UIET MDU
Rohtak

Design and Analysis of Algorithms
Lab File

Submitted To:

Ms. Rainu Nandal
Assistant Professor

Submitted By:

Payal

Roll No. : 23609

CSE-II(5th Sem.)

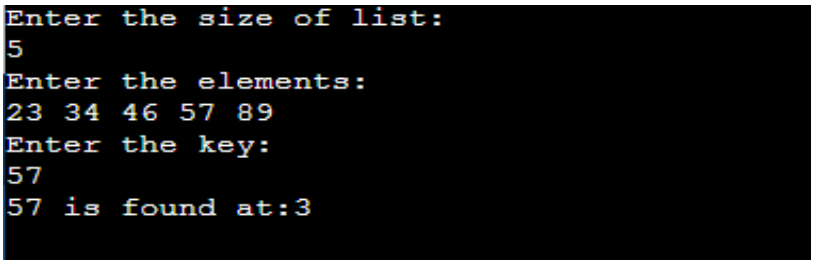
Sr. No.	Programs	Remarks
1	Binary Search	
2	Quick & Merge Sort	
3	Strassen's Matrix Multiplication	
4	Fractional Knapsack Problem	
5	Prim's & Kruskal's Algorithm	
6	Dijkstra Algorithm	
7	Optimal Binary Search Tree	
8	0/1 Knapsack Problem	
9	Travelling Salesman Problem	
10	8 Queens Problem	
11	Graph Coloring Problem	
12	Hamiltonian Cycle	

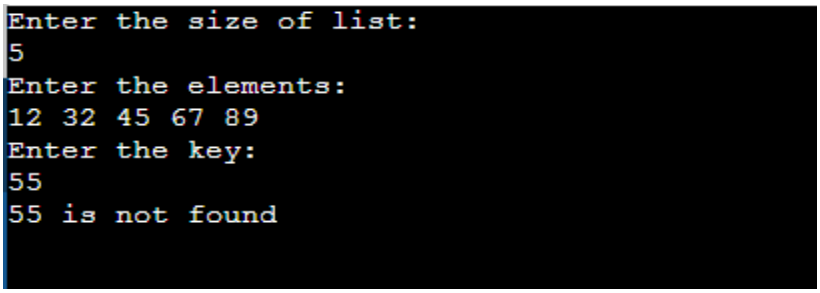
Binary Search

Divide and Conquer Strategy

```
#include<iostream>
using namespace std;
 int main()
{
   int n;
   cout<<"Enter the size of list:"<<endl;
   cin>>n;

   int list[n];
   cout<<"Enter the elements:"<<endl;
   for(int i=0;i<n;i++)
   cin>>list[i];

   int key;
   cout<<"Enter the key:"<<endl;
   cin>>key;

   int loc=-1;
   int beg=0;
   int end=n-1;
   while(beg<=end)
   {
      int mid=(beg+end)/2;
      if(list[mid]==key)
      {
         loc=mid;
         break;
      }
      else if(list[mid]>key)
```

```
        end=mid-1;
        else
        beg=mid+1;
    }

    if(loc!=-1)
    cout<<key<<" is found at:"<<loc<<endl;
    else
    cout<<key<<" is not found"<<endl;
}
```





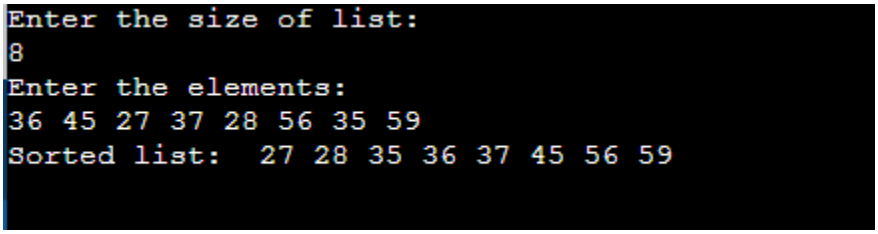
Merge Sort & Quick Sort

Divide and Conquer Strategy

Quick Sort

```
#include<iostream>
using namespace std;

int partition(int list[],int n,int lb,int ub)
{
   int beg=lb;
   int end=ub;
   int pivot=list[lb];
   while(beg<end)
   {
      while(beg<ub&&list[beg]<=pivot)
      beg++;
      while(list[end]>pivot)
      end--;
      if(beg<end)
      list[end]=list[end]+list[beg]-(list[beg]=list[end]);
   }
   list[lb]=list[lb]+list[end]-(list[end]=list[lb]);

   return end;
}

void quickSort(int list[],int n,int lb,int ub)
{
   if(lb<ub)
   {
      int loc=partition(list,n,lb,ub);
      quickSort(list,n,lb,loc-1);
      quickSort(list,n,loc+1,ub);
   }
}
```

```
int main()
{
   int n;
   cout<<"Enter the size of list:"<<endl;
   cin>>n;

   int list[n];
   cout<<"Enter the elements:"<<endl;
   for(int i=0;i<n;i++)
   cin>>list[i];

   cout<<"Sorted list:  ";
   quickSort(list,n,0,n-1);
   for(int i=0;i<n;i++)
   cout<<list[i]<<" ";
   cout<<endl;

}
```



Merge Sort

```
#include<iostream>
using namespace std;

void merge(int list[],int n,int lb,int mid,int ub)
{
   int x=lb;
   int y=mid+1;
   int z=0;
   int b[ub-lb+1];
```

```
    while(x<=mid&&y<=ub)
    {
        if(list[x]<list[y])
        {
            b[z]=list[x];
            x++;
        }
        else
        {
            b[z]=list[y];
            y++;
        }
        z++;
    }
    if(x>mid)
    {
        while(y<=ub)
        {
            b[z]=list[y];
            y++;z++;
        }
    }
    else
    {
        while (x <=mid)
        {
            b[z] = list[x];
            x++;z++;
        }
    }
    int i=lb;
    for(int j=0;j<(ub-lb+1);j++)
    {
        list[i] = b[j];
        i++;
    }
}

void mergeSort(int list[],int n,int lb,int ub)
```

```
{
   if(lb<ub)
   {
      int mid=(lb+ub)/2;
      mergeSort(list,n,lb,mid);
      mergeSort(list,n,mid+1,ub);
      merge(list,n,lb,mid,ub);
   }
}

int main()
{
   int n;
   cout << "Enter the size of list:" << endl;
   cin >> n;

   int list[n];
   cout << "Enter the elements:" << endl;
   for (int i = 0; i < n; i++)
      cin >> list[i];

   cout << "Sorted list:  ";
   mergeSort(list,n,0,n-1);
   for (int i = 0; i < n; i++)
      cout << list[i] << " ";
   cout << endl;
}
```

```
Enter the size of list:
8
Enter the elements:
54 36 28 56 14 46 39 89
Sorted list:  14 28 36 39 46 54 56 89
```

Strassen's Matrix Multiplication
Divide and Conquer

```
#include <bits/stdc++.h>
using namespace std;
typedef long long lld;


/* Strassen's Algorithm for matrix multiplication
  Complexity:    O(n^2.808) */


inline lld** MatrixMultiply(lld** a, lld** b, int n,
                            int l, int m)
{
   lld** c = new lld*[n];
   for (int i = 0; i < n; i++)
      c[i] = new lld[m];


   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         c[i][j] = 0;
         for (int k = 0; k < l; k++) {
            c[i][j] += a[i][k] * b[k][j];
         }
      }
   }
   return c;
}


inline lld** Strassen(lld** a, lld** b, int n,
                      int l, int m)
{
   if (n == 1 || l == 1 || m == 1)
      return MatrixMultiply(a, b, n, l, m);


   lld** c = new lld*[n];
   for (int i = 0; i < n; i++)
```

```
   c[i] = new lld[m];


int adjN = (n >> 1) + (n & 1);
int adjL = (l >> 1) + (l & 1);
int adjM = (m >> 1) + (m & 1);


lld**** As = new lld***[2];
for (int x = 0; x < 2; x++) {
   As[x] = new lld**[2];
   for (int y = 0; y < 2; y++) {
      As[x][y] = new lld*[adjN];
      for (int i = 0; i < adjN; i++) {
         As[x][y][i] = new lld[adjL];
         for (int j = 0; j < adjL; j++) {
            int I = i + (x & 1) * adjN;
            int J = j + (y & 1) * adjL;
            As[x][y][i][j] = (I < n && J < l) ? a[I][J] : 0;
         }
      }
   }
}


lld**** Bs = new lld***[2];
for (int x = 0; x < 2; x++) {
   Bs[x] = new lld**[2];
   for (int y = 0; y < 2; y++) {
      Bs[x][y] = new lld*[adjN];
      for (int i = 0; i < adjL; i++) {
         Bs[x][y][i] = new lld[adjM];
         for (int j = 0; j < adjM; j++) {
            int I = i + (x & 1) * adjL;
            int J = j + (y & 1) * adjM;
            Bs[x][y][i][j] = (I < l && J < m) ? b[I][J] : 0;
         }
      }
   }
}


lld*** s = new lld**[10];
```

```
for (int i = 0; i < 10; i++) {
  switch (i) {
  case 0:
    s[i] = new lld*[adjL];
    for (int j = 0; j < adjL; j++) {
      s[i][j] = new lld[adjM];
      for (int k = 0; k < adjM; k++) {
        s[i][j][k] = Bs[0][1][j][k] - Bs[1][1][j][k];
      }
    }
    break;
  case 1:
    s[i] = new lld*[adjN];
    for (int j = 0; j < adjN; j++) {
      s[i][j] = new lld[adjL];
      for (int k = 0; k < adjL; k++) {
        s[i][j][k] = As[0][0][j][k] + As[0][1][j][k];
      }
    }
    break;
  case 2:
    s[i] = new lld*[adjN];
    for (int j = 0; j < adjN; j++) {
      s[i][j] = new lld[adjL];
      for (int k = 0; k < adjL; k++) {
        s[i][j][k] = As[1][0][j][k] + As[1][1][j][k];
      }
    }
    break;
  case 3:
    s[i] = new lld*[adjL];
    for (int j = 0; j < adjL; j++) {
      s[i][j] = new lld[adjM];
      for (int k = 0; k < adjM; k++) {
        s[i][j][k] = Bs[1][0][j][k] - Bs[0][0][j][k];
      }
    }
    break;
  case 4:
    s[i] = new lld*[adjN];
    for (int j = 0; j < adjN; j++) {
      s[i][j] = new lld[adjL];
```

```
      for (int k = 0; k < adjL; k++) {
        s[i][j][k] = As[0][0][j][k] + As[1][1][j][k];
      }
    }
    break;
  case 5:
    s[i] = new lld*[adjL];
    for (int j = 0; j < adjL; j++) {
      s[i][j] = new lld[adjM];
      for (int k = 0; k < adjM; k++) {
        s[i][j][k] = Bs[0][0][j][k] + Bs[1][1][j][k];
      }
    }
    break;
  case 6:
    s[i] = new lld*[adjN];
    for (int j = 0; j < adjN; j++) {
      s[i][j] = new lld[adjL];
      for (int k = 0; k < adjL; k++) {
        s[i][j][k] = As[0][1][j][k] - As[1][1][j][k];
      }
    }
    break;
  case 7:
    s[i] = new lld*[adjL];
    for (int j = 0; j < adjL; j++) {
      s[i][j] = new lld[adjM];
      for (int k = 0; k < adjM; k++) {
        s[i][j][k] = Bs[1][0][j][k] + Bs[1][1][j][k];
      }
    }
    break;
  case 8:
    s[i] = new lld*[adjN];
    for (int j = 0; j < adjN; j++) {
      s[i][j] = new lld[adjL];
      for (int k = 0; k < adjL; k++) {
        s[i][j][k] = As[0][0][j][k] - As[1][0][j][k];
      }
    }
    break;
  case 9:
```

```
        s[i] = new lld*[adjL];
        for (int j = 0; j < adjL; j++) {
            s[i][j] = new lld[adjM];
            for (int k = 0; k < adjM; k++) {
                s[i][j][k] = Bs[0][0][j][k] + Bs[0][1][j][k];
            }
        }
        break;
    }
}


lld*** p = new lld**[7];
p[0] = Strassen(As[0][0], s[0], adjN, adjL, adjM);
p[1] = Strassen(s[1], Bs[1][1], adjN, adjL, adjM);
p[2] = Strassen(s[2], Bs[0][0], adjN, adjL, adjM);
p[3] = Strassen(As[1][1], s[3], adjN, adjL, adjM);
p[4] = Strassen(s[4], s[5], adjN, adjL, adjM);
p[5] = Strassen(s[6], s[7], adjN, adjL, adjM);
p[6] = Strassen(s[8], s[9], adjN, adjL, adjM);


for (int i = 0; i < adjN; i++) {
    for (int j = 0; j < adjM; j++) {
        c[i][j] = p[4][i][j] + p[3][i][j] - p[1][i][j] + p[5][i][j];
        if (j + adjM < m)
            c[i][j + adjM] = p[0][i][j] + p[1][i][j];
        if (i + adjN < n)
            c[i + adjN][j] = p[2][i][j] + p[3][i][j];
        if (i + adjN < n && j + adjM < m)
            c[i + adjN][j + adjM] = p[4][i][j] + p[0][i][j] - p[2][i][j] - p[6][i][j];
    }
}


for (int x = 0; x < 2; x++) {
    for (int y = 0; y < 2; y++) {
        for (int i = 0; i < adjN; i++) {
            delete[] As[x][y][i];
        }
        delete[] As[x][y];
    }
```

```
    delete[] As[x];
}
delete[] As;


for (int x = 0; x < 2; x++) {
    for (int y = 0; y < 2; y++) {
        for (int i = 0; i < adjL; i++) {
            delete[] Bs[x][y][i];
        }
        delete[] Bs[x][y];
    }
    delete[] Bs[x];
}
delete[] Bs;


for (int i = 0; i < 10; i++) {
    switch (i) {
    case 0:
    case 3:
    case 5:
    case 7:
    case 9:
        for (int j = 0; j < adjL; j++) {
            delete[] s[i][j];
        }
        break;
    case 1:
    case 2:
    case 4:
    case 6:
    case 8:
        for (int j = 0; j < adjN; j++) {
            delete[] s[i][j];
        }
        break;
    }
    delete[] s[i];
}
delete[] s;
```

```
    for (int i = 0; i < 7; i++) {
        for (int j = 0; j < (n >> 1); j++) {
            delete[] p[i][j];
        }
        delete[] p[i];
    }
    delete[] p;


    return c;
}


int main()
{
    lld** matA;
    matA = new lld*[2];
    for (int i = 0; i < 2; i++)
        matA[i] = new lld[3];
    matA[0][0] = 1;
    matA[0][1] = 2;
    matA[0][2] = 3;
    matA[1][0] = 4;
    matA[1][1] = 5;
    matA[1][2] = 6;


    lld** matB;
    matB = new lld*[3];
    for (int i = 0; i < 3; i++)
        matB[i] = new lld[2];
    matB[0][0] = 7;
    matB[0][1] = 8;
    matB[1][0] = 9;
    matB[1][1] = 10;
    matB[2][0] = 11;
    matB[2][1] = 12;


    lld** matC = Strassen(matA, matB, 2, 3, 2);
    for (int i = 0; i < 2; i++) {
```

```
        for (int j = 0; j < 2; j++) {
            printf("%lld ", matC[i][j]);
        }
        printf("\n");
    }


    return 0;
}
```

```
58 64
139 154
```

Fractional Knapsack
Greedy Method

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

bool compare(pair<int,int> p1,pair<int,int> p2)
{
   double f1=(double)p1.first/p1.second;
   double f2=(double)p2.first/p2.second;
   return f1>f2;
}

int main()
{
   int n;
   cout<<"Enter number of items:"<<endl;
   cin>>n;

   vector<pair<int,int>> v(n);
   cout<<"Enter the profits and weights of elements:"<<endl;
   for(int i=0;i<n;i++)
   {
      cin>>v[i].first>>v[i].second;
   }

   int capacity;
   cout<<"Enter the capacity of knapsack:"<<endl;
   cin>>capacity;

   sort(v.begin(),v.end(),compare);

   double profit=0;
   int weight=capacity;
```

```
    for(int i=0;i<n;i++)
    {
        if(weight-v[i].second<0)
        {
            double f=(double)weight/v[i].second;
            profit+=v[i].first*f;
            break;
        }
        else
        {
            weight-=v[i].second;
            profit+=v[i].first;
        }
    }

    cout<<"Profit is "<<profit<<endl;
}
```

```
Enter number of items:
7
Enter the profits and weights of elements:
5 1
10 3
15 5
7 4
8 1
9 3
4 2
Enter the capacity of knapsack:
15
Profit is 51
```

Minimum Spanning Tree Algorithms

Prims Algorithm
Greedy Method

```
#include<iostream>
#include<climits>
using namespace std;

int main()
{
   int n;
   cout<<"Enter the size of graph:"<<endl;
   cin>>n;

   int cost[n][n];
   int near[n];
   int tree[n-1][2];
   int minCost=0;
   int min=INT_MAX;
   int k,l;
   cout<<"Enter the cost matrix:"<<endl;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
          cin>>cost[i][j];

         if(cost[i][j]!=0&&cost[i][j]<min)
         {
            min=cost[i][j];
            k=i;l=j;
         }
      }
   }

   tree[0][0]=k;tree[0][1]=l;
   minCost=min;
   for(int i=0;i<n;i++)
   {
```

```
    if(cost[k][i]!=0&&cost[l][i]!=0)
    {
        if(cost[k][i]<cost[l][i])
        near[i]=k;
        else
        near[i]=l;
    }
    else
    {
        if(cost[k][i]==0)
        near[i]=l;
        else
        near[i]=k;
    }

}
near[k]=-1;
near[l]=-1;

for(int i=1;i<n-1;i++)
{
    min=INT_MAX;
    k=-1;
    for(int j=0;j<n;j++)
    {
        if(near[j]!=-1&&cost[near[j]][j]!=0&&cost[near[j]][j]<min)
        {
            min=cost[near[j]][j];
            k=j;
        }
    }
    tree[i][0]=k;tree[i][1]=near[k];
    minCost+=min;
    near[k]=-1;
    for(int j=0;j<n;j++)
    {
        if(near[j]!=-1&&cost[j][k]!=0&&cost[j][near[j]]>cost[j][k])
        near[j]=k;
    }
}

cout<<"Minimum Cost: "<<minCost<<endl;
```

```
    cout<<"Edges in MST are:"<<endl;
    for(int i=0;i<n-1;i++)
    cout<<tree[i][0]<<" "<<tree[i][1]<<endl;
}
```

```
Enter the size of graph:
6
Enter the cost matrix:
0 4 0 0 0 2
4 0 6 0 0 3
0 6 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 4
2 3 1 0 4 0
Minimum Cost: 11
Edges in MST are:
2 5
0 5
1 5
3 2
4 3
```

Kruskal Algorithm
Greedy Method

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> parent;
vector<int> sz;

int findSet(int v)
{
   if (parent[v] == v)
      return v;
   return findSet(parent[v]);
}

void unionSet(int a, int b)
{
   a = findSet(a);
   b = findSet(b);
   if (a != b)
   {
      if (sz[a] < sz[b])
         swap(a, b);
      parent[b] = a;
      sz[a] += sz[b];
   }
}

int main()
{
   int m, n;
   cout << "Enter the number of edges and vertices:" << endl;
   cin >> m >> n;
```

```
    for (int i = 0; i < n; i++)
    {
        parent.push_back(i);
        sz.push_back(1);
    }


    vector<vector<int>> edges;
    cout << "Enter edges and corresponding weights:" << endl;
    for (int i = 0; i < m; i++)
    {
        int w, u, v;
        cin >> w >> u >> v;
        edges.push_back({w, u, v});
    }
    sort(edges.begin(), edges.end());
    int minCost = 0;


    cout<<"Edges in MST are:"<<endl;
    for (auto i : edges)
    {
        int w = i[0];
        int u = i[1];
        int v = i[2];
        int x = findSet(u);
        int y = findSet(v);
        if (x == y)
            continue;
        else
        {
            cout << u << " " << v << endl;
            minCost += w;
            unionSet(u, v);
        }
    }
    cout <<"Minimum cost : "<< minCost << endl;
}
```

```
Enter the number of edges and vertices:
9 8
Enter edges and corresponding weights:
5 0 1
6 1 2
2 3 2
9 0 3
5 2 4
10 4 5
7 5 6
1 6 7
1 7 4
Edges in MST are:
6 7
7 4
3 2
0 1
2 4
1 2
5 6
Minimum cost : 27
```

Dijkstra Algorithm
Greedy Method

```
#include<iostream>
#include<climits>
using namespace std;

int minimum(int result[],bool visited[],int n)
{
   int min=INT_MAX;
   int loc=-1;
   for(int i=0;i<n;i++)
   {
      if(visited[i]==false&&result[i]<min)
      {
         min=result[i];
         loc=i;
      }
   }
   return loc;
}

int main()
{
   int n;
   cout<<"Enter the size of graph:"<<endl;
   cin>>n;

   int graph[n][n];
   cout<<"Enter the adjacency matrix:"<<endl;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      cin>>graph[i][j];
   }
```

```
    int src;
    cout<<"Enter the source vertex:"<<endl;
    cin>>src;

    int result[n];
    for(int i=0;i<n;i++)
    result[i]=INT_MAX;

    bool visited[n];
    for(int i=0;i<n;i++)
    visited[i]=false;

    result[src]=0;
    for(int i=0;i<n-1;i++)
    {
       int u=minimum(result,visited,n);
       visited[u]=true;
       for(int x=0;x<n;x++)
       {

if(visited[x]==false&&graph[u][x]!=0&&result[u]!=INT_MAX&&result[u]+graph[u][x]<result[x])
          result[x]=result[u]+graph[u][x];
       }
    }
    for(int i=0;i<n;i++)
    cout<<i<<"--->"<<result[i]<<endl;
}
```

```
Enter the size of graph:
9
Enter the adjacency matrix:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0
Enter the source vertex:
0
0--->0
1--->4
2--->12
3--->19
4--->21
5--->11
6--->9
7--->8
8--->14
```

Optimal Binary Search Tree
Dynamic Programming

```
#include<iostream>
#include<climits>
using namespace std;

int main()
{
   int n;
   cout<<"Enter the size of tree:"<<endl;
   cin>>n;

   int keys[n+1];
   cout<<"Enter the keys:"<<endl;
   for(int i=1;i<=n;i++)
   cin>>keys[i];

   int p[n+1];
   cout<<"Enter the probabilities of successful search:"<<endl;
   for(int i=1;i<=n;i++)
   cin>>p[i];

   int q[n+1];
   cout<<"Enter the probabilities of unsuccessful search:"<<endl;
   for(int i=0;i<=n;i++)
   cin>>q[i];

   int cost[n+1][n+1];
   int w[n+1][n+1];
   int r[n+1][n+1];

   for(int g=0;g<=n;g++)
   {
      for(int i=0,j=g;j<=n;i++,j++)
      {
```

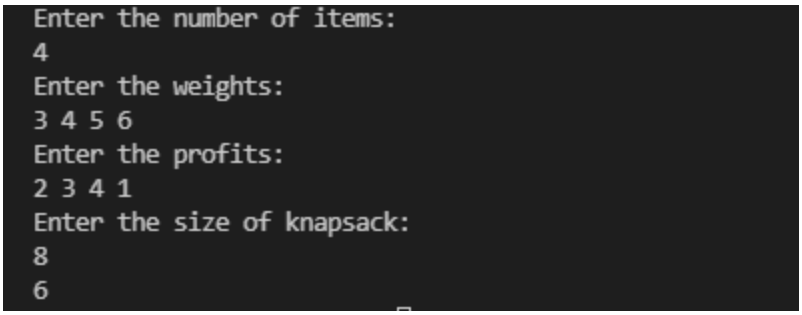
```
        if(g==0)
        {
           w[i][j]=q[i];
           cost[i][j]=0;
           r[i][j]=0;
        }
        else
        {
           w[i][j]=w[i][j-1]+p[j]+q[j];
           int min=INT_MAX;
           for(int k=i+1;k<=j;k++)
           {
              if(cost[i][k-1]+cost[k][j]<min)
                 {
                    min = cost[i][k - 1] + cost[k][j];
                    r[i][j] = k;
                 }
           }
           cost[i][j]=min+w[i][j];
        }
     }
  }
  cout<<cost[0][n]<<endl;
  cout<<r[0][n]<<endl;

}
```

```
Enter the size of tree:
4
Enter the keys:
10 20 30 40
Enter the probabilities of successful search:
3 3 1 1
Enter the probabilities of unsuccessful search:
2 3 1 1 1
Cost:32
Root of tree:2
```

0/1 Knapsack
Dynamic Programming

```
#include<iostream>
using namespace std;


int main()
{
   int n;
   cout<<"Enter the number of items:"<<endl;
   cin>>n;


   int w[n+1];
   cout<<"Enter the weights:"<<endl;
   for(int i=1;i<=n;i++)
   cin>>w[i];


   int p[n+1];
   cout<<"Enter the profits:"<<endl;
   for(int i=1;i<=n;i++)
   cin>>p[i];


   int m;
   cout<<"Enter the size of knapsack:"<<endl;
   cin>>m;


   int dp[n+1][m+1];
   for(int i=0;i<=n;i++)
   {
      for(int j=0;j<=m;j++)
      {
         if(i==0||j==0)
         dp[i][j]=0;
         else if(j<w[i])
```

```
        {
            dp[i][j]=dp[i-1][j];
        }
        else
        {
            if(p[i]+dp[i-1][j-w[i]]>dp[i-1][j])
            dp[i][j]=p[i]+dp[i-1][j-w[i]];
            else
            dp[i][j]=dp[i-1][j];
        }
      }
   }
   cout<<dp[n][m]<<endl;
}
```



Travelling Salesperson Problem
Dynamic Programming

```
#include<iostream>
#include<climits>
using namespace std;


int costMatrix[100][100];
int result[100];
int n;


bool isSafe(int k,int x)
{
   for(int i=0;i<k;i++)
   {
      if(result[i]==x)
      return false;
   }
   return true;
}


int tsp(int k)
{
   int min=INT_MAX;
   for(int i=0;i<n;i++)
   {
      if(k==n)
      {
         return costMatrix[result[0]][result[n-1]];
      }
      if(isSafe(k,i))
      {
         result[k]=i;
         int cost=costMatrix[i][result[k-1]]+tsp(k+1);
         if(cost<min)
```

```
            min=cost;
        }
    }
    return min;
}


int main()
{
    cout<<"Enter the size of graph:"<<endl;
    cin>>n;


    cout<<"Enter the cost matrix:"<<endl;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        cin>>costMatrix[i][j];
    }


    int src;
    cout<<"Enter the start vertex:"<<endl;
    cin>>src;


    result[0]=src;
    int minCost=tsp(1);
    cout << "Minimum cost is:" << minCost<< endl;
}
```

```
Enter the size of graph:
4
Enter the cost matrix:
0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0
Enter the start vertex:
0
Minimum cost is:35
```

8 Queens Problem
Backtracking

```
#include<iostream>
using namespace std;


bool isSafe(int result[],int row,int col,int n)
{
   for(int i=0;i<row;i++)
   {
      if(result[i]==col||abs(row-i)==abs(col-result[i]))
      return false;
   }
   return true;
}


void placeNQueen(int result[],int row,int n)
{
   for(int col=0;col<n;col++)
   {
      if(row==n)
      {
         for(int i=0;i<n;i++)
         cout<<result[i]<<" ";
         cout<<endl;
         return;
      }
      if(isSafe(result,row,col,n))
      {
         result[row]=col;
         placeNQueen(result,row+1,n);
      }
   }
}


int main()
```

```
{
   int n;
   cout<<"Enter n:"<<endl;
   cin>>n;
   int result[n];


   placeNQueen(result,0,n);


}
```

```
Enter size of chessboard:
8
0 4 7 5 2 6 1 3
0 5 7 2 6 3 1 4
0 6 3 5 7 1 4 2
0 6 4 7 1 3 5 2
1 3 5 7 2 0 6 4
1 4 6 0 2 7 5 3
1 4 6 3 0 7 5 2
1 5 0 6 3 7 2 4
1 5 7 2 0 3 6 4
1 6 2 5 7 4 0 3
1 6 4 7 0 3 5 2
1 7 5 0 2 4 6 3
2 0 6 4 7 1 3 5
2 4 1 7 0 6 3 5
2 4 1 7 5 3 6 0
2 4 6 0 3 1 7 5
2 4 7 3 0 6 1 5
2 5 1 4 7 0 6 3
2 5 1 6 0 3 7 4
2 5 1 6 4 0 7 3
2 5 3 0 7 4 6 1
2 5 3 1 7 4 6 0
2 5 7 0 3 6 4 1
2 5 7 0 4 6 1 3
2 5 7 1 3 0 6 4
2 6 1 7 4 0 3 5
2 6 1 7 5 3 0 4
2 7 3 6 0 5 1 4
```

```
3 0 4 7 5 2 6 1
3 1 4 7 5 0 2 6
3 1 6 2 5 7 0 4
3 1 6 2 5 7 4 0
3 1 6 4 0 7 5 2
3 1 7 4 6 0 2 5
3 1 7 5 0 2 4 6
3 5 0 4 1 7 2 6
3 5 7 1 6 0 2 4
3 5 7 2 0 6 4 1
3 6 0 7 4 1 5 2
3 6 2 7 1 4 0 5
3 6 4 1 5 0 2 7
3 6 4 2 0 5 7 1
3 7 0 2 5 1 6 4
3 7 0 4 6 1 5 2
3 7 4 2 0 6 1 5
4 0 3 5 7 1 6 2
4 0 7 3 1 6 2 5
4 0 7 5 2 6 1 3
4 1 3 5 7 2 0 6
4 1 3 6 2 7 5 0
4 1 5 0 6 3 7 2
4 1 7 0 3 6 2 5
4 2 0 5 7 1 3 6
4 2 0 6 1 7 5 3
4 2 7 3 6 0 5 1
4 6 0 2 7 5 3 1
4 6 0 3 1 7 5 2
4 6 1 3 7 0 2 5
4 6 1 5 2 0 3 7
4 6 1 5 2 0 7 3
4 6 3 0 2 7 5 1
4 7 3 0 2 5 1 6
4 7 3 0 6 1 5 2
```

```
5 0 4 1 7 2 6 3
5 1 6 0 2 4 7 3
5 1 6 0 3 7 4 2
5 2 0 6 4 7 1 3
5 2 0 7 3 1 6 4
5 2 0 7 4 1 3 6
5 2 4 6 0 3 1 7
5 2 4 7 0 3 1 6
5 2 6 1 3 7 0 4
5 2 6 1 7 4 0 3
5 2 6 3 0 7 1 4
5 3 0 4 7 1 6 2
5 3 1 7 4 6 0 2
5 3 6 0 2 4 1 7
5 3 6 0 7 1 4 2
5 7 1 3 0 6 4 2
6 0 2 7 5 3 1 4
6 1 3 0 7 4 2 5
6 1 5 2 0 3 7 4
6 2 0 5 7 4 1 3
6 2 7 1 4 0 5 3
6 3 1 4 7 0 2 5
6 3 1 7 5 0 2 4
6 4 2 0 5 7 1 3
7 1 3 0 6 4 2 5
7 1 4 2 0 6 3 5
7 2 0 5 1 4 6 3
7 3 0 2 5 1 6 4
```

Graph Coloring
Backtracking

```
#include<iostream>
using namespace std;

int graph[100][100];
int result[100];

bool isSafe(int color,int node)
{
   for(int i=0;i<node;i++)
   {
      if(graph[i][node]==1&&result[i]==color)
      return false;
   }
   return true;
}

void colorGraph(int n,int m,int node)
{
   for(int color=1;color<=m;color++)
   {
      if(node==n)
      {
         for(int i=0;i<n;i++)
         cout<<result[i]<<" ";
         cout<<endl;
         return;
      }
      else
      {
         if(isSafe(color,node))
         {
            result[node]=color;
            colorGraph(n,m,node+1);
         }
      }
   }
}
```

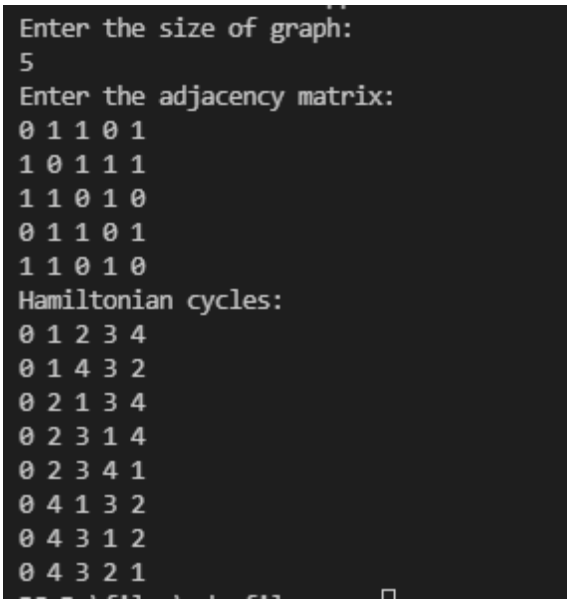
```
int main()
{
   int n;
   cout<<"Enter the size of graph:"<<endl;
   cin>>n;

   cout<<"Enter the adjacency matrix for graph:"<<endl;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      cin>>graph[i][j];
   }

   int m;
   cout<<"Enter number of colors:"<<endl;
   cin>>m;

   colorGraph(n,m,0);
}
```

```
Enter the size of graph:
4
Enter the adjacency matrix for graph:
1 1 0 1
1 1 1 0
0 1 1 1
1 0 1 1
Enter number of colors:
3
1 2 1 2
1 2 1 3
1 2 3 2
1 3 1 2
1 3 1 3
1 3 2 3
2 1 2 1
2 1 2 3
2 1 3 1
2 3 1 3
2 3 2 1
2 3 2 3
3 1 2 1
3 1 3 1
3 1 3 2
3 2 1 2
3 2 3 1
3 2 3 2
```

Hamiltonian Cycle
Backtraking

```
#include<iostream>
using namespace std;


int graph[100][100];
int result[100];
int n;


bool isSafe(int k,int v)
{
   if(graph[v][result[k-1]]==0)
   return false;
   for(int i=0;i<k;i++)
   {
      if(result[i]==v)
      return false;
   }
   return true;
}


void hamiltonian(int k)
{
   for(int i=1;i<n;i++)
   {
      if(k==n)
      {
         if (graph[result[0]][result[n - 1]] == 1)
         {
            for (int j = 0; j < n; j++)
               cout<<result[j]<<" ";
            cout<<endl;
         }
         return;
```

```
        }
        if (isSafe( k, i))
        {
            result[k] = i;
            hamiltonian( k + 1);
            result[k] = -1;
        }
    }
}


int main()
{
    cout<<"Enter the size of graph:"<<endl;
    cin>>n;


    cout<<"Enter the adjacency matrix:"<<endl;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            cin>>graph[i][j];
        }
    }



    for(int i=0;i<n;i++)
    result[i]=-1;
    result[0] = 0;
    cout<<"Hamiltonian cycles:"<<endl;
    hamiltonian(1);
}
```


Enter the size of graph:
5
Enter the adjacency matrix:
0 1 1 0 1
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
1 1 0 1 0
Hamiltonian cycles:
0 1 2 3 4
0 1 4 3 2
0 2 1 3 4
0 2 3 1 4
0 2 3 4 1
0 4 1 3 2
0 4 3 1 2
0 4 3 2 1