

front++;

}

int start()

{

if(front>rear || front==-1)

{

return -1;

}

return queue[front];

}

void bfs(Node \*\*v,int j)

{

int i,k;

Node \*headj;

enQueue(j);

while(j=start()!=-1)

{

k=start();

deQueue();

printf("%d ",k);

visited[k]=1;

headj=v[j];

while(headj)

{

i=headj->data;

if(visited[i]==0)

{

enQueue(i);

visited[i]=1;

}headj=headj->next;

}

}

}

int main() {

int vertices,edges,i,a,b;

printf("Enter the number of vertices and edges ");

scanf("%d%d",&vertices,&edges);

Node \*\*graph=(Node \*\*)malloc(sizeof(Node \*)\*(vertices+1));

for(i=1;i<=edges;i++)

{

scanf("%d%d",&a,&b);

add(graph+a,b);

add(graph+b,a);

}

for(i=1;i<=vertices;i++)

{

if(visited[i]==0)

{

bfs(graph,i);

}

}

return 0;

}

Input->7 9

1 2

1 3

1 4

3 5

3 6

2 5

4 6

5 7

6 7

Output->1 4 3 2 5 6 7

8. Implementation of DFS Traversal using adjacency matrix

Code->

#include<stdio.h>  
void dfs(int);  
int stack[10],top=-1,n,a[10][10],visited[10];  
void push(int);  
int pop();  
void main()  
{  
    int i,j;  
    printf("Enter the number of nodes->");  
    scanf("%d",&n);  
    for(i=1;i<=n;i++)  
    {  
        for(j=1;j<=n;j++)  
        {  
            scanf("%d",&a[i][j]);  
        }  
    }  
    for(i=1;i<=n;i++)  
    {  
        visited[i]=0;  
    }  
    dfs(1);  
}  
void dfs(int i)  
{  
   int k;  
   push(i);  
   k=pop();  
   visited[i]=1;  
   if(k!=0)  
   {  
       printf("%d ",k);  
   }  
   
   while(k!=0)  
   {for(int j=1;j<=n;j++)  
   {  
       if(a[k][j]==1 && visited[j]==0)  
       {  
           push(j);  
           visited[j]=1;  
       }  
   }  
   k=pop();  
   if(k!=0)  
   {  
       printf("%d ",k);  
   }}  
   for(int j=1;j<=n;j++)  
   {  
       if(visited[j]==0)  
       {  
           dfs(j);  
       }  
   }  
}  
void push(int k)  
{  
    stack[++top]=k;  
}  
int pop()  
{  
    int p;  
    if(top==-1)  
    {  
        return 0;  
    }  
    p=stack[top];  
    top--;  
   return p;  
}

9. Implementation of DFS Traversal using adjacency list

Code->

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node \*next;

}Node;

int visited[30],stack[20],top=-1;

void add(Node \*\*head,int j)

{

Node \*newNode=(Node \*)malloc(sizeof(Node));

newNode->data=j;

newNode->next=\*head;

\*head=newNode;

}

void push(int x)

{

stack[++top]=x;

}

int pop()

{

--top;

}

int get()

{

if(top==-1)

{

return -1;

}

return stack[top];

}

void dfs(Node \*\*v,int j)

{

int i,k;

Node \*headj;

push(j);

while(j=get()!=-1)

{

k=get();

pop();

printf("%d ",k);

visited[k]=1;

headj=v[k];

while(headj)

{

i=headj->data;

if(visited[i]==0)

{

push(i);

visited[i]=1;

}headj=headj->next;

}

}

}

int main() {

int vertices,edges,i,a,b;

printf("Enter the number of vertices and edges ");

scanf("%d%d",&vertices,&edges);

Node \*\*graph=(Node \*\*)malloc(sizeof(Node \*)\*(vertices+1));

for(i=1;i<=edges;i++)

{

scanf("%d%d",&a,&b);

add(graph+a,b);

add(graph+b,a);

}

for(i=1;i<=vertices;i++)

{

if(visited[i]==0)

{

dfs(graph,i);

}

}

return 0;

}

Input->7 9

1 2

1 3

1 4

3 5

3 6

2 5

4 6

5 7

6 7

Output-> 1 2 5 7 6 3 4

10. Implement connected components using appropriate traversal.

Code->

#include<stdio.h>

void dfs(int,int[]);

int stack[10],top=-1,n,a[10][10];

void main()

{

int i,j,count=0,p,visited[10];

printf("Enter the number of nodes->\n");

scanf("%d",&n);

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

{

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

{

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

}

}

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

{

visited[i]=0;

}

for(p=1;p<=n;p++){

if(visited[p]==0)

{

printf("The connected components are->\n");

dfs(p,visited);count++;

}}

printf("\n%d are the number of connected components of the graph ",count);

}

void dfs(int i,int visited[10])

{

int k;

visited[i]=1;

printf("%d ",i);

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

{

if(visited[k]==0 && a[i][k]==1)

{

dfs(k,visited);

}

}

}

Input->

Case 1:

7

0 0 0 0 1 0 0

0 0 0 1 0 1 0

0 0 0 0 0 0 1

0 1 0 0 0 0 0

1 0 0 0 0 0 0

0 1 0 0 0 0 0

0 0 1 0 0 0 0

Output->Case 1:

Enter the number of nodes->

The connected components are->1 5

The connected components are->2 4 6

The connected components are->3 7

3 are the number of connected components of the graph

11. Implementation of Inorder, Preorder and Post order Traversal on Binary search trees with and without using recursion.

Using recursion->

#include<stdio.h>

#include<stdlib.h>

struct bstnode{

int value;

struct bstnode \*l;

struct bstnode \*r;

};

struct bstnode \*root=NULL;

struct bstnode \*temp=NULL;

void insert();

struct bstnode\* delete(struct bstnode \*,int);

struct bstnode \*create();

struct bstnode \*min(struct bstnode \*);

void search(struct bstnode \*,struct bstnode \*);

void inorder(struct bstnode \*);

void preorder(struct bstnode \*);

void postorder(struct bstnode \*);

void main()

{

int i,m,n,y;

printf("Enter the number of nodes u want to insert>\n");

scanf("%d",&m);

printf("Enter the number of nodes u want to delete->\n");

scanf("%d",&n);

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

{

insert();

}

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

{

printf("Enter the value of the element u want to delete->");

scanf("%d",&y);

delete(root,y);

}

printf("Elements in inorder->\n");

inorder(root);

printf("\n");

printf("Elements in preorder->\n");

preorder(root);

printf("\n");

printf("Elements in postorder->\n");

postorder(root);

printf("\n");

}

void insert()

{

if(root==NULL)

{

root=create();

}

else

{

temp=create();

search(root,temp);

}

}

struct bstnode \*create()

{

int data;

printf("Enter the data to be inserted->\n");

scanf("%d",&data);

temp=(struct bstnode\*)malloc(sizeof(struct bstnode));

temp->value=data;

temp->l=NULL;

temp->r=NULL;

return temp;

}

void search(struct bstnode \*t,struct bstnode \*temp)

{

if((temp->value>t->value) && (t->r!=NULL))

{

search(t->r,temp);

}

else if((temp->value>t->value)&& (t->r==NULL))

{

t->r=temp;

}

else if((temp->value<t->value)&& (t->l!=NULL))

{

search(t->l,temp);

}

else if((temp->value<t->value)&& (t->l==NULL))

{

t->l=temp;

}

}

void inorder(struct bstnode \*t)

{

if(t==NULL)

{

printf("No elements");

return;

}

if(t->l!=NULL)

{

inorder(t->l);

}

printf("%d ",t->value);

if(t->r!=NULL)

{

inorder(t->r);

}

}

void preorder(struct bstnode \*t)

{

if(root==NULL)

{

printf("No elements");

return;

}

printf("%d ",t->value);

if(t->l!=NULL)

{

preorder(t->l);

}

if(t->r!=NULL)

{

preorder(t->r);

}

}

void postorder(struct bstnode \*t)

{

if(root==NULL)

{

printf("No elements");

return;

}

if(t->l!=NULL)

{

postorder(t->l);

}

if(t->r!=NULL)

{

postorder(t->r);

}

printf("%d ",t->value);

}

struct bstnode\* delete(struct bstnode \*root,int y)

{

int val;

struct bstnode\* p=NULL;

struct bstnode\* q=root;

while(q!=NULL && q->value!=y)

{

p=q;

if(y<q->value)

{

q=q->l;

}

else{

q=q->r;

}

}

if(q==NULL)

return;

if(q->l==NULL && q->r==NULL)

{

if(q!=root)

{

if(p->l==q)

{p->l=NULL;}

else

{

p->r=NULL;

}

}

else{

root=NULL;

}

free(q);

}

else if(q->l && q->r)

{

struct bstnode\* suc=min(q->r);

val=suc->value;

q->value=val;

delete(q->r,suc->value);

}

else{

struct bstnode \*s=(q->l)?q->l:q->r;

if(q!=root)

{

if(q==p->l)

p->l=s;

else

p->r=s;

}

else

{root=s;}

free(q);

}

return q;

}

struct bstnode \*min(struct bstnode \*q)

{

while(q->l!=NULL)

{

q=q->l;

}

return q;

}

Output->

Enter the number of nodes u want to insert>

14

Enter the number of nodes u want to delete->

3

Enter the data to be inserted->

10

Enter the data to be inserted->

6

Enter the data to be inserted->

20

Enter the data to be inserted->

4

Enter the data to be inserted->

8

Enter the data to be inserted->

15

Enter the data to be inserted->

25

Enter the data to be inserted->

3

Enter the data to be inserted->

5

Enter the data to be inserted->

7

Enter the data to be inserted->

19

Enter the data to be inserted->

22

Enter the data to be inserted->

30

Enter the data to be inserted->

2

Enter the value of the element u want to delete->2

Enter the value of the element u want to delete->15

Enter the value of the element u want to delete->6

Elements in inorder->

3 4 5 7 8 10 19 20 22 25 30

Elements in preorder->

10 7 4 3 5 8 20 19 25 22 30

Elements in postorder->

3 5 4 8 7 19 22 30 25 20 10

Without recursion->

12. Implementation of Kruskal’s Minimum cost spanning tree algorithm.

Code->

#include<stdio.h>

struct kruskal{

int src;

int dst;

int cost;

}k[50];

int findparent(int,int[]);

int main()

{

int n,m,i,j,temp,count=0,srcparent,dstparent,currentedge,parent[10],output[20],sum=0;

printf("Enter the number of vertices and edges\n");

scanf("%d%d",&n,&m);

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

{

scanf("%d%d%d",&k[i].src,&k[i].dst,&k[i].cost);

}

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

{

parent[i]=i;

}

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

{

for(j=0;j<m-i-1;j++)

{

if(k[j].cost>k[j+1].cost)

{

temp=k[j].cost;

k[j].cost=k[j+1].cost;

k[j+1].cost=temp;

temp=k[j].src;

k[j].src=k[j+1].src;

k[j+1].src=temp;

temp=k[j].dst;

k[j].dst=k[j+1].dst;

k[j+1].dst=temp;

}

}

}

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

{

printf("%d %d %d\n",k[i].src,k[i].dst,k[i].cost);

}

i=0;

while(count!=n-1)

{

currentedge=k[i].cost;

srcparent=findparent(k[i].src,parent);

dstparent=findparent(k[i].dst,parent);

if(srcparent!=dstparent)

{

output[count++]=k[i].cost;

parent[srcparent]=dstparent;

}i++;

}

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

{

printf("The path selected is %d\n",output[i]);

sum+=output[i];

}

printf("Total cost is %d ",sum);

}

int findparent(int v,int parent[10])

{

if(parent[v]==v)

{

return v;

}

return findparent(parent[v],parent);

}

Input->

7 9

1 2 26

1 6 10

2 7 14

2 4 18

2 3 16

3 4 12

4 5 22

5 6 25

5 7 24

Output->

Enter the number of vertices and edges

1 6 10

3 4 12

2 7 14

2 3 16

2 4 18

4 5 22

5 7 24

5 6 25

1 2 26

The path selected is 10

The path selected is 12

The path selected is 14

The path selected is 16

The path selected is 22

The path selected is 25

Total cost is 99

13. Implementation of Prim’s Minimum cost spanning tree algorithm.

Code->

#include<stdio.h>

# define MAX 20

#define infinity 99999

int prims();

int spanning[20][20],G[20][20],n;

int main()

{

int i,j,tcost;

printf("Enter number of vertices");

scanf("%d",&n);

printf("enter adjacency matrix\n");

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

{

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

{

scanf("%d",&G[i][j]);

}

}

tcost=prims();

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

{

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

{

printf("%d ",spanning[i][j]);

}printf("\n");

}

printf("Total cost is: %d ",tcost);

return 0;

}

int prims()

{

int cost[20][20],i,j,distance[10],visited[10],u,v,from[10],min\_cost=0,noofedges=n-1,min\_distance;

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

{

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

{

if(G[i][j]==0)

{

cost[i][j]=infinity;

}

else{

cost[i][j]=G[i][j];

spanning[i][j]=0;

}

}

}

distance[0]=0;

visited[0]=1;

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

{

distance[i]=cost[0][i];

visited[i]=0;

from[i]=0;

}

while(noofedges>0)

{

min\_distance=infinity;

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

{

if(visited[i]==0 && distance[i]<min\_distance)

{

v=i;

min\_distance=distance[i];

}

}

u=from[v];

visited[v]=1;

spanning[u][v]=distance[v];

spanning[v][u]=distance[v];

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

{

if(visited[i]==0 && cost[i][v]<distance[i])

{

distance[i]=cost[i][v];

from[i]=v;

}

}

min\_cost+=cost[u][v];

noofedges--;

}

return min\_cost;

}

Input->

6

0 3 1 6 0 0

3 0 5 0 3 0

1 5 0 5 6 4

6 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

Output->

0 3 1 0 0 0

3 0 0 0 3 0

1 0 0 0 0 4

0 0 0 0 0 2

0 3 0 0 0 0

0 0 4 2 0 0

Total cost is: 13

14. Implementation of Topological sort.

Code->

#include<stdio.h>

int a[10][10],n;

int main()

{

int i,j,indegree[20],k;

printf("Enter the number of nodes->");

scanf("%d",&n);

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

{

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

{

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

}

}

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

{

indegree[i]=0;

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

{

indegree[i]+=a[j][i];

}

}

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

{

if(indegree[i]==0)

{ printf("%d ",i);

indegree[i]=-1;}

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

{

if(a[i][k]==1)

{

indegree[k]-=1;

}

}

}

return 0;}

(Or)

#include<stdio.h>

int a[10][10],n;

void add(int);

int delete();

int front=-1,rear=-1,que[20];

int main()

{

int i,j,indegree[20],k;

printf("Enter the number of nodes->");

scanf("%d",&n);

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

{

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

{

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

}

}

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

{

indegree[i]=0;

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

{

indegree[i]+=a[j][i];

}

}

for(i=1;i<=n;i++){printf("%d ",indegree[i]);}

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

{

if(indegree[i]==0)

{

add(i);

}

}

while(front<=rear)

{

k=delete();

printf("%d ",k);

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

{

if(a[k][i]==1)

{

indegree[i]-=1;

a[k][i]=0;

if(indegree[i]==0)

{

add(i);

}}

}

}

return 0;

}

void add(int item)

{

if(front==-1)

{

que[++rear]=item;

front++;

}

else

que[++rear]=item;

}

int delete()

{

int k;

if((front>rear)||(front==-1))

return(0);

else

{

k=que[front++];

return k;

}

}

Input->

Case 1:

8

0 0 1 1 1 0 0 0

0 0 1 0 0 0 0 1

0 0 0 0 0 1 0 0

0 0 0 0 0 1 0 1

0 0 0 0 0 0 0 1

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0

Case 2:

4

0 1 1 0

0 0 0 1

0 0 0 1

0 0 0 0

Output->

Case 1:

1 2 3 4 5 6 7 8

Case 2:

1 2 3 4

15. Implementation of fractional Knapsack algorithm using Greedy approach.

Code->

#include<stdio.h>

int main() {

int w[10],p[10],i,n,j,temp;

float r,profit=0.0,sum=0.0,capacity;

float s[10];

printf("Enter the capacity");

scanf("%f",&capacity);

printf("Enter the number of items");

scanf("%d",&n);

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

{

scanf("%d%d",&w[i],&p[i]);

s[i]=(float)(p[i])/(float)(w[i]);

}

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

{

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

{

if(s[j]<s[j+1])

{

temp=s[j];

s[j]=s[j+1];

s[j+1]=temp;

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

}

}

}

i=0;

while(sum<=capacity)

{

if((float)w[i]<=capacity-sum)

{

sum+=w[i];

profit+=p[i];

}

else{

r=(capacity-sum)/w[i];

sum+=capacity-sum;

profit+=(r\*p[i]);

}i++;

}

printf(" %f ",profit);

return 0;

}

Output->

60

4

40 280

10 100

20 120

24 120

440.000000

16. Implementation of Optimal storage on single and multiple tapes using Greedy approach.

Single tape->

#include<stdio.h>  
int main()  
{  
    int n,a[10],i,temp,j;  
    float sum=0.0;  
    printf("Enter the number of tapes");  
    scanf("%d",&n);  
    printf("Enter times");  
    for(i=0;i<n;i++)  
    {  
        scanf("%d",&a[i]);  
    }  
    for(i=0;i<n;i++)  
    {  
        for(j=0;j<n-i-1;j++)  
        {  
            if(a[j]>a[j+1])  
            {  
                temp=a[j];  
                a[j]=a[j+1];  
                a[j+1]=temp;  
                 
            }  
        }  
    }  
    for(i=0;i<n;i++)  
    {  
        sum+=(n-i)\*a[i];  
    }  
    printf("%f",sum/n);  
    return 0;  
}

Input->4  
5  
7  
2  
3

Output-> 8.50000

Multiple tapes->

Code->

#include<stdio.h>

int main()

{

int n,a[30],i,temp,j,m,allocate[10][10],k,r=0,b[10][10],f=0,d[10],l;

int sum=0;

printf("Enter the number of programs\n");

scanf("%d",&n);

printf("Enter no of tapes\n");

scanf("%d",&m);

printf("Enter length of programs->\n");

for(i=0;i<30;i++)

{

a[i]=0;

}

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

{

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

}

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

{

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

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

if(n%m==0)

{

k=n/m;

}

else{

k=(n/m)+(n%m);

}

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

{

for(j=0;j<m;j++)

{

b[j][i]=a[r];

r++;

}

}

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

{

for(j=0;j<k && b[i][j]!=0;j++)

{

d[f]=b[i][j];

f++;

}

for(l=0;l<=f;l++)

{

sum+=(f-l)\*d[l];

}

f=0;

printf("Time of tapes : %d\n",sum);

sum=0;

}

return 0;

}

Input->

Case 1:

9

3

6 3 2 8 10 5 9 1 4

Case 2:

10

3

6 3 2 8 10 5 9 1 4 7

Output->

Case1:

Time of tapes : 19

Time of tapes : 25

Time of tapes : 31

Case 2:

Time of tapes : 40

Time of tapes : 24

Time of tapes : 30

17. Implementation of Optimal job scheduling with deadlines using Greedy approach.

Code->

#include<stdio.h>  
int main()  
{  
    int d[10],J[10],p[10],i,r,k,n,j,temp;  
    printf("Enter the number of jobs");  
    scanf("%d",&n);  
    for(i=1;i<=n;i++)  
    {  
        scanf("%d%d",&p[i],&d[i]);  
    }  
    for(i=1;i<=n;i++)  
    {  
        for(j=1;j<=n-i-1;j++)  
        {  
            if(p[j]<p[j+1])  
            {  
                temp=p[j];  
                p[j]=p[j+1];  
                p[j+1]=temp;  
                temp=d[j];  
                d[j]=d[j+1];  
                d[j+1]=temp;  
            }  
        }  
    }  
    d[0]=J[0]=0;  
    J[1]=1;  
    k=1;  
    for(i=2;i<=n;i++)  
   {r=k;  
    while(d[J[r]]>d[i] && d[J[r]]!=r)  
    {  
        r-=1;  
    }  
    if(d[J[r]]<=d[i] && d[i]>r)  
    {  
        for(j=k;j<r+1;j++)  
        {  
            J[j+1]=J[j];  
        }  
        J[r+1]=i;  
        k+=1;  
    }  
}  
    for(i=1;i<=k;i++)  
    {  
        printf(" %d ",J[i]);  
    }  
    return 0;  
}

input->

5  
25 2  
15 2  
10 1  
5 3  
1 3

output->

1 2 4

18. Implementation of Optimal merge patterns using Greedy approach.

Code->

#include<stdio.h>

void main()

{

int i,k,a[10],c[10],n,l,sum=0;

printf("Enter the no. of elements\t");

scanf("%d",&n);

printf("\nEnter the sorted elments\n");

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

{

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

}

i=0;k=0;

c[k]=a[i]+a[i+1];

i=2;

while(i<n)

{

k++;

if((c[k-1]+a[i])<=(a[i]+a[i+1]))

{

c[k]=c[k-1]+a[i];

}

else{

c[k]=a[i]+a[i+1];

i+=2;

while(i<n)

{

k++;

if((c[k-2]+a[i])<=a[i]+c[k-1])

{

c[k]=c[k-2]+a[i];

}

else{

c[k]=c[k-1]+a[i];

}i++;

}

}i++;

}

k++;

c[k]=c[k-1]+c[k-2];

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

{

printf("%d ",c[l]);

sum+=c[l];

}

printf("\nThe path length is %d",sum);

}

Input->

10

3 5 11 12 28 32 35 53 84 91

Output->

8 19 31 59 67 112 151 203 354

The path length is 1004

19. Implementation of Huffman encoding algorithm using Greedy approach.

Code->

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

**typedef** **struct** node

{

**char** ch;

**int** freq;

**struct** node \*left;

**struct** node \*right;

}node;

node \* heap[100];

**int** heapSize=0;

**void** Insert(node \* element)

{

heapSize++;

heap[heapSize] = element;

**int** now = heapSize;

**while**(heap[now/2] -> freq > element -> freq)

{

heap[now] = heap[now/2];

now /= 2;

}

heap[now] = element;

}

node \* DeleteMin()

{

node \* minElement,\*lastElement;

**int** child,now;

minElement = heap[1];

lastElement = heap[heapSize--];

**for**(now = 1; now\*2 <= heapSize ;now = child)

{

child = now\*2;

**if**(child != heapSize && heap[child+1]->freq < heap[child] -> freq )

{

child++;

}

**if**(lastElement -> freq > heap[child] -> freq)

{

heap[now] = heap[child];

}

**else**

{

**break**;

}

}

heap[now] = lastElement;

**return** minElement;

}

**void** print(node \*temp,**char** \*code)

{

**if**(temp->left==NULL && temp->right==NULL)

{

printf("char %c code %s\n",temp->ch,code);

**return**;

}

**int** length = strlen(code);

**char** leftcode[10],rightcode[10];

strcpy(leftcode,code);

strcpy(rightcode,code);

leftcode[length] = '0';

leftcode[length+1] = '\0';

rightcode[length] = '1';

rightcode[length+1] = '\0';

print(temp->left,leftcode);

print(temp->right,rightcode);

}

**int** main()

{

heap[0] = (node \*)malloc(**sizeof**(node));

heap[0]->freq = 0;

**int** n ;

printf("Enter the no of characters: ");

scanf("%d",&n);

printf("Enter the characters and their frequencies: ");

**char** ch;

**int** freq,i;

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

{

scanf(" %c",&ch);

scanf("%d",&freq);

node \* temp = (node \*) malloc(**sizeof**(node));

temp -> ch = ch;

temp -> freq = freq;

temp -> left = temp -> right = NULL;

Insert(temp);

}

**if**(n==1)

{

printf("char %c code 0\n",ch);

**return** 0;

}

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

{

node \* left = DeleteMin();

node \* right = DeleteMin();

node \* temp = (node \*) malloc(**sizeof**(node));

temp -> ch = 0;

temp -> left = left;

temp -> right = right;

temp -> freq = left->freq + right -> freq;

Insert(temp);

}

node \*tree = DeleteMin();

**char** code[10];

code[0] = '\0';

print(tree,code);

}

20. Implementation of Dijkstra’s Single source shortest path algorithm using Greedy approach.

Code->

#include<stdio.h>  
//#include<conio.h>  
#define INFINITY 9999  
#define MAX 10  
   
void dijkstra(int G[MAX][MAX],int n,int startnode);   
int main()  
{  
int G[MAX][MAX],i,j,n,u;  
printf("Enter no. of vertices:");  
scanf("%d",&n);  
printf("\nEnter the adjacency matrix:\n");  
  
for(i=0;i<n;i++)  
for(j=0;j<n;j++)  
scanf("%d",&G[i][j]);  
  
printf("\nEnter the starting node:");  
scanf("%d",&u);  
dijkstra(G,n,u);  
  
return 0;  
}  
   
void dijkstra(int G[MAX][MAX],int n,int startnode)  
{  
   
int cost[MAX][MAX],distance[MAX],pred[MAX];  
int visited[MAX],count,mindistance,nextnode,i,j;  
  
//pred[] stores the predecessor of each node  
//count gives the number of nodes seen so far  
//create the cost matrix  
for(i=0;i<n;i++)  
for(j=0;j<n;j++)  
if(G[i][j]==0)  
cost[i][j]=INFINITY;  
else  
cost[i][j]=G[i][j];  
  
//initialize pred[],distance[] and visited[]  
for(i=0;i<n;i++)  
{  
distance[i]=cost[startnode][i];  
pred[i]=startnode;  
visited[i]=0;  
}  
  
distance[startnode]=0;  
visited[startnode]=1;  
count=1;  
  
while(count<n-1)  
{  
mindistance=INFINITY;  
  
//nextnode gives the node at minimum distance  
for(i=0;i<n;i++)  
if(distance[i]<mindistance&&!visited[i])  
{  
mindistance=distance[i];  
nextnode=i;  
}  
  
//check if a better path exists through nextnode  
visited[nextnode]=1;  
for(i=0;i<n;i++)  
if(!visited[i])  
if(mindistance+cost[nextnode][i]<distance[i])  
{  
distance[i]=mindistance+cost[nextnode][i];  
pred[i]=nextnode;  
}  
count++;  
}  
   
//print the path and distance of each node  
for(i=0;i<n;i++)  
if(i!=startnode)  
{  
printf("\nDistance of node%d=%d",i,distance[i]);  
printf("\nPath=%d",i);  
  
j=i;  
do  
{  
j=pred[j];  
printf("<-%d",j);  
}while(j!=startnode);  
}  
}

output->

6  
0 3 1 6 0 0  
3 0 5 0 3 0  
1 5 0 5 6 4  
6 0 5 0 0 2  
0 3 6 0 0 6  
0 0 4 2 6 0  
1

Enter no. of vertices:  
Enter the adjacency matrix:  
  
Enter the starting node:  
Distance of node0=3  
Path=0<-1  
Distance of node2=4  
Path=2<-0<-1  
Distance of node3=9  
Path=3<-0<-1  
Distance of node4=3  
Path=4<-1  
Distance of node5=8  
Path=5<-2<-0<-1

DYNAMIC PROGRAMMING

1. 0-1 Knapsack problem

Code->REcursion

#include<stdio.h>

int MAX(int a,int b)

{

return a>b?a:b;

}

int Knapsack(int c,int wt[],int v[],int n)

{

if(n==0||c==0)

return 0;

if(wt[n-1]>c)

return Knapsack(c,wt,v,n-1);

else return MAX(Knapsack(c,wt,v,n-1),v[n-1]+Knapsack(c-wt[n-1],wt,v,n-1));

}

int main()

{

int i,n,wt[30],v[30],c,r;

printf("Enter number of items:->");

scanf("%d",&n);

printf("Enter capacity");

scanf("%d",&c);

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

{

scanf("%d%d",&wt[i],&v[i]);

}

r=Knapsack(c,wt,v,n);

printf("%d",r);

}

Input->

4

60

40 280

10 100

20 120

24 120

Output->

400

Using DP->

int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int n)

{

   int i, w;

   int K[n+1][W+1];

   // Build table K[][] in bottom up manner

   for (i = 0; i <= n; i++)

   {

       for (w = 0; w <= W; w++)

       {

           if (i==0 || w==0)

               K[i][w] = 0;

           else if (wt[i-1] <= w)

                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);

           else

                 K[i][w] = K[i-1][w];

       }

   }

   return K[n][W];

}

int main()

{

    int val[] = {60, 100, 120};

    int wt[] = {10, 20, 30};

    int  W = 50;

    int n = sizeof(val)/sizeof(val[0]);

    printf("%d", knapSack(W, wt, val, n));

    return 0;

}

2. Longest Common subsequence

Code->

#include<bits/stdc++.h>

int max(int a, int b);

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs( char \*X, char \*Y, int m, int n )

{

   int L[m+1][n+1];

   int i, j;

   /\* Following steps build L[m+1][n+1] in bottom up fashion. Note

      that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] \*/

   for (i=0; i<=m; i++)

   {

     for (j=0; j<=n; j++)

     {

       if (i == 0 || j == 0)

         L[i][j] = 0;

       else if (X[i-1] == Y[j-1])

         L[i][j] = L[i-1][j-1] + 1;

       else

         L[i][j] = max(L[i-1][j], L[i][j-1]);

     }

   }

   /\* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] \*/

   return L[m][n];

}

/\* Utility function to get max of 2 integers \*/

int max(int a, int b)

{

    return (a > b)? a : b;

}

/\* Driver program to test above function \*/

int main()

{

  char X[] = "AGGTAB";

  char Y[] = "GXTXAYB";

  int m = strlen(X);

  int n = strlen(Y);

  printf("Length of LCS is %d", lcs( X, Y, m, n ) );

  return 0;

}

Output->

LCS is 4

3. Travelling salesman problem

Code->

#include<stdio.h>

int min(int a,int b)

{

return a<b?a:b;

}

int n=4;

int visited\_all=(1<<4)-1;

int dist[10][10]={

{0,20,42,25},{20,0,30,34},{42,30,0,10},{25,34,10,0}

};

int dp[16][4];

int tsp(int mask,int pos)

{

if(mask==visited\_all){

return dist[pos][0];

}

if(dp[mask][pos]!=-1){

return dp[mask][pos];

}

int ans=9999999;

for(int city=0;city<n;city++){

if((mask&(1<<city))==0){ //if 2 brackets are not kept then only city part is compared and not the whole thing

int newAns=dist[pos][city]+tsp(mask|(1<<city),city);

ans=min(ans,newAns);

}

}

return dp[mask][pos]=ans;

}

int main() {

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

{

for(int j=0;j<n;j++)

{

dp[i][j]=-1;

}

}

int r=tsp(1,0);

printf("%d\n",r);

}

Output->85

4. Optimal Binary search tree

5. All pairs shortest paths

Code->

#include<stdio.h>

#define MIN 9999

int min(int a,int b)

{

if(a<b)

{

return a;

}

return b;

}

int cost[20][20];

int main() {

int n,i,j,a[20][20],k;

printf("Enter the number of nodes\n");

scanf("%d",&n);

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

{

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

{

scanf("%d",&cost[i][j]);

}

}

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

{

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

{

a[i][j]=cost[i][j];

}

}

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

{

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

{

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

{

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

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

{

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

{

printf("%d ",a[i][j]);

}

printf("\n");

}

return 0;

}

Input->

4

0 3 9999 7

8 0 2 9999

5 9999 0 1

2 9999 9999 0

Output->

0 3 5 6

5 0 2 3

3 6 0 1

2 5 7 0

6. Multistage graph

Code->

#include<stdio.h>

#define N 8

#define INF 9999

int min(int a,int b)

{

return a<b?a:b;

}

int shortestDist(int graph[N][N]) {

int dist[N];

**dist[N-1] = 0;**

for (int i = N-2 ; i >= 0 ; i--)

{

dist[i] = INF;

for (int j = i ; j < N ; j++)

{

if (graph[i][j] == INF)

continue;

dist[i] = min(dist[i], graph[i][j] +

dist[j]);

}

}

return dist[0];

}

int main()

{

int graph[N][N] =

{{INF, 1, 2, 5, INF, INF, INF, INF},

{INF, INF, INF, INF, 4, 11, INF, INF},

{INF, INF, INF, INF, 9, 5, 16, INF},

{INF, INF, INF, INF, INF, INF, 2, INF},

{INF, INF, INF, INF, INF, INF, INF, 18},

{INF, INF, INF, INF, INF, INF, INF, 13},

{INF, INF, INF, INF, INF, INF, INF, 2}};

printf("%d ",shortestDist(graph));

return 0;

}

Output->

9

To print the path->

#include<stdio.h>

int main() {

int i,j,n,a[20][20],k,cost[20],d[20],p[20],s,min=999;

printf("Enter number of nodes and stages\n");

scanf("%d%d",&n,&s);

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

{

for(j=0;j<n+1;j++)

{

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

}

}

cost[n]=0;

for(i=n-1;i>=1;i--)

{ min=999;

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

{

if(a[i][k]!=0 && a[i][k]+cost[k]<min)

{

min=cost[k]+a[i][k];

d[i]=k;

}

}cost[i]=min;

}

p[1]=1;p[s]=12;

for(i=2;i<s;i++)

{

p[i]=d[p[i-1]];

}

printf("The path is->");

for(i=1;i<=s;i++)

{

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

}

return 0;

}

Input->12 5

0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 9 7 3 2 0 0 0 0 0 0 0

0 0 0 0 0 0 4 2 0 0 0 0 0

0 0 0 0 0 0 2 7 0 0 0 0 0

0 0 0 0 0 0 0 0 11 0 0 0 0

0 0 0 0 0 0 0 0 8 0 0 0 0

0 0 0 0 0 0 0 0 0 6 5 0 0

0 0 0 0 0 0 0 0 0 4 3 0 0

0 0 0 0 0 0 0 0 0 0 5 6 0

0 0 0 0 0 0 0 0 0 0 0 0 4

0 0 0 0 0 0 0 0 0 0 0 0 2

0 0 0 0 0 0 0 0 0 0 0 0 5

0 0 0 0 0 0 0 0 0 0 0 0 0

Output->The path is->1->2->7->10->12->

7. Matrix chain multiplication

Code->

#include<stdio.h>

int main() {

int m[5][5]={0};

int s[5][5]={0};

int p[5]={5,4,6,2,7};

int i,j,k,d,min,n=5,q;

for(d=1;d<n-1;d++)

{

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

{

j=i+d;

min=9999;

for(k=i;k<=j-1;k++)

{

q=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(q<min)

{

min=q;

s[i][j]=k;

}

}m[i][j]=min;

}

}

printf("%d ",m[1][n-1]);

}s

Output-> 158