PROGRAM : To implement Optimal Binary Search tree

#include<iostream>

#define MAX 10

#define infinity 999

using namespace std;

int Find(int c[][MAX], int r[][MAX], int i, int j)

{

int MIN = infinity, l;

for(int m = r[i][j-1]; m <= r[i + 1][j]; m++)

{

if( (c[i][m - 1] + c[m][j]) < MIN)

{

MIN = c[i][m - 1] + c[m][j];

l = m;

}

}

return l;

}

void OBST(int p[], int q[], int n)

{

int w[10][10], r[10][10], c[10][10];

int j, k;

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

{

w[i][i] = q[i];

r[i][i] = 0;

c[i][i] = 0;

w[i][i+1] = q[i] + q[i+1] + p[i];

r[i][i+1] = i + 1;

c[i][i+1] = q[i] + q[i+1] + p[i];

}

w[n][n] = q[n];

r[n][n] = 0;

c[n][n] = 0;

for(int m = 2; m <= n; m++)

{

for(int i = 0; i <= n - m; i++)

{

j = i + m;

w[i][j] = w[i][j - 1] + p[j-1] + q[j];

k = Find(c, r, i, j);

c[i][j] = w[i][j] + c[i][k - 1] + c[k][j];

r[i][j] = k;

}

}

cout << "W - Matrix" << endl;

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

{

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

{

if(i > j)

cout << " " << "\t";

else

cout << w[i][j] << "\t";

}

cout << endl;

}

cout << "C - Matrix" << endl;

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

{

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

{

if(i > j)

cout << " " << "\t";

else

cout << c[i][j] << "\t";

}

cout << endl;

}

cout << "R - Matrix" << endl;

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

{

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

{

if(i > j)

cout << " " << "\t";

else

cout << r[i][j] << "\t";

}

cout << endl;

}

}

int main()

{

int p[10], q[10], n;

cout << "Enter the number of elements : ";

cin >> n;

cout << "Enter the pP array: "<<endl;

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

cin >> p[i];

cout << "Enter the Q array: "<<endl;

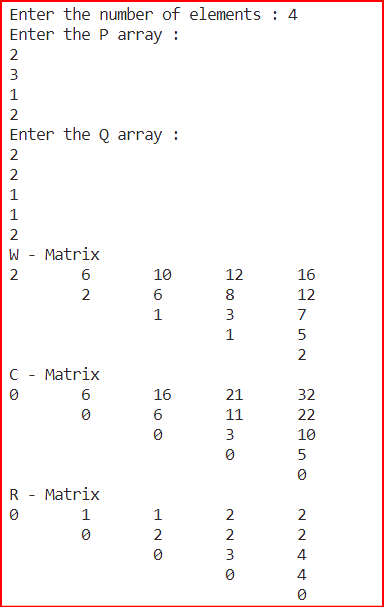
for(int i = 0; i <= n; i++)

cin >> q[i];

OBST(p, q, n);

}

OUTPUT :



PROGRAM : To implement Graph Colouring Algorithm

#include<iostream>

#define MAX 10

using namespace std;

int x[MAX] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, g[MAX][MAX];

int m, n;

void printSoln()

{

cout << "\nSolution Vector : \n( ";

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

cout << "x" << i << ", ";

cout << ") = ( ";

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

cout << x[i] << " ";

cout << " )";

}

void nextValue(int k)

{

int j;

do

{

x[k] = (x[k] + 1) % (m + 1);

if(x[k] == 0)

return;

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

{

if( (g[k][j] == 1) && (x[k] == x[j]) )

break;

}

if(j == n + 1)

return;

}while(true);

}

void mColor(int k)

{

do

{

nextValue(k);

if(x[k] == 0)

return;

if(k == n )

{

printSoln();

}

else

mColor(k+1);

}while(false);

}

int main()

{

cout << "Enter the number of colors : ";

cin >> m;

cout << "Enter the number of vertices : ";

cin >> n;

cout << "Enter the Adjacency Matrix : " << endl;

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

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

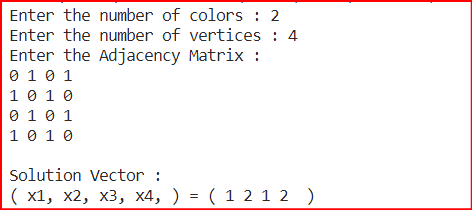
cin >> g[i][j];

mColor(1);

return 0;

}

OUTPUT :



PROGRAM : To implement Sum of Subset Problem

#include <iostream>

using namespace std;

#include <string.h>

#define MAX 20

int n, capacity;

int w[MAX];

void sum\_Of\_Subset(int sum, int item, int remainingCapacity, string solution\_Vector)

{

if ((sum == capacity) && (remainingCapacity == 0))

{

cout << solution\_Vector << endl;

}

if ((sum > capacity) || (remainingCapacity == 0))

{

return;

}

sum\_Of\_Subset(sum+w[item], item+1, remainingCapacity-w[item], solution\_Vector + "1 ");

sum\_Of\_Subset(sum, item+1, remainingCapacity-w[item], solution\_Vector + "0 ");

}

int main()

{

int totalW = 0;

string solution\_Vector = "";

cout << "Enter no. of elements : ";

cin >> n;

cout << "Enter the weights : ";

for(size\_t i{0}; i<n; i++)

{

cin >> w[i];

totalW += w[i];

}

cout << "Enter the Capacity : ";

cin >> capacity;

cout << endl << "Solution Vectors : " << endl;

sum\_Of\_Subset(0, 0, totalW, solution\_Vector);

return 0;

}

OUTPUT :

