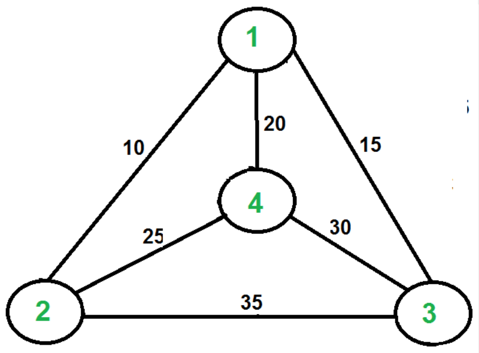
Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)

Travelling Salesman Problem (TSP): Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.
Note the difference between Hamiltonian Cycle and TSP. The Hamiltoninan cycle problem is to find if there exist a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

For example, consider the graph shown in figure on right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.



The problem is a famous NP hard problem. There is no polynomial time know solution for this problem.

Following are different solutions for the traveling salesman problem.

Naive Solution:
1) Consider city 1 as the starting and ending point.
2) Generate all (n-1)! Permutations of cities.
3) Calculate cost of every permutation and keep track of minimum cost permutation.
4) Return the permutation with minimum cost.

Time Complexity: ?(n!)

Dynamic Programming:
Let the given set of vertices be {1, 2, 3, 4,….n}. Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once. Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far. Now the question is how to get cost(i)?
To calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term *C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i*.
We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

```
If size of S is 2, then S must be {1, i},
 C(S, i) = dist(1, i)
Else if size of S is greater than 2.
 C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.
```

For a set of size n, we consider n-2 subsets each of size n-1 such that all subsets don't have nth in them.
Using the above recurrence relation, we can write dynamic programming based solution. There are at most $O(n*2^n)$ subproblems, and each one takes linear time to solve. The total running time is therefore $O(n^2*2^n)$. The time complexity is much less than O(n!), but still exponential. Space required is also exponential. So this approach is also infeasible even for slightly higher number of vertices.

We will soon be discussing approximate algorithms for travelling salesman problem.

Next Article: Traveling Salesman Problem | Set 2

References:

http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf
http://www.cs.berkeley.edu/~vazirani/algorithms/chap6.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

22 Comments **GeeksforGeeks** **Tejas Joshi**

Recommend 4 **Share** **Sort by Newest**

Join the discussion…

AllergicToBitches • 3 months ago

Solution of Travelling Salesman Problem using Branch and Bound - http://ideone.com/0TBgxr

References:
https://www.seas.gwu.edu/~bell...
http://research.ijcaonline.org...

1 • Reply • Share ›

Nagadeep • 4 months ago

First link in references is not working!

Reply • Share ›

Jayesh Saita • 5 months ago

Hey,
I am confused with the complexity part.
You say that 0(n^2 * 2^n) is less than 0(n!).
Consider the no. of vertices as 5,
5^2 * 2^5 = 800 whereas 5! is 120. There is a huge difference.
Can you explain me this part.
Thank you :)

Reply • Share ›

Anirudh → Jayesh Saita • 4 months ago

Complexity means Asymptotic complexity.
Don't take such small values like n=5,
n should be large, like n=100;
Now calculate both the complexity and you'll find that n! is greater than that time complexity.
https://en.wikipedia.org/wiki/...

1 • Reply • Share ›

Kriti • 5 months ago

Really helpful, great article. For more on travelling - http://www.tripreviewsite.com/...

Reply • Share ›

sumit prakash • a year ago

Go thru dis link.. a layman way for TSP

http://www.mbaskool.com/busine...

• Reply • Share ›

Billionaire • a year ago

Good read

• Reply • Share ›

xyz • 2 years ago

please share the implementation

• Reply • Share ›

sumit prakash xyz • a year ago

You can see it here

http://www.mbaskool.com/busine...

• Reply • Share ›

helper xyz • 2 years ago

we implemented it in our 2nd year. the basic idea is for reaching i to j via a subset S, there MUST BE A LAST VISITED CITY IN S.
To represent a set, we used binary numbers:
let a={s0,s1,s2,s3,s4,s5}, then its subset x={s0,s1,s4} will be represented as 010011, i.e. starting from the rightmost point, ith bit represents whether that subset contains s(i). To add s(j) to a subset and it with 1<<j.......... removal="" etc.="" can="" also="" be="" managed="" by="" all="" those="" bitwise="" operations...="" :)="">

• Reply • Share ›

rohan • 2 years ago

http://ideone.com/tYdrK8

It is well commented and contains the recursion tree for an example input graph :) enjoy

Using the DP equation-

g(i,S)= min(Cij + g(j, S-{ j }) where j belongs to S.

where g(i,S) is length of a shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1.

8 • Reply • Share ›

NB • 2 years ago

Why is the Time Complexity: big omega (n!) ? This should be an upper bound, not lower bound right

2 • Reply • Share ›

john • 2 years ago

and that dp-derived path (preferably, as it pertains to the top-down sol'n)?

• Reply • Share ›

Lohith Ravi • 3 years ago

Hi Moderators,
I really like geeks for geeks for the service it renders by spreading so much of knowledge.
I also want you guys to make a point some where that, Dynamic programming is just a method where we cache the result so that it need not be computer once more and a HashMap could be of great help as a simple cache.

// for the above problem, I have a DP code writtern in Java, I wish you use it in the above post so that it helps people

```
import java.util.ArrayList;

import java.util.HashMap;

public class TravelingSalesManProblem {

public static HashMap<string,integer> DynamicProgrammingCache = new
HashMap<string,integer>();

public static void main(String[] args) {
```

see more

1 • Reply • Share ›

j Lohith Ravi • 2 years ago

How to track the path and get the shortest path?

• Reply • Share ›

Guest • 3 years ago

```
import java.util.ArrayList;

public class TravelingSalesManProblem {

public static void main(String[] args) {

int source = 0;

int matrix[][] = { { 0, 10, 15, 20 }, { 10, 0, 35, 25 },

{ 15, 35, 0, 30 }, { 20, 25, 30, 0 } };

ArrayList<integer> visitedNodes = new ArrayList<integer>();

visitedNodes.add(source);
```

int minDistance = findMinimumDistance(matrix, source, visitedNodes);

System.out.println(minDistance);

}

see more

1 ^ | ˅ • Reply • Share ›

Sriharsha g.r.v • 3 years ago

hello sir, i am naive to this concept, but i would like to know if disktras or bellmann algo could be applied to very vertex one after th other and discard that vertex from graph..in that way dont we get the shortest tour possible?

2 ^ | ˅ • Reply • Share ›

mysticPrince ➦ Sriharsha g.r.v • 3 years ago

Look at this recurrence:

C(S, i) = min { C(S-{i}, j) + dis(j, i)}
Doesn't it look similar to relaxation property in Bellman ford's?

Why not the same complexity then?

1. Cause there are two constraints in particular:
a)Each vertex must be visited, and only once (except 1 where tour starts and ends)

b) For nth vertex, in tour (from 1st), we are looking for a n-length path subject to above two constraints.

2. Cause it NP Complete problem :p incorporating above two constraints

Well, I'm not the author of the article. But maybe it helps. :)

4 ^ | ˅ • Reply • Share ›

Kaushik Srinivasan • 3 years ago

Can someone please explain the 2nd recurrence relation?

1 ^ | ˅ • Reply • Share ›

Bharathkumar Hegde ➦ Kaushik Srinivasan • 3 years ago

I think if subset is greater than two, then

```
loop for each j in S-{i}:
  required = min ( required , C (S-{i}, j) + dist ( i , j ) );
```

now 'required' has minimum cost of travelling through S - { i } considering some city ' j ' in S - { i } is last node plus the dist from ' j ' (possible last city) to i.

^ | ˅ • Reply • Share ›

mysticPrince ➜ Kaushik Srinivasan • 3 years ago

C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1
This one I suppose.
As the author has stated:
C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.

For each subset size>2,
C(S,i) is given as minimum of all path lengths visiting every vertex in subset S exactly once, starting at 1, ending at i (i belongs to S) , with j as intermediate vertex, for all j in S (j!=i, j!=1), for all i in S (i!=1).
So the expressions is of form: C(S-{i}, j) +dist (j,i)
I hope it helps :)

• Reply • Share ›

Shubham Mittal • 3 years ago

For each vertex, we have 2^(n-1) subsets of the set excluding 1. Therefore, the total number of subproblems will be n*2^(n-1). Since, each problem takes linear time to solve, the complexity is O(n^2*2^(n-1)) which is equivalent to O(n^2*2^n).

Please correct me if I am wrong.

26 • Reply • Share ›