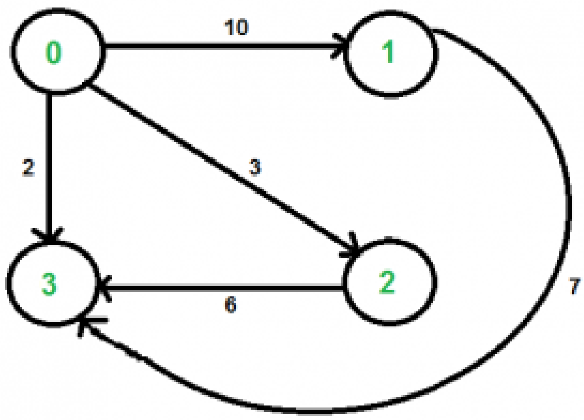
Shortest path with exactly k edges in a directed and weighted graph

Given a directed and two vertices 'u' and 'v' in it, find shortest path from 'u' to 'v' with exactly k edges on the path.

The graph is given as adjacency matrix representation where value of graph[i][j] indicates the weight of an edge from vertex i to vertex j and a value INF(infinite) indicates no edge from i to j.

For example consider the following graph. Let source 'u' be vertex 0, destination 'v' be 3 and k be 2. There are two walks of length 2, the walks are {0, 2, 3} and {0, 1, 3}. The shortest among the two is {0, 2, 3} and weight of path is 3+6 = 9.



The idea is to browse through all paths of length k from u to v using the approach discussed in the previous post and return weight of the shortest path. A **simple solution** is to start from u, go to all adjacent vertices and recur for adjacent vertices with k as k-1, source as adjacent vertex and destination as v. Following is C++ implementation of this simple solution.

```
// C++ program to find shortest path with exactly k edges
#include <iostream>
#include <climits>
using namespace std;

// Define number of vertices in the graph and inifinite value
#define V 4
#define INF INT_MAX

// A naive recursive function to count walks from u to v with k edges
int shortestPath(int graph[][V], int u, int v, int k)
{
   // Base cases
   if (k == 0 && u == v)             return 0;
   if (k == 1 && graph[u][v] != INF) return graph[u][v];
   if (k <= 0)                       return INF;

   // Initialize result
   int res = INF;

   // Go to all adjacents of u and recur
   for (int i = 0; i < V; i++)
   {
       if (graph[u][i] != INF && u != i && v != i)
       {
           int rec_res = shortestPath(graph, i, v, k-1);
           if (rec_res != INF)
              res = min(res, graph[u][i] + rec_res);
       }
   }
   return res;
}

// driver program to test above function
int main()
```

```
{
    /* Let us create the graph shown in above diagram*/
     int graph[V][V] = { {0, 10, 3, 2},
                        {INF, 0, INF, 7},
                        {INF, INF, 0, 6},
                        {INF, INF, INF, 0}
                      };
    int u = 0, v = 3, k = 2;
    cout << "Weight of the shortest path is " <<
          shortestPath(graph, u, v, k);
    return 0;
}
```

Output:

```
Weight of the shortest path is 9
```

The worst case time complexity of the above function is $O(V^k)$ where V is the number of vertices in the given graph. We can simply analyze the time complexity by drawing recursion tree. The worst occurs for a complete graph. In worst case, every internal node of recursion tree would have exactly V children.
We can optimize the above solution using **Dynamic Programming**. The idea is to build a 3D table where first dimension is source, second dimension is destination, third dimension is number of edges from source to destination, and the value is count of walks. Like other Dynamic Programming problems, we fill the 3D table in bottom up manner.

```
// Dynamic Programming based C++ program to find shortest path with
// exactly k edges
#include <iostream>
#include <climits>
using namespace std;

// Define number of vertices in the graph and inifinite value
#define V 4
#define INF INT_MAX

// A Dynamic programming based function to find the shortest path from
// u to v with exactly k edges.
int shortestPath(int graph[][V], int u, int v, int k)
{
    // Table to be filled up using DP. The value sp[i][j][e] will store
    // weight of the shortest path from i to j with exactly k edges
    int sp[V][V][k+1];

    // Loop for number of edges from 0 to k
    for (int e = 0; e <= k; e++)
    {
        for (int i = 0; i < V; i++)  // for source
        {
            for (int j = 0; j < V; j++) // for destination
            {
                // initialize value
                sp[i][j][e] = INF;
```

```
                // from base cases
                if (e == 0 && i == j)
                    sp[i][j][e] = 0;
                if (e == 1 && graph[i][j] != INF)
                    sp[i][j][e] = graph[i][j];

                //go to adjacent only when number of edges is more than 1
                if (e > 1)
                {
                    for (int a = 0; a < V; a++)
                    {
                        // There should be an edge from i to a and a
                        // should not be same as either i or j
                        if (graph[i][a] != INF && i != a &&
                            j!= a && sp[a][j][e-1] != INF)
                          sp[i][j][e] = min(sp[i][j][e], graph[i][a] +
                                                         sp[a][j][e-1]);
                    }
                }
           }
        }
    }
    return sp[u][v][k];
}

// driver program to test above function
int main()
{
    /* Let us create the graph shown in above diagram*/
     int graph[V][V] = { {0, 10, 3, 2},
                        {INF, 0, INF, 7},
                        {INF, INF, 0, 6},
                        {INF, INF, INF, 0}
                      };
    int u = 0, v = 3, k = 2;
    cout << shortestPath(graph, u, v, k);
    return 0;
}
```

Output:

```
Weight of the shortest path is 9
```

Time complexity of the above DP based solution is $O(V^3K)$ which is much better than the naive solution.

This article is contributed by **Abhishek**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- [Assign directions to edges so that the directed graph remains acyclic](#)

- K Centers Problem | Set 1 (Greedy Approximate Algorithm)
- Find the minimum cost to reach destination using a train
- Applications of Breadth First Traversal
- Optimal read list for given number of days
- Print all paths from a given source to a destination
- Minimize Cash Flow among a given set of friends who have borrowed money from each other
- Boggle (Find all possible words in a board of characters)

Tags: Dynamic Programming

Tweet 1 g+1 1

Writing code in comment? Please use **ideone.com** and share the link here.

15 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Sumit Kesarwani • 3 months ago
https://gist.github.com/sumitd...

please look on this (sortestPath(......)) methos rest is just java beans
Reply • Share ›

Dheeraj Sachan • 4 months ago
Solved using DFS

https://github.com/dheeraj9198...
Reply • Share ›

Dharmendra Verma • 5 months ago
why are we taking 3D array??
destination is always same so is there any need to have a dimension for destination??
Reply • Share ›

juanvillegas → Dharmendra Verma • 2 months ago
of course, as the algorithm calculates every possible source->destination path to make the final dynamic solution. you could store the matrix cube and use it again for other source/destination..
Reply • Share ›

Kenneth • 5 months ago
My DP solution with time complexity O(KE) and space complexity O(KV):

http://ideone.com/GICYLO

1 • Reply • Share ›

d_k Kenneth • 4 months ago

http://www.geeksforgeeks.org/f...

plz help me for solving this problem

• Reply • Share ›

graph333 • 7 months ago

What would be the time complexity of the recursive solution if the graph were to be represented as an adjacency list instead of a matrix?

• Reply • Share ›

Pavel Podlipensky • 8 months ago

The problem statement says nothing about negative edges. Can negative edges be present in the graph? If not - then the algorithm could be improved by using augmented Dijkstra algo instead of Bellman-Ford. Here is python implementation

```
def aug_dijkstra(G, s, t, k):
    n = len(G)
    heap = [(0, -1, s)]
    w = [[sys.maxint for i in xrange(n)] for j in xrange(k)]
    while len(heap):
        weight, step, node = heapq.heappop(heap)
        if step+1 == k:
            continue
        for i in xrange(n):
            if G[node][i] > 0 and G[node][i] + weight < w[step+1][i]:
                w[step+1][i] = G[node][i] + weight
                heapq.heappush(heap, (w[step+1][i], step+1, i))

    return w[k-1][t]
```

4 • Reply • Share ›

Preethi • 8 months ago

Can any one explain the idea behind the DP approach

• Reply • Share ›

swati • 8 months ago

I have written a DP for above problem.

Following is the link http://ideone.com/GE6paQ

Please let me know if there is some error.

^ | v • Reply • Share ›

shiva ➦ swati • 5 months ago

It looks like u wrote recursive solution, which is given as naive solution above!!

^ | v • Reply • Share ›

wenchao • 8 months ago

can think of a O(nk) algo

1. from source node, solve problem (node, k-1) recursively for adjacent nodes.

2. in each recursion, record result of (node,k-i) in 2D array for reuse.

If you visualize the recursion tree. Let i denote the level of the tree. In each level of the tree, the problem of (node,k-i) can be solved only once and the result recorded in 2D array for reuse.

So in worst case, each level takes log(N), for the whole tree, it takes O(nk), as k is the depth of recursion tree.

^ | v • Reply • Share ›

Satya • 9 months ago

O(nk) algo possible with 2D dp matrix possible.

here is the ideone link for the code..http://ideone.com/2mzNYt

1 ^ | v • Reply • Share ›

Sidharth ➦ Satya • 8 months ago

plz share the ideone link for the source code and algorithm in comment rather than pasting unindented code in here...

^ | v • Reply • Share ›

Satya ➦ Sidharth • 8 months ago

hey sorry for bad code and late comment..

Actually it's an O(nk) algorithm(edited above).

Its similar to what wenchao has mentioned

here is the ideone link for the code..http://ideone.com/2mzNYt

1 ^ | v • Reply • Share ›

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding "extern" keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.
    - Check if a binary tree is BST or not
    - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

    - lt_k

      i need help for coding this function in java...

      Java Programming Language · 2 hours ago

    - Piyush

      What is the purpose of else if (recStack[*i])...

Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago