**DAA ASSIGNMENT-5**

****

**SUBMITTED TO:**

**PROF. SUBHASH TATALE**

**VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE**

**COMPUTER ENGINEERING DEPARTMENT**

**BY:**

**NAME: ABHISHEK MORE**

**G.R NO.: 21810033**

**ROLL NO.: 323036**

**CLASS: T.Y COMP**

**BATCH: COMP C2**

**Assignment-5**

**Aim:**

Solve the given problem using branch and bound technique.

**Objective:**

Our objective is to find solution for Travelling salesman problem using Branch and Bound.

**Theory:**

**Branch and bound** is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case.

**Source code:**

**package** assignment5\_TSP;

**import** java.util.\*;

**class** TSP\_BB

{

**static** **int** *N* = 4;

**static** **int** *final\_path*[] = **new** **int**[*N* + 1];

**static** **boolean** *visited*[] = **new** **boolean**[*N*];

**static** **int** *final\_res* = Integer.***MAX\_VALUE***;

**static** **void** copyToFinal(**int** curr\_path[])

{

**for** (**int** i = 0; i < *N*; i++)

{

*final\_path*[i] = curr\_path[i];

}

*final\_path*[*N*] = curr\_path[0];

}

**static** **int** firstMin(**int** adj[][], **int** i)

{

**int** min = Integer.***MAX\_VALUE***;

**for** (**int** k = 0; k < *N*; k++)

{

**if** (adj[i][k] < min && i != k)

min = adj[i][k];

}

**return** min;

}

**static** **int** secondMin(**int** adj[][], **int** i)

{

**int** first = Integer.***MAX\_VALUE***, second = Integer.***MAX\_VALUE***;

**for** (**int** j=0; j<*N*; j++)

{

**if** (i == j)

**continue**;

**if** (adj[i][j] <= first)

{

second = first;

first = adj[i][j];

}

**else** **if** (adj[i][j] <= second && adj[i][j] != first)

second = adj[i][j];

}

**return** second;

}

**static** **void** TSPRec(**int** adj[][], **int** curr\_bound, **int** curr\_weight,

**int** level, **int** curr\_path[])

{

**if** (level == *N*)

{

**if** (adj[curr\_path[level - 1]][curr\_path[0]] != 0)

{

**int** curr\_res = curr\_weight + adj[curr\_path[level-1]][curr\_path[0]];

**if** (curr\_res < *final\_res*)

{

*copyToFinal*(curr\_path);

*final\_res* = curr\_res;

}

}

**return**;

}

**for** (**int** i = 0; i < *N*; i++)

{

**if** (adj[curr\_path[level-1]][i] != 0 &&

*visited*[i] == **false**)

{

**int** temp = curr\_bound;

curr\_weight += adj[curr\_path[level - 1]][i];

**if** (level==1)

curr\_bound -= ((*firstMin*(adj, curr\_path[level - 1]) + *firstMin*(adj, i))/2);

**else**

curr\_bound -= ((*secondMin*(adj, curr\_path[level - 1]) + *firstMin*(adj, i))/2);

**if** (curr\_bound + curr\_weight < *final\_res*)

{

curr\_path[level] = i;

*visited*[i] = **true**;

*TSPRec*(adj, curr\_bound, curr\_weight, level + 1, curr\_path);

}

curr\_weight -= adj[curr\_path[level-1]][i];

curr\_bound = temp;

Arrays.*fill*(*visited*,**false**);

**for** (**int** j = 0; j <= level - 1; j++)

*visited*[curr\_path[j]] = **true**;

}

}

}

**static** **void** TSP(**int** adj[][])

{

**int** curr\_path[] = **new** **int**[*N* + 1];

**int** curr\_bound = 0;

Arrays.*fill*(curr\_path, -1);

Arrays.*fill*(*visited*, **false**);

**for** (**int** i = 0; i < *N*; i++)

curr\_bound += (*firstMin*(adj, i) + *secondMin*(adj, i));

curr\_bound = (curr\_bound==1)? curr\_bound/2 + 1 : curr\_bound/2;

*visited*[0] = **true**;

curr\_path[0] = 0;

*TSPRec*(adj, curr\_bound, 0, 1, curr\_path);

}

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args)

{

Scanner sc=**new** Scanner(System.***in***);

**int** adj[][] = **new** **int**[4][4];

System.***out***.println("Enter adjancency matrix");

**for**(**int** i=0; i<4; i++)

{

**for**(**int** j=0; j<4; j++)

{

adj[i][j]=sc.nextInt();

}

}

*TSP*(adj);

System.***out***.printf("Minimum cost : %d\n", *final\_res*);

System.***out***.printf("Path Taken : ");

**for** (**int** i = 0; i <= *N*; i++)

{

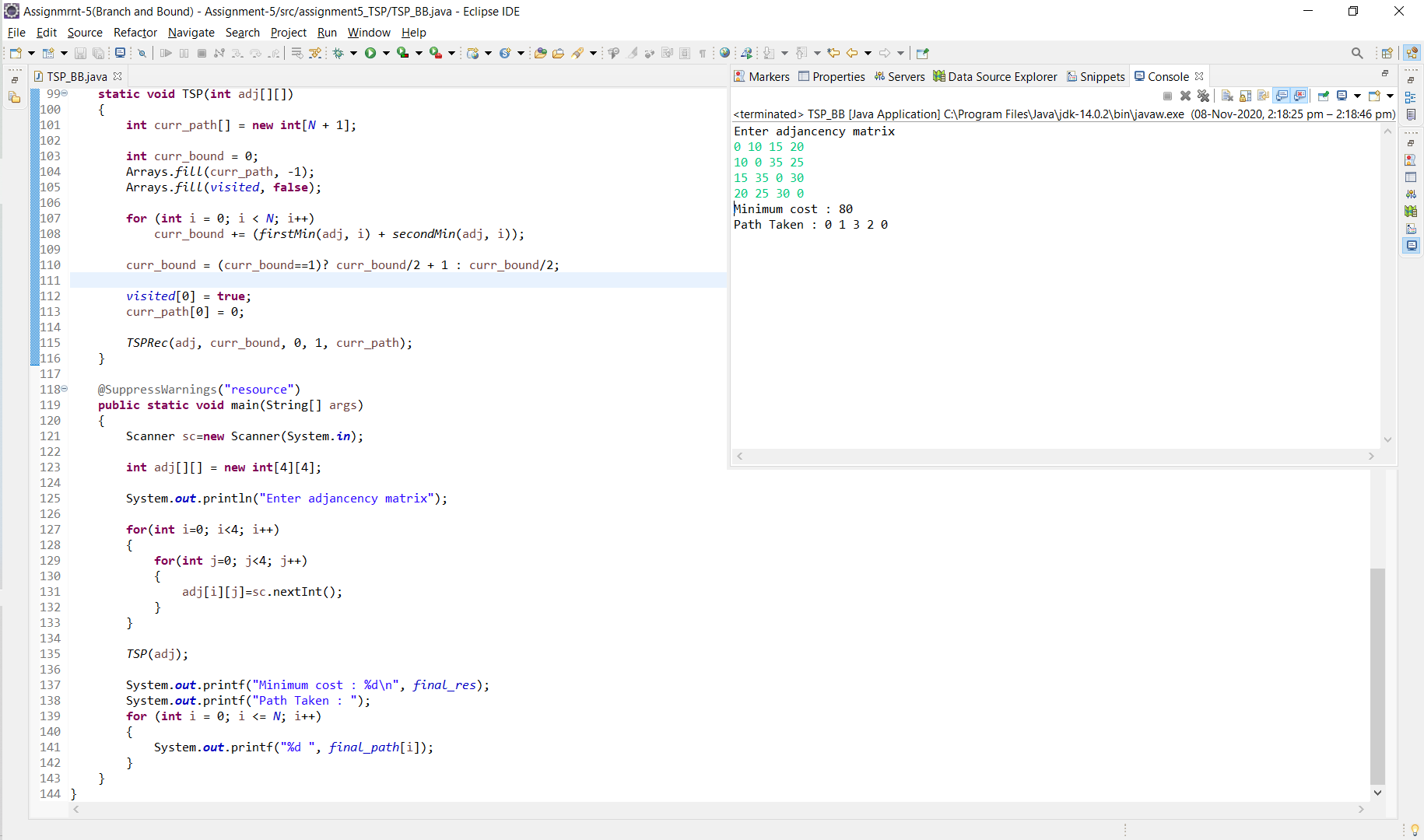
System.***out***.printf("%d ", *final\_path*[i]);

}

}

}

**Output:**

****

**Conclusion:**

Successfully found the solution for Travelling salesman problem using branch and bound approach approach.