Experiment No.1

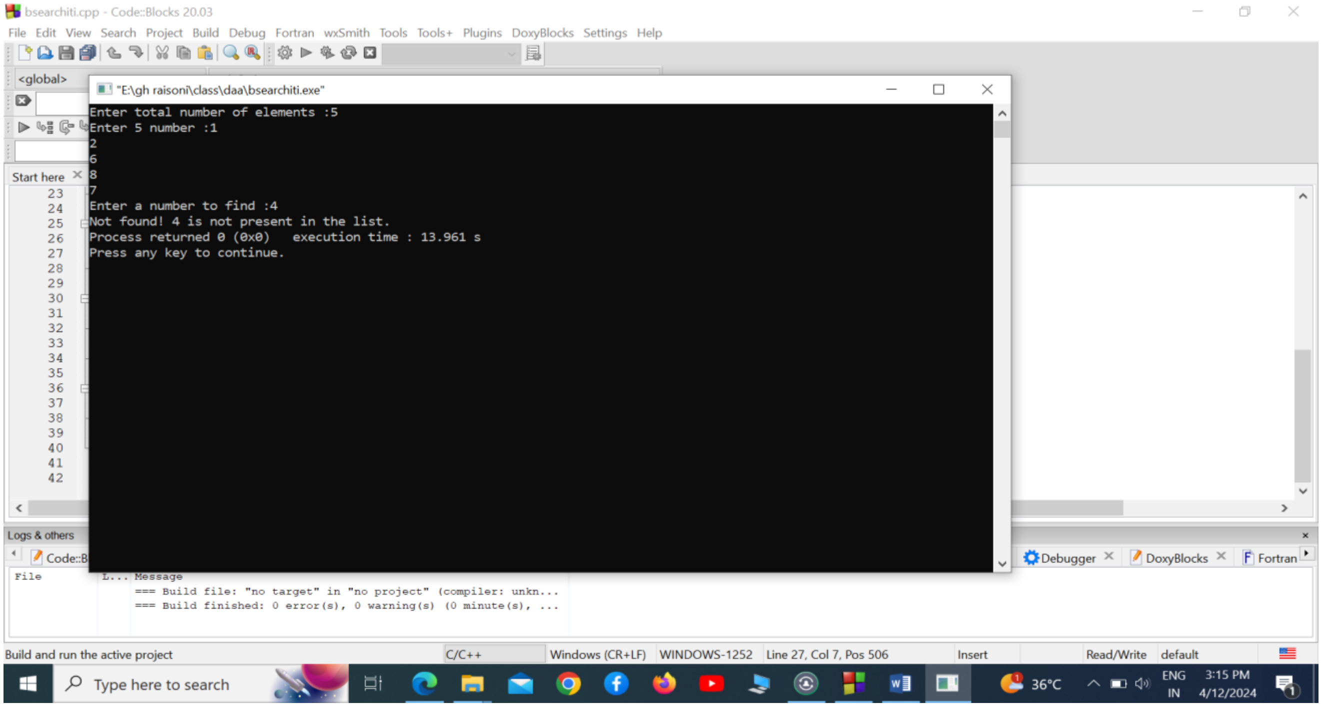
```
#include<iostream>
using namespace std;
int factorial(int n);
int main()
{
 int n;
 cout << "Enter a positive integer: ";
 cin >> n;
 cout << "Factorial of " << n << " = " << factorial(n);
 return 0;
}
int factorial(int n)
{
 if(n > 1)
 return n * factorial(n - 1);
 else
 return 1;
}
```

OUTPUT:

```
Enter a positive integer: 5
Factorial of 5 = 120
Process returned 0 (0x0)   execution time : 1.358 s
```

Experiment No.2

```
#include<iostream>

using namespace std;

int main()

{

int n, i, arr[50], search, first, last, middle;

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

cin>>n;

cout<<"Enter "<<n<<" number :";

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

{

cin>>arr[i];

}

cout<<"Enter a number to find :";

cin>>search;

first = 0;

last = n-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

cout<<search<<" found at location "<<middle+1<<"\n";

break;

}

else
{
last = middle - 1;
}
middle = (first + last)/2;
}
if(first > last)
{
cout<<"Not found! "<<search<<" is not present in the list.";
}
return 0;
}
```



bsearchiti.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Start here
"E:\gh raisoni\class\daa\bsearchiti.exe"
Logs & others
Code::B
Debugger
DoxyBlocks
Fortran
Build and run the active project
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 27, Col 7, Pos 506
Insert
Read/Write
default
Type here to search
36°C
ENG
IN
3:15 PM
4/12/2024

```
Enter total number of elements :5
Enter 5 number :1
2
6
8
7
Enter a number to find :4
Not found! 4 is not present in the list.
Process returned 0 (0x0)   execution time : 13.961 s
Press any key to continue.
```

```
File          L... Message
                   === Build file: "no target" in "no project" (compiler: unkn...
                   === Build finished: 0 error(s), 0 warning(s) (0 minute(s), ...
```

Experiment No.3

```
#include <iostream>
using namespace std;
void quick_sort(int[],int,int);
int partition(int[],int,int);
int main()
{
 int a[50],n,i;
 cout<<"How many elements?";
 cin>>n;
 cout<<"\nEnter array elements:";
 for(i=0;i<n;i++)
 cin>>a[i];

 quick_sort(a,0,n-1);
 cout<<"\nArray after sorting:";
 for(i=0;i<n;i++)
 cout<<a[i]<<" ";
 return 0;
}
void quick_sort(int a[],int l,int u)
{
 int j;
 if(l<u)
 {
 j=partition(a,l,u);
 quick_sort(a,l,j-1);
 quick_sort(a,j+1,u);
 }
}
```

```
int partition(int a[],int l,int u)
{
 int v,i,j,temp;
 v=a[l];
 i=l;
 j=u+1;
 do
 {
 do
 i++;
 while(a[i]<v&&i<=u);
 do
 j--;
 while(v<a[j]);
 if(i<j)
 {
 temp=a[i];
 a[i]=a[j];
 a[j]=temp;
 }
 }while(i<j);
 a[l]=a[j];
 a[j]=v;
 return(j);
}
```

OUTPUT:

```
How many elements?5


Enter array elements:4
```

```
2

1

8

6


Array after sorting:1 2 4 6 8

Process returned 0 (0x0)   execution time : 7.265 s
```

Experiment No.4

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#define V 5
using namespace std;
void printSolution(int path[]);
/* check if the vertex v can be added at index 'pos' in the Hamiltonian Cycle */
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
 if (graph [path[pos-1]][v] == 0)
 return false;
 for (int i = 0; i < pos; i++)
 if (path[i] == v)
 return false;
 return true;
}
/* solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
 if (pos == V)
 {
 if (graph[ path[pos-1] ][ path[0] ] == 1)
 return true;
 else
 return false;
 }
 for (int v = 1; v < V; v++)
 {
 if (isSafe(v, graph, path, pos))
 {
 path[pos] = v;
 if (hamCycleUtil (graph, path, pos+1) == true)
 return true;
 path[pos] = -1;
 }
 }
 return false;
}
/* solves the Hamiltonian Cycle problem using Backtracking.*/
bool hamCycle(bool graph[V][V])
{
 int *path = new int[V];
 for (int i = 0; i < V; i++)
 path[i] = -1;
 path[0] = 0;
 if (hamCycleUtil(graph, path, 1) == false)
 {
 cout<<"\nSolution does not exist"<<endl;
 return false;
 }
 printSolution(path);
 return true;
}
/* Main */
void printSolution(int path[])
{
 cout<<"Solution Exists:";
 cout<<" Following is one Hamiltonian Cycle \n"<<endl;
 for (int i = 0; i < V; i++)
 cout<<path[i]<<" ";
 cout<< path[0]<<endl;
}
int main()
{
 bool graph1[V][V] = {{0, 1, 0, 1, 0},
 {1, 0, 1, 1, 1},
 {0, 1, 0, 0, 1},
 {1, 1, 0, 0, 1},
 {0, 1, 1, 1, 0},
 };
 hamCycle(graph1);
 bool graph2[V][V] = {{0, 1, 0, 1, 0},
 {1, 0, 1, 1, 1},
 {0, 1, 0, 0, 1},
 {1, 1, 0, 0, 0},
 {0, 1, 1, 0, 0},
 };
 hamCycle(graph2);
 return 0;
}
```

OUTPUT:

```
Solution Exists: Following is one Hamiltonian Cycle

0 1 2 4 3 0


Solution does not exist
```

```
Process returned 0 (0x0)   execution time : 0.157 s

Press any key to continue.
```

Experiment No.5

```
#include<iostream>

#include<conio.h>

#include<stdio.h>

using namespace std;

int shortest(int ,int);

int cost[10][10],dist[20],i,j,n,k,m,S[20],v,totcost,path[20],p;

main()

{

int c;

cout <<"enter no of vertices";

cin >> n;

cout <<"enter no of edges";

cin >>m;

cout <<"\nenter\nEDGE Cost\n";

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

{

cin >> i >> j >>c;

cost[i][j]=c;

}

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

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

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

cost[i][j]=31999;

cout <<"enter initial vertex";

cin >>v;

cout << v<<"\n";

shortest(v,n);

}

int shortest(int v,int n)
```

```
{

int min;

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

{

S[i]=0;

dist[i]=cost[v][i];

}

path[++p]=v;

S[v]=1;

dist[v]=0;

for(i=2;i<=n-1;i++)

{

k=-1;

min=31999;

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

{

if(dist[j]<min && S[j]!=1)

{

min=dist[j];

k=j;

}

}

if(cost[v][k]<=dist[k])

p=1;

path[++p]=k;

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

cout<<path[j];

cout <<"\n";

//cout <<k;
S[k]=1;
for(j=1;j<=n;j++)
if(cost[k][j]!=31999 && dist[j]>=dist[k]+cost[k][j] && S[j]!=1)
dist[j]=dist[k]+cost[k][j];
}
}
```

OUTPUT

```
 enter no of vertices 6
 enter no of edges 11
enter EDGE Cost
 1 2 50
 1 3 45
 1 4 10
 2 3 10
 2 4 15
 3 5 30
 4 1 10
 4 5 15
 5 2 20
5 3 35
 6 5 3
 enter initial vertex 1
  1
 1 4
 1 4 5
 1 4 5 2
1 4 5 2 3
```

Experiment No.7

```
#include<iostream>
using namespace std;
#define INT_MAX 999999
int n=4;
int dist[10][10] = {
 {0,20,42,25},
 {20,0,30,34},
 {42,30,0,10},
 {25,34,10,0}
};
int VISITED_ALL = (1<<n) -1;
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];
}
//Now from current node, we will try to go to every other node and take the min ans
int ans = INT_MAX;
//Visit all the unvisited cities and take the best route
for(int city=0;city<n;city++){
if((mask&(1<<city))==0){
int newAns = dist[pos][city] + tsp( mask|(1<<city), city);
ans = min(ans, newAns);
}
}
return dp[mask][pos] = ans;
}
int main(){
 /* init the dp array */
 for(int i=0;i<(1<<n);i++){
 for(int j=0;j<n;j++){
 dp[i][j] = -1;
 }
 }
cout<<"Travelling Salesman Distance is "<<tsp(1,0);
return 0;
}
```

OUTPUT:

```
Travelling Salesman Distance is 85
Process returned 0 (0x0)   execution time : 0.030 s
```

Experiment No.8

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#define N 8
using namespace std;
/* print solution */
void printSolution(int board[N][N])
{
 for (int i = 0; i < N; i++)
 {
 for (int j = 0; j < N; j++)
 cout<<board[i][j]<<" ";
 cout<<endl;
 }
}
/* check if a queen can be placed on board[row][col]*/
bool isSafe(int board[N][N], int row, int col)
{
 int i, j;
 for (i = 0; i < col; i++)
 {
 if (board[row][i])
 return false;
 }
 for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
 {
 if (board[i][j])
 return false;
 }
 for (i = row, j = col; j >= 0 && i < N; i++, j--)
 {
 if (board[i][j])
 return false;
 }
 return true;
}
/*solve N Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
 if (col >= N)
 return true;
 for (int i = 0; i < N; i++)
 {
 if ( isSafe(board, i, col) )
 {
 board[i][col] = 1;
 if (solveNQUtil(board, col + 1) == true)
 return true;
 board[i][col] = 0;
 }
 }
 return false;
}
/* solves the N Queen problem using Backtracking.*/
bool solveNQ()
{
 int board[N][N] = {0};
 if (solveNQUtil(board, 0) == false)
 {
 cout<<"Solution does not exist"<<endl;
 return false;
 }
 printSolution(board);
 return true;
}
// Main
int main()
{
 solveNQ();
 return 0;
}
```

OUTPUT:

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

Process returned 0 (0x0)   execution time : 0.197 s
```

Experiment No.9

```
#include<iostream>
using namespace std;
int main()
{
int i,j,k,n,min,g[20][20],c[20][20],s,s1[20][1],s2,lb;
cout << ("\n TRAVELLING SALESMAN PROBLEM");
cout << ("\n Input number of cities:");
cin >> n;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
c[i][j]=0;
}}
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(i==j)
continue;
else{
cout<<"input"<<i<<"to"<<j<<"cost:";
cin>>c[i][j];
}
}
}
for(i=2;i<=n;i++)
{
g[i][0]=c[i][1];
```

```
}
for(i=2;i<=n;i++)
{
for(j=2;j<=n;j++)
{
if(i!=j)
g[i][j]=c[i][j]+g[j][0];
}
}
for(i=2;i<=n;i++)
{
for(j=2;j<=n;j++)
{
if(i!=j)
break;
}
}
for(k=2;k<=n;k++){
if(i!=k && j!=k){
if((c[i][j]+g[i][k])<(c[i][k]+g[k][j]))
{
g[i][j]=c[i][j]+g[j][k];
s1[i][j]=j;
}
else
{
g[i][1]=c[i][k]+g[k][j];
s1[i][1]=k;
}
```

```
}

}

min=c[1][2]+g[2][1];

s=2;

for(i=3;i<n;i++)

{

if((c[i][i]+g[i][i])<min)

{

min=c[1][i]+g[i][1];

s=i;

}

}

int y=g[i][1]+g[i][j]+g[i][i];

lb=(y/2);

cout<<"Edge Matrix";

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

{

cout<<"\n";

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

{

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

}

}

cout<<"\n min"<<min;

cout<<"\n\b"<<lb;

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

{

if(s!=i && s1[s][1]!=i)

{
```

```
s2=i;

}

}

cout<<"\n"<<1<<"-->"<<s<<"-->"<<s1[s][1]<<"-->"<<s2<<"-->"<<1<<"\n";

return (0);

}
```

OUTPUT:

```
 TRAVELLING SALESMAN PROBLEM

 Input number of cities:3

input1to2cost:12

input1to3cost:11

input2to1cost:01

input2to3cost:35

input3to1cost:25

input3to2cost:12

Edge Matrix

       0      12     11

       1      0      35

       25     12     0

 min12

6

1-->2-->0-->3-->1


Process returned 0 (0x0)   execution time : 12.237 s
```