Pract-1 Removal of Recursion

1. **Write a program to implement removal of recursion for**

   a. **Finding maximum from array**

```
#include<iostream>
#include<conio.h>
using namespace std;
class MaxEle
{
int S[50],addr,top,A[50],n,i;
public:
MaxEle()
{
i=1;
}
void Get();
void Max();
};
void MaxEle :: Get()
{
cout << "\n Enter the numbers of elements :";
cin >> n;
cout << "\n Enter the elements : ";
for(int m=1;m<=n;m++)
{
cin >> A[m];
}
```

```
}
void MaxEle :: Max()
{
int j,k;
top=0;
L1:if(i<n)
{
S[++top]=i;
S[++top]=2;
i++;
goto L1;
L2:j=S[top--];
if(A[i] > A[j])
{
k=i;
}
else
{
k=j;
}
}
else
{
k=n;
}
if(top==0)
cout << "\n Maximum element is : " << A[k] ;
else
{
```

```
addr=S[top--];
i=S[top--];
S[++top]=k;
if(addr==2)
goto L2;
}
}
int main()
{
MaxEle M;
int val;
M.Get();
M.Max();
getch();
}
```

b. Binomial Coefficient B(n,m)= B(n-1, m-1)+B(n-1,m), B(n,n)=B(n,0)=1

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
using namespace std;
class Bino
{
int k,S[30],add,top;
public:
int Binomial(int,int);
};
int Bino :: Binomial(int i,int j)
{
```

```
top=-1;
k=0;
L1: if((i!=j) && (j!=0))
{
S[++top]=i-1;
S[++top]=j-1;
S[++top]=2;
S[++top]=i-1;
S[++top]=j;
S[++top]=2;
}
else
k++;
if(top==-1)
return(k);
else
{
add=S[top--];
j=S[top--];
i=S[top--];
if(add==2)
{
goto L1;
}
}
return(0);
}
int main()
{
Bino B;
int a,b,val;
```

```
cout << "\n\n Enter two values:";
cin >> a >> b;
if(a>b)
{
val=B.Binomial(a,b);
cout << "\n\n Binomial coefficient of "<<a<<" & "<<b<<" is:"<<val;
}
else
{
cout << "\n\n Invalid Input";
}
getch();
}
```

c. Searching element from array

```
#include<iostream>
#include<conio.h>
using namespace std;
class SearchEle
{
int S[50],addr,top,A[50],n,i,no,j,k;
public:
SearchEle()
{
i=1;
}
void Get();
void Search();
};
void SearchEle :: Get()
{
```

```
cout << "\n Enter the numbers of elements :";
cin >> n;
cout << "\n Enter the elements : ";
for(int m=1;m<=n;m++)
{
cin >> A[m];
}
cout << "\n Enter the element to be searched:";
cin >> no;
}
void SearchEle :: Search()
{
int j,k;
top=0;
L1:if(i<=n)
{
S[++top]=i;
S[++top]=2;
i++;
goto L1;
L2:j=S[top--];
if(A[j]==no)
{
k=j;
cout << "\n Element is found at position : " << k ;
return;
}
else
{
k=0;
}
```

```
}
if(top==0 && k==0)
{
cout << "\n Element is not found.";
}
else
{
addr=S[top--];
if(addr==2)
goto L2;
}
}
int main()
{
SearchEle S;
int val;
S.Get();
S.Search();
getch();
}
```

Pract-2 Elementary Data Structure–Tree

1) **Write a program for creating Max/Min. heap using INSERT.**
   A) **Insert Max Heap**

```
#include<iostream>
#include<conio.h>
using namespace std;
class Insertmaxheap
{
int a[10];
int n;
public:
void Insert(int);
void get();
void show();
};
void Insertmaxheap::get()
{
cout<<"\nEnter the size of Heap : ";
cin>>n;
cout<<"\nEnter the elements :";
for(int i=1;i<=n;i++)
{
cin>>a[i];
}
cout<<"\nBefore :\n";
show();
for( int i=1;i<=n;i++)
```

```
{
Insert(i);
}
}
void Insertmaxheap::Insert(int n)
{
int i=n;
int item=a[n];
while(i>1 && a[i/2]<item)
{
a[i]=a[i/2];
i=i/2;
}
a[i]=item;
}
void Insertmaxheap::show()
{
for(int i=1;i<=n;i++)
cout<<a[i]<<"\t";
}
int main()
{
Insertmaxheap a;
a.get();
cout<<"\nAfter Insert :\n";
a.show();
getch();
}
```

B) Insert Min Heap

```
#include<iostream>
#include<conio.h>
using namespace std;
class Insertminheap
{
int a[10];
int n;
public:
void Insert(int);
void get();
void show();
};
void Insertminheap::get()
{
cout<<"\nEnter the size of heap : ";
cin>>n;
cout<<"\nEnter the elements :\n";
for(int i=1;i<=n;i++)
{
cin>>a[i];
}
cout<<"\nBefore :\n";
show();
for( int i=1;i<=n;i++)
{
Insert(i);
}
```

```
}
void Insertminheap::Insert(int n)
{
int i=n;
int item=a[n];
while(i>1 && a[i/2]>item)
{
a[i]=a[i/2];
i=i/2;
}
a[i]=item;
}
void Insertminheap::show()
{
for(int i=1;i<=n;i++)
cout<<a[i]<<"\t";
}
int main()
{
Insertminheap a;
a.get();
cout<<"\nAfter Insert :\n";
a.show();
getch();
}
```

2) Write a program for creating Max/Min. heap using ADJUST/HEAPIFY.

A) Adjust Max Heap

```
#include<iostream>
#include<conio.h>
using namespace std;
class Adjustmaxheap
{
int a[10];
int n;
public:
void Adjust(int,int);
void Heapify(int);
void get();
void show();
};
void Adjustmaxheap::get()
{
cout<<"\n Enter the size of heap :";
cin>>n;
cout<<"\n Enter the elements :";
for(int i=1;i<=n;i++)
{
cin>>a[i];
}
Heapify(n);
}
void Adjustmaxheap::Adjust(int i,int n)
{
int j=2*i;
```

```
int item=a[i];
while(j<=n)
{
if((j<n)&&(a[j]<a[j+1]))
{
j++;
}
if(item>=a[j])
break;
a[j/2]=a[j];
j=2*j;
}
a[j/2]=item;
}
void Adjustmaxheap::Heapify(int n)
{
for(int i=n/2;i>=1;i--)
Adjust(i,n);
}
void Adjustmaxheap::show()
{
cout<<"\nMax heap is :\n";
for(int i=1;i<=n;i++)
cout<<a[i]<<"\t";
}
int main()
{
Adjustmaxheap a;
a.get();
```

```
a.show();
getch();
}
```

B) Adjust Min Heap

```
#include<iostream>
#include<conio.h>
using namespace std;
class Adjustminheap
{
int a[10];
int n;
public:
void Adjust(int,int);
void Heapify(int);
void get();
void show();
};
void Adjustminheap::get()
{
cout<<"\n Enter the size of heap :";
cin>>n;
cout<<"\n Enter the elements :";
for(int i=1;i<=n;i++)
{
cin>>a[i];
}
Heapify(n);
```

```
}
void Adjustminheap::Adjust(int i,int n)
{
int j=2*i;
int item=a[i];
while(j<=n)
{
if((j<n)&&(a[j]>a[j+1]))
{
j++;
}
if(item>=a[j])
break;
a[j/2]=a[j];
j=2*j;
}
a[j/2]=item;
}
void Adjustminheap::Heapify(int n)
{
for(int i=n/2;i>=1;i--)
Adjust(i,n);
}
void Adjustminheap::show()
{
cout<<"\nMax heap is :\n";
for(int i=1;i<=n;i++)
cout<<a[i]<<"\t";
}
```

```
int main()
{
Adjustminheap a;
a.get();
a.show();
getch();
}
```

3) **Write a program for sorting given array in ascending/descending order with n=1000,2000,3000.Find exact time of execution using Heap Sort**

```
#include<iostream>
#include<conio.h>
#define Max 100
#include<time.h>
#include<stdlib.h>
using namespace std;
class Heap
{
int Sort[Max];
int N;
public:
void GetData();
void Heap_Sort(int [],int);
void Adjust(int [],int,int);
void Heapify(int [],int);
void PutData();
};
void Heap::GetData()
{
cout<<"\nENTER THE TOTAL ELEMENTS :";
cin>>N;
cout<<"\nENTER THE ELEMENTS :" << endl;
rand();
for(int i=1;i<=N;i++)
{
Sort[i]=rand()%100;
```

```
cout << Sort[i] << endl;
}
Heap_Sort(Sort,N);
};
void Heap::Heap_Sort(int Sort[],int N)
{
Heapify(Sort,N);
for(int i=N;i>=2;i--)
{
int Temp=Sort[i];
Sort[i]=Sort[1];
Sort[1]=Temp;
Adjust(Sort,1,i-1);
}
};
void Heap::Heapify(int Sort[],int N)
{
for(int i=(N/2);i>=1;i--)
{
Adjust(Sort,i,N);
}
};
void Heap::Adjust(int Sort[],int i,int N)
{
int j=2*i;
int Item=Sort[i];
while(j<=N)
{
if(j<N && Sort[j]<Sort[j+1])
```

```
{
j=j+1;
}
if(Item>=Sort[j])
break;
else
{
Sort[j/2]=Sort[j];
j=j*2;
}
}
Sort[j/2]=Item;
};
void Heap::PutData()
{
cout<<"\nAFTER SORTING ELEMENTS ARE :\n";
for(int i=1;i<=N;i++)
{
cout << Sort[i] << endl;
}
};
int main()
{
int start, end;
Heap B;
start = clock();
B.GetData();
B.PutData();
end = clock();
```

```
cout<<"\n The execution time is : " << (end - start) / CLK_TCK;
return 0;
}
```

4) Write a program to implement Weighted UNION and Collapsing FIND operations

```
#include<iostream>
#include<conio.h>
using namespace std;
class Set
{
int Root1,Root2,i,n,A[10];
public:
void Show();
void Get();
void Union();
void Find(int);
};
void Set :: Union()
{
int r1,r2;
r1=A[Root1];
r2=A[Root2];
if(A[Root1] < A[Root2])
{
A[Root2] = Root1;
r2=r1+r2;
A[Root1]=r2;
}
else
{
A[Root1]=Root2;
r1=r1+r2;
```

```
A[Root2]=r1;
}
}
void Set :: Find(int x)
{
int m=x;
while(A[m]>0) m=A[m];
cout << "\n Root is :" << m;
}
void Set :: Get()
{
for(int z=0;z<10;z++)
A[z]=0;
cout << "\n Enter the first set size : ";
cin >> n;
cout << "\n Enter the elements: ";
cin >> Root1;
A[Root1]=-n;
int j=Root1;
for(int k=1;k<n;k++)
{
cin >> i; A[i]=j;
}
cout << "\n Enter the second set size : ";
cin >> n;
cout << "\n Enter the elements: ";
cin >> Root2;
A[Root2]=-n;
j=Root2;
```

```
for(int k=1;k<n;k++)
{
cin >> i;
A[i]=j;
}
}
void Set :: Show()
{
cout << "\n";
for(int i=1;i<10;i++)
cout << A[i] << "\t";
}
int main()
{
Set s;
s.Get();
int t;
cout << "\n Enter the value to find :";
cin >> t;
s.Find(t);
cout << "\n Before Union : \n";
s.Show();
s.Union();
cout << "\n After Union : \n";
s.Show();
getch();
}
```

Pract-3 Divide and Conquer

1) **Write a program for searching element form given array using binary search forn=1000,2000,3000. Find exact time of execution.**

```
#include<iostream>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
using namespace std;
class BSearch
{
int A[100],Size;
public:
int Get();
void Sort();
int Search(int,int,int);
void Show(int);
};
int BSearch :: Get()
{
cout << "\n Enter the Size of List : ";
cin >> Size;
cout << "\n The elements of List are :\n";
for(int i=1;i<=Size;i++)
{
cin>>A[i];
```

```
}
Sort();
cout << "\n After sorting : \n";
for(int i=1;i<=Size;i++)
{
cout << A[i] << endl;
}
return 0;
}
void BSearch :: Sort()
{
for(int i=1;i<=Size;i++)
{
for(int j=1;j<=Size;j++)
{
if(A[i]<A[j])
{
int temp=A[i];
A[i]=A[j];
A[j]=temp;
}
}
}
}
int BSearch :: Search(int i,int j,int x)
{
int Mid;
if(j==i)
{
```

```
if(x==A[i])
return i;
else
return 0;
}
else
Mid = (i+j)/2;
if(x==A[Mid])
return Mid;
else if(x<A[Mid])
return Search(i,Mid-1,x);
else
return Search(Mid+1,j,x);
}
void BSearch :: Show(int x)
{
int t=Search(1,Size,x);
if(t==0)
cout << "\n Element is Not Found.";
else
cout << "\n Element is found at location: "<< t;
}
int main()
{
int start,end;
BSearch b;
int No;
start=clock();
b.Get();
```

```
cout << "\n Enter element to search : ";
cin >> No;
b.Show(No);
end=clock();
cout << "\n The execution time is : " << (end - start) / CLK_TCK;
}
```

2) Write a program to find minimum and maximum from a given array using MAXMIN

```
#include<iostream>
#include<conio.h>
#include<math.h>
using namespace std;
class MaxMin
{
public:
int a[100],Size;
int Max,Min,i,num;
public:
void Get();
void Maxmin(int,int);
void Show();
};
void MaxMin :: Get()
{
cout << "\n Enter size of List : ";
cin >> Size;
cout << "\n Enter the elements of list : ";
for(int i=1;i<=Size;i++)
{
cin >> a[i];
}
Max = a[1];
Min = a[1];
Maxmin(1,Size);
```

```
}
void MaxMin :: Maxmin(int i,int j)
{
int max1,min1,mid;
if(i==j)
{
Max=a[i];
Min=a[i];
}
else
{
if(i==j-1)
{
if(a[i] < a[j])
{
Max=a[j];
Min=a[i];
}
else
{
Max=a[i];
Min=a[j];
}
}
else
{
mid = ((i+j)/2);
Maxmin(i,mid);
max1=Max;
```

```
min1=Min;
Maxmin(mid+1,j);
if(Max<max1)
Max=max1;
if(Min>min1)
Min=min1;
}
}
}
void MaxMin :: Show()
{
cout << "\n Maximum Element : " << Max;
cout << "\n Minimum Element : " << Min;
}
int main()
{
MaxMin m;
m.Get();
m.Show();
}
```

3) Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using Merge Sort and Quick Sort.

A) Merge Sort

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
using namespace std;
class number
{
int a[50],n;
public:
void getdata();
void mergesort(int low,int high);
void merge(int low,int mid,int high);
};
void number :: getdata()
{
int i;
cout<<"\n\n NUMBER OF ELEMENTS : ";
cin>>n;
cout << "\n Enter the elements : ";
for(i=1;i<=n;i++)
{
cin >> a[i];
}
cout<<"\n\nYOUR ARRAY IS:\n";
```

```
for (i=1; i<=n; i++)
cout<<a[i]<<"\t";
mergesort(1,n);
cout<<"\n\nTHE ARRAY AFTER SORTING :\n";
for (i=1; i<=n; i++)
cout<<a[i]<<"\t";
}
void number :: mergesort(int low,int high)
{
int mid;
if (low < high)
{
mid = floor((low + high) / 2);
mergesort(low,mid);
mergesort(mid+1,high);
merge(low,mid,high);
}
}
void number :: merge(int low,int mid,int high)
{
int h,i,j,k,b[5000];
h = low;
i = low;
j = mid+1;
while ((h <= mid) && (j <=high))
{
if(a[h] <= a[j])
{
b[i] = a[h];
```

```
h=h+1;
}
else
{
b[i] = a[j];
j=j+1;
}
i=i+1;
}
if (h > mid)
{
for (k=j; k<=high; k++)
{
b[i] = a[k];
i=i+1;
}
}
else
{
for (k=h; k<=mid; k++)
{
b[i] = a[k];
i=i+1;
}
}
for (k=low; k<=high; k++)
a[k] = b[k];
}
int main()
```

```
{
number a;
a.getdata();
}
```

B) Quick Sort

```
#include<iostream>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
using namespace std;
class Quick
{
public:
int A[5000],n;
void getdata(void);
void quicksort(int p,int q);
int Partition(int m,int p);
void swap(int &a,int &b);
void putdata(void);
};
void Quick::quicksort(int p,int q)
{
if( p < q)
{
int j=q+1;
j=Partition(p,j);
```

```
quicksort(p,j-1);
quicksort(j+1,q);
}
}
int Quick::Partition(int m,int p)
{
int i;
int v=A[m];
i=m;
do
{
do
{
i++;
}
while(A[i] <= v);
do
{
p--;
}
while(A[p] > v);
if(i<p)
{
swap(A[i],A[p]);
}
else
break;
}
while(1);
```

```
A[m]=A[p];
A[p]=v;
return(p);
}
void Quick::swap(int &a,int &b)
{
int temp=a;
a=b;
b=temp;
}
void Quick :: getdata(void)
{
cout << "\n\n\t Enter the limit of the array : ";
cin >> n;
cout << "\n\t Enter the elements of the array : ";
for(int p = 1;p <= n;p++)
{
cin>>A[p];
}
}
void Quick :: putdata(void)
{
cout << "\n Elements after sorting are :\n ";
for(int k =1 ;k<=n;k++)
cout<<"\t"<<A[k];
}
int main(void)
{
Quick Q;
```

```
Q.getdata();
Q.quicksort(1,Q.n);
Q.putdata();
}
```

4) Write a program for matrix multiplication using Strassen's Matrix Multiplication.

```
#include<iostream>
#include<conio.h>
using namespace std;
class Matrix
{
int A[2][2],B[2][2],Result[2][2];
public:
void Get();
void Mult();
void Put();
};
void Matrix :: Get()
{
cout <<"\n Enter the first 2X2 matrix :\n";
for(int i=1;i<=2;i++)
{
for(int j=1;j<=2;j++)
{
cin >> A[i][j];
}
}
cout << "\n Enter the second 2X2 matrix :\n";
for(int i=1;i<=2;i++)
{
for(int j=1;j<=2;j++)
{
cin >> B[i][j];
```

```
}
}
}
void Matrix :: Mult()
{
int p,q,r,s,t,u,v;
p=(A[1][1]+A[2][2])*(B[1][1]+B[2][2]);
q=(A[2][1]+A[2][2])*B[1][1];
r=A[1][1]*(B[1][2]-B[2][2]);
s=A[2][2]*(B[2][1]-B[1][1]);
t=(A[1][1]+A[1][2])*B[2][2];
u=(A[2][1]-A[1][1])*(B[1][1]+B[1][2]);
v=(A[1][2]-A[2][2])*(B[2][1]+B[2][2]);
Result[1][1] = p+s-t+v;
Result[1][2] = r+t;
Result[2][1] = q+s;
Result[2][2] = p+r-q+u;
}
void Matrix :: Put()
{
cout << "\n Result is : \n";
for(int i=1;i<=2;i++)
{
for(int j=1;j<=2;j++)
{
cout << "\t" << Result[i][j];
}
cout << "\n";
}
```

```
}
int main()
{
Matrix m;
m.Get();
m.Mult();
m.Put();
getch();
}
```

Pract-4 Greedy Algorithms

1) **Write a program to find solution of Fractional Knapsack instance.**

```
#include<iostream>
#include<conio.h>
using namespace std;
class Data
{
public:
float p,w,x,Ratio;
char Name;
};
class Knapsack
{
public:
Data d[10];
int i,m,n;
void Show();
void Get();
Knapsack();
};
void Knapsack :: Get()
{
cout <<"\n Size of Kanpsack : ";
cin >> m;
cout << "\n Enter the size : ";
cin >> n;
```

```
for(int i=1;i<=n;i++)
{
cout << "\n Enter the weight : ";
cin >> d[i].w;
cout << "\n Enter the profit : ";
cin >> d[i].p;
cout << "\n Enter the Name : ";
cin >>d[i].Name;
d[i].Ratio=d[i].p/d[i].w;
}
}
Knapsack :: Knapsack()
{
Get();
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
if(d[i].Ratio > d[j].Ratio)
{
Data t = d[i];
d[i]=d[j];
d[j]=t;
}
}
}
for(int i=1;i<=n;i++)
d[i].x=0.0;
int u=m;
```

```
for(int i=1;i<=n;i++)
{
if(d[i].w > u)
break;
d[i].x=1.0;
u=u-d[i].w;
}
if(i<n)
{
d[i].x=u/d[i].w;
}
Show();
}
void Knapsack :: Show()
{
cout << "\nName Weight Profit Ratio x\n";
for(int i=1;i<=n;i++)
cout <<d[i].Name<<"\t"<<d[i].w<<"\t"<<d[i].p<<"\t"<<d[i].Ratio<<" "<<d[i].x<<"\n";
float pf=0.0;
for(int i=1;i<=n;i++)
{
pf=pf+(d[i].p*d[i].x);
}
cout << "\n Total Profit : " << pf << endl;
}
int main()
{
Knapsack k;
}
```

2) Write a program to find Minimum Spanning Tree using Prim's algorithm.

```
#include<iostream>
#include<conio.h>
using namespace std;
class Prims
{
int n,t[10][2],Cost[10][10],No,Near[10],k,l,j;
public:
void Get();
int prims();
void Show();
};
void Prims :: Get()
{
cout <<"\n Enter the size of matrix : ";
cin >> n;
cout << "\n Enter the cost matrix : \n";
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
cin >> Cost[i][j];
}
}
}
int Prims :: prims()
{
```

```
int Mincost=999;
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
if(Mincost > Cost[i][j])
{
Mincost = Cost[i][j];
k=i;
l=j;
}
}
}
t[1][0]=k;
t[1][1]=l;
for(int i=1;i<=n;i++)
{
if((Cost[i][l]) < (Cost[i][k]))
{
Near[i]=l;
}
else
{
Near[i]=k;
}
}
Near[k]=Near[l]=0;
for(int i=2;i<=n-1;i++)
{
```

```
int min=999;
for(int s=1;s<=n;s++)
{
if(Near[s]!=0 && min > Cost[s][Near[s]])
{
min=Cost[s][Near[s]];
j=s;
}
}
t[i][0]=j;
t[i][1]=Near[j];
Mincost=Mincost+Cost[j][Near[j]];
Near[j]=0;
for(int k=1;k<=n;k++)
{
if((Near[k]!=0) && (Cost[k][Near[k]]) > (Cost[k][j]))
Near[k]=j;
}
}
return Mincost;
}
void Prims :: Show()
{
int MCost=prims();
cout << "\n The minimum spanning tree is \n";
cout << "\n Cost" << endl;
for(int i=1;i<n;i++)
{
int u=t[i][0];
```

```
int v=t[i][1];
cout <<"\n"<<u<<"\t"<<v<<"\t"<<Cost[u][v]<<endl;
}
cout << "\n Minimum cost is : " << MCost << endl;
}
int main()
{
Prims p;
p.Get();
p.Show();
}
```

3) Write a program to find Minimum Spanning tree using Kruskal's algorithm

```
#include<iostream>
#include<conio.h>
using namespace std;
struct edge
{
int u;
int v;
int cost;
};
class Kruskals
{
edge k[10];
int n,e,p[20],t[10][3];
public:
void Get();
void Heapify();
void Adjust(int,int);
int Kruskal();
int Find(int);
void Display();
};
void Kruskals :: Get()
{
cout <<"\n How many vertices?: : ";
cin >> n;
cout << "\n Enter the edges : ";
```

```
cin >> e;
for(int i=1;i<=e;i++)
{
cout << "\n Enter the u vertex : ";
cin >> k[i].u;
cout << "\n Enter the v vertex :";
cin >> k[i].v;
cout << "\n Enter cost of edge :";
cin >> k[i].cost;
}
}
void Kruskals :: Adjust(int i,int s)
{
int j=2*i;
edge item = k[i];
while(j <= s)
{
if((j<s) && (k[j].cost>k[j+1].cost))
{
j++;
}
if(item.cost <= k[j].cost)
break;
k[j/2]=k[j];
j=2*j;
}
k[j/2]=item;
}
void Kruskals :: Heapify()
```

```
{
for(int i=e/2;i>=1;i--)
Adjust(i,e);
}
int Kruskals :: Kruskal()
{
edge temp;
int mincost;
for(int i=1;i<=n;i++)
{
p[i]=-1;
}
int i=0;
int size=e;
mincost=0;
int u1,v1,c;
while(i<=n-1 && size > 0)
{
u1=k[1].u;
v1=k[1].v;
c=k[1].cost;
temp=k[1];
k[1]=k[size];
k[size]=temp;
size--;
Adjust(1,size);
int j=Find(u1);
int k=Find(v1);
if(j!=k)
```

```
{
i++;
t[i][1]=u1;
t[i][2]=v1;
mincost=mincost+c;
p[j]=k;
}
}
if(i != n-1)
{
cout <<"\n No spanning Tree.";
return -1;
}
else
return mincost;
}
int Kruskals :: Find(int i)
{
while(p[i] >= 0)
i=p[i];
return(i);
}
void Kruskals :: Display()
{
int u,v;
cout << "\n Spanning Tree ";
for(int i=1;i<n;i++)
{
u=t[i][1];
```

```
v=t[i][2];
cout << "\n[" << u << " , "<< v<<"]";
}
}
int main()
{
Kruskals k;
k.Get();
k.Heapify();
int x = k.Kruskal();
cout << "\n";
k.Display();
cout << "\n Minimum cost of spanning Tree is : " << x;
}
```

4) Write a program to find Single Source Shortest Path using Dijkstra's algorithm.

```
#include<iostream>
#include<conio.h>
using namespace std;
class SSSP
{
int cost[20][20],dist[20],s[20],n,u,v;
public:
void getdata();
int minimum(int *);
void Shortest_Path();
void display();
};
void SSSP::getdata()
{
cout<<"\nNumber Of Vertices:";
cin>>n;
cout << "\n Enter the cost matrix : \n";
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
cin>>cost[i][j] ;
}
}
cout<<"\n Enter starting vertex:\n";
cin>>v;
```

```
}
void SSSP::display()
{
cout<<"\nDistance Matrix Is...";
for(int i=1;i<=n;i++)
{
cout<<"\n"<<dist[i];
}
}
int SSSP::minimum(int *a)
{
int min,x=999;
for(int i=2;i<=n;i++)
{
if(s[i] == 0)
{
if(dist[i] < x)
{
x=a[i];
min = i;
}
}
}
return min;
}
void SSSP::Shortest_Path()
{
int min;
for(int i=1;i<=n;i++)
```

```
{
s[i] = 0;
dist[i] = cost[v][i];
}
for(int num=2;num<=n-1;num++)
{
min = minimum(dist);
s[min] = 1;
u=min;
for(int w=1;w<=n;w++)
{
 if(s[w] == 0)
 {
 if (dist[w] > dist[u]+ cost[u][w])
dist[w] = dist[u]+ cost[u][w];
 }
}
}
}
int main()
{
SSSP obj;
obj.getdata();
obj.Shortest_Path();
obj.display();
}
```

Pract-5 Dynamic Programming

1) Write a program to find solution of Knapsack Instance (0/1)

```
#include<iostream>
#include<conio.h>
using namespace std;
class Knapsack_01
{
public:
int b[10],n,wt[10],W,i,j,B[10][10];
void Getdata()
{
cout<<"\nEnter number of items: ";
cin>>n;
cout<<"\nEnter maximum capacity of bag: ";
cin>>W;
cout<<"\nEnter weight of each item: ";
for(i=1;i<=n;i++)
{
cin>>wt[i];
}
cout<<"\nEnter benefit of each item: ";
for(i=1;i<=n;i++)
{
cin>>b[i];
}
}
int Knapsack()
```

```
{
 for(int w=0;w<=W;w++)
B[0][w]=0;
 for(i=0;i<=n;i++)
B[i][0]=0;
 cout<<"\n\n";
for(i=1;i<=n;i++)
{
for(int w=0;w<=W;w++)
{
if(wt[i]<=w)
{
if((b[i]+B[i-1][w-wt[i]])>B[i-1][w])
{
B[i][w]=b[i]+B[i-1][w-wt[i]];
}
else
{
B[i][w]=B[i-1][w];
}
}
else
B[i][w]=B[i-1][w];
}
}
cout<<"\nMatrix B is: \n";
for(i=0;i<n+1;i++)
{
for(int w=0;w<=W;w++)
```

```
{
cout<<" "<<B[i][w];
}
cout<<"\n";
}
cout<<"\nMaximum profit is: "<<B[n][W];
}
};
int main()
{
Knapsack_01 k;
k.Getdata();
k.Knapsack();
}
```

2) Write a program to find solution of Matrix Chain Multiplication.

```
#include<iostream>
#include<conio.h>
#include<string.h>
using namespace std;
class MCM
{
int P[10],S[10][10];
long M[10][10],q;
char A[20];
int j,r;
public:
void Get();
void Matrix_Chain_Order();
void Print_Optimal_Parens(int S[10][10],int,int);
};
void MCM :: Get()
{
cout << "\n Enter the size of matrix list:";
cin >> r;
cout << "\n Enter the matrix list:";
for(int i=1;i<=r;i++)
{
cin >> A[i];
}
cout << "\n Enter the dimensions:";
for(int i=0;i<=r;i++)
{
cin >> P[i];
```

```
}
}
void MCM :: Matrix_Chain_Order()
{
int n=r;
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
M[i][j]=0;
S[i][j]=0;
}
}
for(int l=2;l<=n;l++)
{
for(int i=1;i<=n-l+1;i++)
{
j=i+l-1;
M[i][j]=99999;
for(int k=i;k<=j-1;k++)
{
q=(M[i][k]+M[k+1][j]+(P[i-1]*P[k]*P[j]));
if(q < M[i][j])
{
M[i][j]=q;
S[i][j]=k;
}
}
}
```

```
}
cout <<"\n M[i][j] : \n";
for(int i=1;i<=r;i++)
{
cout << "\n";
for(int j=1;j<=r;j++)
{
cout << "\t" << M[i][j];
}
}
cout <<"\n\n\n S[i][j] : \n";
for(int i=1;i<=r;i++)
{
cout << "\n";
for(int j=1;j<=r;j++)
{
cout << "\t" << S[i][j];
}
}
cout << "\n\n\n\n";
Print_Optimal_Parens(S,1,r);
}
void MCM :: Print_Optimal_Parens(int S[10][10],int i,int j)
{
if(i==j)
{
cout << "A" <<i;
}
else
```

```
{
cout << "(";
Print_Optimal_Parens(S,i,S[i][j]);
Print_Optimal_Parens(S,S[i][j]+1,j);
cout << ")";
}
}
int main()
{
MCM m;
m.Get();
m.Matrix_Chain_Order();
}
```

3) Write a program to find shortest path using All Pair Shortest Path algorithm.

```
#include<iostream>
#include<conio.h>
using namespace std;
class Allpath
{
int cost[20][20],n;
public:
void get();
void path();
int min(int,int);
};
void Allpath::get()
{
cout<<"\nEnter no. of vertices :";
cin>>n;
cout<<"\nEnter cost :";
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
cin>>cost[i][j];
}
}
}
void Allpath::path()
{
```

```
int A[20][20];
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
A[i][j]=cost[i][j];
}
}
for(int k=1;k<=n;k++)
{
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
{
A[i][j]=min(A[i][j],A[i][k]+A[k][j]);
}
}
cout<<endl<<"A"<<k<<endl;
for(int i=1;i<=n;i++)
{
cout<<"\n";
for(int j=1;j<=n;j++)
{
cout<<"\t";
cout<<A[i][j];
}
}
}
}
```

```
int Allpath::min(int a,int b)
{
if(a>b)
return b;
else
return a;
}
int main()
{
Allpath p;
p.get();
p.path();
}
```

4) Write a program to Traverse Graph Depth First Search Breadth First Search.

1) DFS

```
#include<iostream>
#include<conio.h>
using namespace std;
class DFS
{
int Matrix[10][10],n;
int visited[10],Res[10];
public:
void DFT();
void Get();
void Dfs(int);
};
void DFS :: DFT()
{
for(int i=1;i<=n;i++)
visited[i]=0;
for(int i=1;i<=n;i++)
if(visited[i]==0)
Dfs(i);
}
void DFS :: Get()
{
cout << "\n Enter the size of matrix : ";
cin >> n;
cout << "\n Enter the matrix : ";
```

```
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
cin >> Matrix[i][j];
for(int i=1;i<=n;i++)
visited[i]=0;
}
void DFS :: Dfs(int v)
{
visited[v]=1;
cout << v << "\t";
for(int w=1;w<=n;w++)
{
if(Matrix[v][w]==1)
{
if(visited[w]==0)
{
Dfs(w);
}
}
}
}
int main()
{
DFS d;
d.Get();
d.DFT();
}
```

2) BFS

```
#include<iostream>
#include<conio.h>
using namespace std;
class BFS
{
int Matrix[10][10],n;
int visited[10],Res[10];
int q[10],Rear,Front;
public:
void BFT();
void Get();
void Bfs(int);
};
void BFS :: BFT()
{
for(int i=1;i<=n;i++)
visited[i]=0;
for(int i=1;i<=n;i++)
if(visited[i]==0)
Bfs(i);
}
void BFS :: Get()
{
Rear=Front=1;
cout << "\n Enter the size of matrix : ";
cin >> n;
cout << "\n Enter the matrix : ";
```

```
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
cin >> Matrix[i][j];
for(int i=1;i<=n;i++)
visited[i]=0;
}
void BFS :: Bfs(int v)
{
for(int i=1;i<=n;i++)
visited[i]=0;
int u=v;
visited[v]=1;
cout << v << "\t";
do
{
for(int w=1;w<=n;w++)
{
if(Matrix[u][w] == 1)
{
if(visited[w]==0)
{
q[Rear++]=w;
visited[w]=1;
cout << w<<"\t";
}
}
}
if(Front == Rear)
break;
```

```
u=q[Front++];
}while(1);
}
        int main()
        {
        BFS b;
        b.Get();
        b.BFT();
        }
```

Pract-6 Backtracking

1) Write a program to find all solutions for N- Queen problem using backtracking.

```
#include<iostream>
#include<conio.h>
#include<math.h>
using namespace std;
class NQueen
{
int i, X[10],k,count,N;
public:
NQueen()
{
k=1;
count=1;
X[k]=0;
}
void Nqueen(void);
int Place(int);
};
void NQueen :: Nqueen(void)
{
cout << "\n Enter the numbers of queens :";
cin >> N;
while(k>0)
{
```

```
X[k]++;
if ((k==1) && (X[k] > N/2)) // for mirror images
{
cout << "\n\n\n The total no of solutions of " << N <<" queen after removing mirror images are :" << count-1;
return; // for mirror images
}
while(X[k] <= N && Place(k) == 0)
X[k]=X[k]+1;
if(X[k] <= N)
{
if(k==N)
{
cout << "\n Solutions No : " << count << endl;
for(i=1;i<=N;i++)
cout << "\t" << X[i];
count++;
}
else
{
k++;
X[k]=0;
}
}
else
{
k--;
}
}
cout << "\n The total no of solutions of " << N <<" queen problem is :" << count-1;
```

```
}
int NQueen :: Place(int k)
{
for(i=1;i<k;i++)
{
if((X[i] == X[k]) || (abs(X[i]-X[k])==abs(i-k)))
return 0;
}
return 1;
}
int main()
{
NQueen q;
q.Nqueen();
}
```

2) Write a program for Graph Coloring using Backtracking.

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<process.h>
using namespace std;
class GraphColor
{
int X[10],m,G[10][10],N,j;
public:
void GetData();
int NextValue(int);
void Mcoloring(int);
};
void GraphColor :: GetData()
{
cout << "\n Enter the numbers of nodes : ";
cin >> N;
cout << "\n Enter Graph : \n";
for(int i=1;i<=N;i++)
{
for(int j=1;j<=N;j++)
{
cin >> G[i][j];
X[j]=0;
}
}
cout << "\n Enter the colors : ";
```

```
cin >> m;
}
void GraphColor :: Mcoloring(int k)
{
while(1)
{
NextValue(k);
if(X[k]==0)
exit(0);
if(k==N)
{
for(int i=1;i<=N;i++)
cout << "\n Node " <<i<<" is colored with color " << X[i];
cout << endl;
getch();
exit(0);
}
else
{
Mcoloring(k+1);
}
}
}
int GraphColor :: NextValue(int k)
{
int j;
while(1)
{
X[k]=(X[k]+1)%(m+1);
```

```
if(X[k]==0)
{
cout << "\n Color is not sufficient";
getch();
return 0;
}
else
{
for(j=1;j<=N;j++)
{
if(G[j][k]!=0 && X[k]==X[j])
break;
}
if(j==N+1)
return (0);
}
}
}
int main()
{
GraphColor g;
g.GetData();
g.Mcoloring(1);
}
```