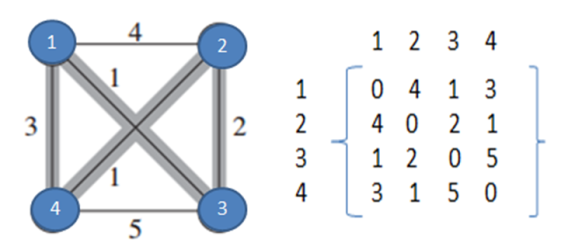
**Travelling Salesman Problem (TSP) Using Dynamic Programming**

**Example Problem**



Above we can see a complete directed graph and cost matrix which includes distance between each village. We can observe that cost matrix is symmetric that means distance between village 2 to 3 is same as distance between village 3 to 2.

Here problem is travelling salesman wants to find out his tour with minimum cost.

Say it is T (1,{2,3,4}), means, initially he is at village 1 and then he can go to any of {2,3,4}. From there to reach non-visited vertices (villages) becomes a new problem. Here we can observe that main problem spitted into sub-problem, this is property of dynamic programming.

**Note:** While calculating below right side values calculated in bottom-up manner. Red color values taken from below calculations.

T ( 1, {2,3,4} ) = minimum of

= { (1,2) + T (2,  {3,4} )     4+6=10

= { (1,3)  + T (3, {2,4} )     1+3=4

= { (1,4) + T (4, {2,3} )     3+3=6

Here minimum of above 3 paths is answer but we know only values of (1,2) , (1,3) , (1,4) remaining thing which is T ( 2, {3,4} ) …are new problems now. First we have to solve those and substitute here.

T (2, {3,4} )   = minimum of

=  { (2,3) + T (3, {4} )     2+5=7

= { (2,4) + T {4, {3} )     1+5=6

T (3, {2,4} )   = minimum of

=  { (3,2) + T (2, {4} )     2+1=3

= { (3,4) + T {4, {2} )     5+1=6

T (4, {2,3} )   = minimum of

=  { (4,2) + T (2, {3} )     1+2=3

= { (4,3) + T {3, {2} )     5+2=7

T ( 3, {4} ) =  (3,4) + T (4, {} )     5+0=5

T ( 4, {3} ) =  (4,3) + T (3, {} )     5+0=5

T ( 2, {4} ) =  (2,4) + T (4, {} )     1+0=1

T ( 4, {2} ) =  (4,2) + T (2, {} )     1+0 = 1

T ( 2, {3} ) =  (2,3) + T (3, {} )     2+0 = 2

T ( 3, {2} ) =  (3,2) + T (2, {} )     2+0=2

Here T ( 4, {} ) is reaching base condition in recursion, which returns 0 (zero ) distance.

This is where we can find final answer,

T ( 1, {2,3,4} ) = minimum of

= { (1,2) + T (2,  {3,4} )     4+6=10 in this path we have to add +1 because this path ends with 3. From there we have to reach 1 so 3->1 distance 1 will be added total distance is 10+1=11

= { (1,3)  + T (3, {2,4} )     1+3=4 in this path we have to add +3 because this path ends with 3. From there we have to reach 1 so 4->1 distance 3 will be added total distance is 4+3=7

= { (1,4) + T (4, {2,3} )     3+3=6 in this path we have to add +1 because this path ends with 3. From there we have to reach 1 so 3->1 distance 1 will be added total distance is 6+1=7

### Time Complexity

Since we are solving this using Dynamic Programming, we know that Dynamic Programming approach contains sub-problems.

Here after reaching ith node finding remaining minimum distance to that ith node is a sub-problem.

If we solve recursive equation we will get total **(n-1) 2(n-2)**sub-problems, which is**O (n2n)**.

Each sub-problem will take **O (n)** time (finding path to remaining **(n-1)** nodes).

Therefore total time complexity is **O (n2n) \* O (n) = O (n22n)**

Space complexity is also number of sub-problems which is **O (n2n)**

### Program for Travelling Salesman Problem in C++

#include<iostream>

using namespace std;

int ary[10][10],completed[10],n,cost=0;

void takeInput()

{

int i,j;

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

cin>>n;

cout<<"\nEnter the Cost Matrix\n";

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

{

cout<<"\nEnter Elements of Row: "<<i+1<<"\n";

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

cin>>ary[i][j];

completed[i]=0;

}

cout<<"\n\nThe cost list is:";

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

{

cout<<"\n";

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

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

}

}

int least(int c)

{

int i,nc=999;

int min=999,kmin;

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

{

if((ary[c][i]!=0)&&(completed[i]==0))

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

{

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

nc=i;

}

}

if(min!=999)

cost+=kmin;

return nc;

}

void mincost(int city)

{

int i,ncity;

completed[city]=1;

cout<<city+1<<"--->";

ncity=least(city);

if(ncity==999)

{

ncity=0;

cout<<ncity+1;

cost+=ary[city][ncity];

return;

}

mincost(ncity);

}

int main()

{

takeInput();

cout<<"\n\nThe Path is:\n";

mincost(0); //passing 0 because starting vertex

cout<<"\n\nMinimum cost is "<<cost;

return 0;

}

**Output**

Enter the number of villages: 4

Enter the Cost Matrix

Enter Elements of Row: 1  
0 4 1 3

Enter Elements of Row: 2  
4 0 2 1

Enter Elements of Row: 3  
1 2 0 5

Enter Elements of Row: 4  
3 1 5 0  
The cost list is:  
0 4 1 3  
4 0 2 1  
1 2 0 5  
3 1 5 0

The Path is:  
1—>3—>2—>4—>1

Minimum cost is 7