NAME: Aditya Somani

PRN: 71901204L ROLL:

T1851061

SL-5(Group'B')

Assignment No. 6

AIM: Write a program to solve the travelling salesman problem and to print the path and the cost using Branch and Bound.

OBJECTIVE:

1. To understand the concept of Branch and Bound.

2. To understand the concept of travelling salesman problem.

3. To solve the travelling salesman problem and to print the path and the cost.

THEORY:

Branch and Bound :

A counter-part of the backtracking search algorithm which, in the absence of a cost criteria, the algorithm traverses a spanning tree of the solution space using the breadth-first approach. That is, a queue is used, and the nodes are processed in first-in-first-out order. If a cost criteria is available, the node to be expanded next (i.e., the branch) is the one with the best cost within the queue. In such a case, the cost function may also be used to discard (i.e., the bound) from the queue nodes that can be determined to be expensive. A priority queue is needed here.

• Branch and bound is a systematic method for solving optimization problems

• Branch and bound is a rather general optimization technique that applies where the

greedy method and dynamic programming fail.

• However, it is much slower. Indeed, it often leads to exponential time complexities in the worst case.

• On the other hand, if applied carefully, it can lead to algorithms that run reasonably fast on average.

• The general idea of Branch and bound is a BFS-like search for the optimal solution, but not all nodes get expanded (i.e., their children generated). Rather, a carefully selected criterion determines which node to expand and when, and another criterion tells the algorithm when an optimal solution has been found.

Algorithm:

```
function CheckBounds(st,des,cost[n][n]) // Cal. the bounds
//Global variable: cost[N][N] - the cost assignment.
pencost[0] = t
for i ← 0, n − 1 do
for j ← 0, n − 1 do
reduced[i][j] = cost[i][j]
end for
end for
for j ← 0, n − 1 do

reduced[st][j] = ∞
end for
for i ← 0, n − 1 do
reduced[i][des] = ∞
end for
```

```
reduced[des][st] = ∞
RowReduction(reduced)
ColumnReduction(reduced)
pencost[des] = pencost[st] + row + col + cost[st][des]
return pencost[des]
end function
function RowMin(cost[n][n],i) . Cal. min in the row
min = cost[i][0]
for j ← 0, n − 1 do
if cost[i][j] < min then
min = cost[i][j]
end if
end for
return min
end function
function ColMin(cost[n][n],i) . Cal. min in the col
min = cost[0][j]
for i ← 0, n − 1 do
if cost[i][j] < min then
min = cost[i][j]
end if
end for
return min
end function
function Rowreduction(cost[n][n]) . makes row reduction
```

```
row = 0
for i ← 0, n − 1 do
rmin = rowmin(cost, i)
if rmin 6= ∞ then
row = row + rmin
end if
for j ← 0, n − 1 do
if cost[i][j] 6= ∞ then
cost[i][j] = cost[i][j] – rmin
end if
end for
end for
end function
function Columnreduction(cost[n][n]) . makes column reduction
col = 0
for j ← 0, n − 1 do
cmin = columnmin(cost, j)
if cmin 6= ∞ then
col = col + cmin
end if

for i ← 0, n − 1 do
if cost[i][j] 6= ∞ then
cost[i][j] = cost[i][j] – cmin
end if
```

```
end for
end for
end function
function Main // main function
for i ← 0, n − 1 do
select[i] = 0
end for
rowreduction(cost)
columnreduction(cost)
t = row + col
while allvisited(select) 6= 1 do
for i ← 1, n − 1 do
if select[i] = 0 then
edgecost[i] = checkbounds(k, i, cost)
end if
end for
min = ∞
for i ← 1, n − 1 do
if select[i] = 0 then
if edgecost[i] < min then
min = edgecost[i] k = i
end if
end if
end for
select[k] = 1
```

for p ← 1, n − 1 do

cost[j][p] = ∞

end for

for p ← 1, n − 1 do

cost[p][k] = ∞

end for

cost[k][j] = ∞

rowreduction(cost)

columnreduction(cost)

end while

end function

Time Complexity of travelling salesman problem:

The complexity of the algorithm is O(n3

ln(n))

Conclusion:

We successfully solved the travelling salesman problem and printed the path and the cost using Branch and Bound.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int parent_cost,r,no_of_nodes;
int c[10][10],temp[10][10],visited[10],path[10];
void disp(int x[10][10]);
```

```
void copy(int x[10][10],int y[10][10]);
void cal_cost(int set[],int n);
int check_reduced();
int flag=1;
int main()
{
int i,j,set[10];
printf("\n Enter the no. of nodes in the graph:");
scanf("%d",&no_of_nodes);
for(i=1;i<=no_of_nodes;i++)
{
for(j=1;j<=no_of_nodes;j++)
{
if(i!=j)
{
printf("\n Enter the cost of edge between");
printf("%d-%d:",i,j);
scanf("%d", &c[i][j]);
}
else
c[i][j]=99;
}
}
visited[1]=1;
path[r]=1;
```

```
r++;


disp(c);
copy(temp,c);
int root_cost=check_reduced();
parent_cost=root_cost;
copy(c,temp);
printf("\n Reduced cost matrix:\n");
disp(c);
int p=0;
for(i=1;i<=no_of_nodes;i++)
{
if(visited[i]!=1)
{
set[p]=i;
p++;
}
}
cal_cost(set,p);
printf("\n Minimum cost: %d",parent_cost);
printf("\n Resultant Path:");
for(i=0;i<r;i++)
printf("%d->", path[i]);
printf("1");
return 0;
```

```
}
void disp(int x[10][10])
{
for(int i=1;i<=no_of_nodes;i++)
{
printf("\t");
for(int j=1;j<=no_of_nodes;j++)
printf("%d \t", x[i][j]);
printf("\n");

}
}
void copy(int x[10][10],int y[10][10])
{
for(int i=1;i<=no_of_nodes;i++)
{
for(int j=1;j<=no_of_nodes;j++)
{
x[i][j]=y[i][j];
}
}
}
int find_min(int g[10],int n,int q[10])
{
int min=g[0];
```

```
int node=q[0];
for(int i=1;i<n;i++)
{
if(min>g[i])
{
min=g[i];
node=q[i];
}
}
path[r]=node;
r++;
visited[node]=1;
return(min);
}
int check_reduced()
{
int min,sum=0;

for(int i=1;i<=no_of_nodes;i++)
{
min=999;
for(int j=1;j<=no_of_nodes;j++)
{
if(temp[i][j]<min)
min=temp[i][j];
```

```
}
if(min!=0 && min!=99)
{
for(int j=1;j<=no_of_nodes;j++)
{
if(i!=j && temp[i][j]!=99)
temp[i][j]=temp[i][j]-min;
}
sum=sum+min;
}
}
for(int j=1;j<=no_of_nodes;j++)
{
min=999;
for(int i=1;i<=no_of_nodes;i++)
{
if(temp[i][j]<min)
min=temp[i][j];
}
if(min!=0 && min!=99)
{
for(int i=1;i<=no_of_nodes;i++)
{
if(j!=i && temp[i][j]!=99)
temp[i][j]=temp[i][j]-min;
```

```
}
sum=sum+min;
}
}
printf("\n\n");
disp(temp);
printf("\nSum:%d", sum);
return(sum);
}
void cal_cost(int set[],int n)
{
int g[10],q[10];
for(int i=0;i<n;i++)
{
copy(temp,c);
for(int j=0;j<r;j++)
{
for(int k=1;k<=no_of_nodes;k++)
{
temp[path[j]][k]=99;
if(j!=0)
temp[k][path[j]]=99;
}
}
```

```
for(int k=1;k<=no_of_nodes;k++)
temp[k][set[i]]=99;
temp[set[i]][1]=99;//column
int ans=check_reduced();
g[i]=parent_cost+ans+c[path[r-1]][set[i]];
printf("\n Cost: %d", g[i]);
q[i]=set[i];
}

parent_cost=find_min(g,n,q);
int p=0;
for(int i=1;i<=no_of_nodes;i++)
{
if(visited[i]!=1)
{
set[p]=i;
p++;
}
}
if(p!=0)
cal_cost(set,p);
}
```

OUTPUT:

main.cpp

Run

```
105  min=temp[i][j];
106  }
107  if(min!=0 && min!=99)
108  {
109  for(int j=1;j<=no_of_nodes;j++)
110  {
111  if(i!=j && temp[i][j]!=99)
112  temp[i][j]=temp[i][j]-min;
113  }
114  sum=sum+min;
115  }
116  }
117  for(int j=1;j<=no_of_nodes;j++)
118  {
119  min=999;
120  for(int i=1;i<=no_of_nodes;i++)
121  {
122  if(temp[i][j]<min)
123  min=temp[i][j];
124  }
125  if(min!=0 && min!=99)
126  {
127  for(int i=1;i<=no_of_nodes;i++)
```

Output

Clear

```
/tmp/LOYtn7oilB.o
Enter the no. of nodes in the graph:4

 Enter the cost of edge between1-2:4
 Enter the cost of edge between1-3:3
 Enter the cost of edge between1-4:5
 Enter the cost of edge between2-1:5
 Enter the cost of edge between2-3:3
 Enter the cost of edge between2-4:4

 Enter the cost of edge between3-1:3

 Enter the cost of edge between3-2:4

 Enter the cost of edge between3-4:2

 Enter the cost of edge between4-1:3

 Enter the cost of edge between4-2:2

 Enter the cost of edge between4-3:3
 99     4   3   5
    5   99  3   4
```

main.cpp

Run

```
105  min=temp[i][j];
106  }
107  if(min!=0 && min!=99)
108  {
109  for(int j=1;j<=no_of_nodes;j++)
110  {
111  if(i!=j && temp[i][j]!=99)
112  temp[i][j]=temp[i][j]-min;
113  }
114  sum=sum+min;
115  }
116  }
117  for(int j=1;j<=no_of_nodes;j++)
118  {
119  min=999;
120  for(int i=1;i<=no_of_nodes;i++)
121  {
122  if(temp[i][j]<min)
123  min=temp[i][j];
124  }
125  if(min!=0 && min!=99)
126  {
127  for(int i=1;i<=no_of_nodes;i++)
```

Output

Clear

```
 Enter the cost of edge between4-3:3
 99     4   3   5
    5   99  3   4
    3   4   99  2
    3   2   3   99


    99  1   0   2
    1   99  0   1
    0   2   99  0
    0   0   1   99

Sum:11
 Reduced cost matrix:
    99  1   0   2
    1   99  0   1
    0   2   99  0
    0   0   1   99


    99  99  99  99
    99  99  0   1
    0   99  99  0
```

main.cpp

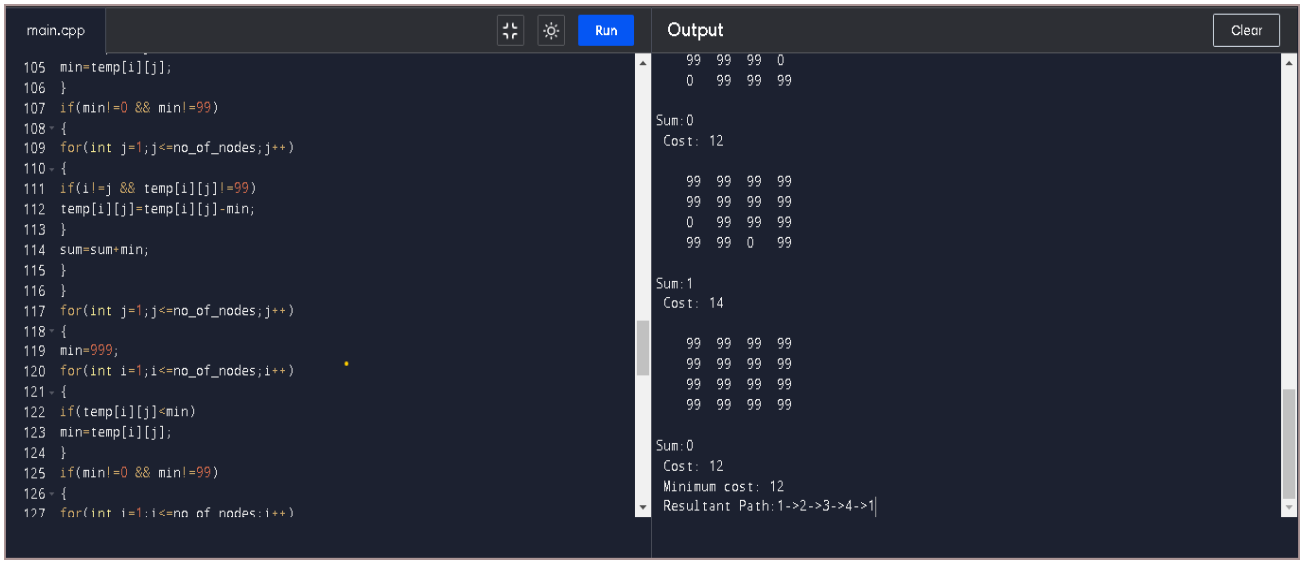
Run

```
105  min=temp[i][j];
106  }
107  if(min!=0 && min!=99)
108  {
109  for(int j=1;j<=no_of_nodes;j++)
110  {
111  if(i!=j && temp[i][j]!=99)
112  temp[i][j]=temp[i][j]-min;
113  }
114  sum=sum+min;
115  }
116  }
117  for(int j=1;j<=no_of_nodes;j++)
118  {
119  min=999;
120  for(int i=1;i<=no_of_nodes;i++)
121  {
122  if(temp[i][j]<min)
123  min=temp[i][j];
124  }
125  if(min!=0 && min!=99)
126  {
127  for(int i=1;i<=no_of_nodes;i++)
```

Output

Clear

```
Sum:0
 Cost: 12

    99  99  99  99
    0   99  99  0
    99  2   99  0
    0   0   99  99

Sum:1
 Cost: 12

    99  99  99  99
    1   99  0   99
    0   2   99  99
    99  0   1   99

Sum:0
 Cost: 13

    99  99  99  99
    99  99  99  99
    99  99  99  0
    0   99  99  99
```


main.cpp
Run
Output
Clear
min=temp[i][j];
}
if(min!=0 && min!=99)
{
for(int j=1;j<=no_of_nodes;j++)
{
if(i!=j && temp[i][j]!=99)
temp[i][j]=temp[i][j]-min;
}
sum=sum+min;
}
}
for(int j=1;j<=no_of_nodes;j++)
{
min=999;
for(int i=1;i<=no_of_nodes;i++)
{
if(temp[i][j]<min)
min=temp[i][j];
}
if(min!=0 && min!=99)
{
99 99 99 0
0 99 99 99
Sum:0
Cost: 12
99 99 99 99
99 99 99 99
0 99 99 99
99 99 0 99
Sum:1
Cost: 14
99 99 99 99
99 99 99 99
99 99 99 99
99 99 99 99
Sum:0
Cost: 12
Minimum cost: 12
Resultant Path:1->2->3->4->1