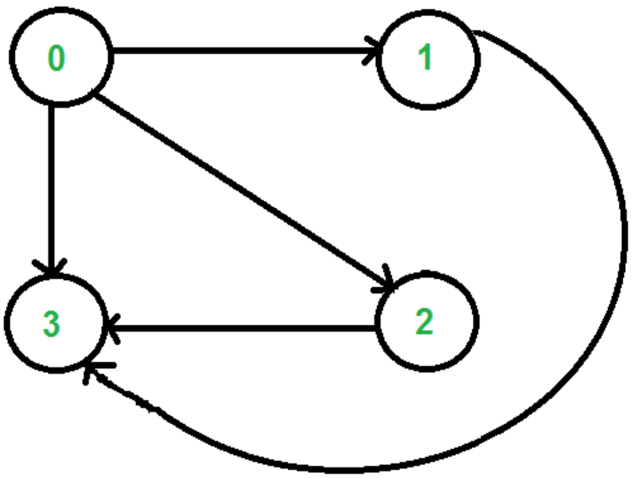
Count all possible walks from a source to a destination with exactly k edges

Given a directed graph and two vertices 'u' and 'v' in it, count all possible walks from 'u' to 'v' with exactly k edges on the walk.

The graph is given as adjacency matrix representation where value of graph[i][j] as 1 indicates that there is an edge from vertex i to vertex j and a value 0 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. The output should be 2 as there are two walk from 0 to 3 with exactly 2 edges. The walks are {0, 2, 3} and {0, 1, 3}



We strongly recommend to minimize the browser and try this yourself first.

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 count walks from u to v with exactly k edges
#include <iostream>
using namespace std;

// Number of vertices in the graph
#define V 4

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

   // Initialize result
   int count = 0;

   // Go to all adjacents of u and recur
   for (int i = 0; i < V; i++)
       if (graph[u][i])  // Check if is adjacent of u
           count += countwalks(graph, i, v, k-1);

   return count;
}

// driver program to test above function
int main()
{
    /* Let us create the graph shown in above diagram*/
     int graph[V][V] = { {0, 1, 1, 1},
```

```
                        {0, 0, 0, 1},
                        {0, 0, 0, 1},
                        {0, 0, 0, 0}
                      };
    int u = 0, v = 3, k = 2;
    cout << countwalks(graph, u, v, k);
    return 0;
}
```

Output:

```
2
```

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 n 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.

```
// C++ program to count walks from u to v with exactly k edges
#include <iostream>
using namespace std;

// Number of vertices in the graph
#define V 4

// A Dynamic programming based function to count walks from u
// to v with k edges
int countwalks(int graph[][V], int u, int v, int k)
{
    // Table to be filled up using DP. The value count[i][j][e] will
    // store count of possible walks from i to j with exactly k edges
    int count[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
                count[i][j][e] = 0;

                // from base cases
                if (e == 0 && i == j)
                    count[i][j][e] = 1;
                if (e == 1 && graph[i][j])
                    count[i][j][e] = 1;

                // go to adjacent only when number of edges is more than 1
```

```
                if (e > 1)
                {
                    for (int a = 0; a < V; a++) // adjacent of source i
                        if (graph[i][a])
                            count[i][j][e] += count[a][j][e-1];
                }
            }
        }
    }
    return count[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, 1, 1, 1},
                         {0, 0, 0, 1},
                         {0, 0, 0, 1},
                         {0, 0, 0, 0}
                       };
    int u = 0, v = 3, k = 2;
    cout << countwalks(graph, u, v, k);
    return 0;
}
```

Output:

```
2
```

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

We can also use **[Divide and Conquer]** to solve the above problem in $O(V^3Logk)$ time. The count of walks of length k from u to v is the [u][v]'th entry in $(graph[V][V])^k$. We can calculate power of by doing O(Logk) multiplication by using the [divide and conquer technique to calculate power]. A multiplication between two matrices of size V x V takes $O(V^3)$ time. Therefore overall time complexity of this method is $O(V^3Logk)$.

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 2 g+1 5

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

36 Comments GeeksforGeeks 1 Login

Recommend 2 Share Sort by Newest

Join the discussion…

Piotr • 21 days ago

You can reduce some memory space by storing only last value of DP array (instead of k times):

http://ideone.com/mhuE5X

• Reply • Share ›

NoobInHell • 2 months ago

This is nothing but a knapsack problem.

The complexity can be reduced to O(V^2k) to get the solution.
Only calc matrix [i][k] which will store no ways you can get to vertex i and with k edges

• Reply • Share ›

Manish M Berwani • 3 months ago

we can also do the problem with same complexity!

If adjacency matrix is A............find A^k that is multiply A with itself (k-1) times. The entries which are 1 are k length entries........

• Reply • Share ›

rihansh • 5 months ago

http://ideone.com/g1TlES

There is only one problem with the dfs based solution is that if there exists a cycle then we cannot traverse the cycle to increase the length of path from src to des .

• Reply • Share ›

rihansh • 5 months ago

I dont know why we cant simply use DFS for finding out the number of K length path from src to des .

• Reply • Share ›

Satya • 7 months ago

Can you please explain in more detail or please give us the alogorithm for divide and conquer approach to solving this problem. I couldnt understand what is meant by searching [u][v]'th entry in (graph[V][V])k

• Reply • Share ›

amitp49 → Satya • 6 months ago

If you take power of given graph matrix K times i.e. V^k then it V[u][v] will give you expected result.

Now to compute A^b, lets take one example of 2^16

2^16 = 2^8 * 2^8
2^8 = 2^4 * 2^4
2^4 = 2^2 * 2*2
2^2 = 2^1 * 2^1

If you observe here, to compute 2^16, we need 4 iteration mainly which is nothing but log(16) = 4.

Lets come back to our original problem where we need to compute V^k. If i can compute V ^ V in O(V cube) complexity then to compute V^k we will need O(V cube) * Log(k) which is nothing but O(V^3 * log k)

Hope this helps!

1 • Reply • Share ›

Akshay • 7 months ago

BFS best approach:-
s--->source
dest-->destination
/*for each vertex V
{
d[v]=0;
color[v]=white;
parent[v]=NIL;
}
Q=NULL;
Q=ENQUE(Q,s);
while(Q!=NULL)
{
u=DEQUE(Q);
for each adj[u]

```
{
if(color[v]==white)
{
```

see more

• Reply • Share ›

MK • 7 months ago

Why compute all for a fixed U & V? We can vary just U in DP solution & the complexity will be O((V^2)K)!

1 • Reply • Share ›

Manav • 8 months ago

We can use bfs.

```
int getwalks(int graph[4][4],int src,int dest,int k){

int queue[5],front=0,walks,i;

queue[front]=src;

while(front>=0){

int v=queue[front--];

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

{if(graph[v][i])

{queue[++front]=i;

graph[v][i]=graph[src][v]+1;

if(i==dest&&graph[v][i]==k)
```

see more

1 • Reply • Share ›

Akshay → Manav • 7 months ago

exactly bfs best solution O(V+E)

• Reply • Share ›

unrealsoul007 • 8 months ago

Why do we need a O(k*V^3) algorithm for this. I think we can do this in O(k*V^2). The algorithm is almost same as above. But in the above algorithm what we are doing extra is we are calculating paths of length k for all possible starting points. Rather we can fix the starting point as 'u' in the base case (e==1 && graph[u][i]) then count[i][e] = 1. See the below code and point out if there are any mistakes.

```
#include <iostream>

using namespace std;

#define V 4

// A Dynamic programming based function to count walks from u

// to v with k edges

int countwalks(int graph[][V], int u, int v, int k)

{
```

see more

4 • Reply • Share ›

Guest • 9 months ago

There is a thm that states-
If A is the adjacency
matrix of a graph G (with vertices v1,…, vn), the (i,
j)-entry of Ar represents the number of distinct r-walks from vertex
vi to vertex vj
in the graph.

complexity- O(KV3)

1 • Reply • Share ›

Urvishsinh Mahida ➦ Guest • 22 days ago

Nice point. I spent some time figuring out about everybody commenting it can be in DP complexity O(V^2*k) and using BFS in O(V*E) but as you pointed out the input is such that will give a O(V^3*k) complexity.
We will require to loop over the whole row each time.
When we are looking for connectivity to a vertex from a given vertex we loop over the while row even though there is no connectivity (0 is the entry).

If we were given graph as input (Node n , Arraylist<nodes> edges equivalent) we could use BFS to achieve O(V*E)

• Reply • Share ›

MS • 9 months ago

```
// Check if there a source to destination path with length =k;
static void printPath(GNode root, GNode dest, int k, Stack<gnode> currentPath) {
if (root == null || k < 0) return;
```

```
if (root == dest && k == 0) {
currentPath.push(root);
print(currentPath);
currentPath.pop();
return;
}

currentPath.push(root);

for (int i = 0; i < root.neigh.size(); i++) {

if (!currentPath.contains(root.neigh.get(i))) {
// System.out.println(currentPath.toString());

printPath(root.neigh.get(i), dest, k - 1, currentPath);

}
}
currentPath.remove(root);
}
```

• Reply • Share ›

rajesh • 9 months ago

wouldn't the top down memoization on the recursive solution provided in this post be O(v^2 * k) ?

• Reply • Share ›

Manish Sharma • 9 months ago

In the recursive solution there is bug in the code. If the graph contains multiple components, the algorithm may never terminate. Consider the case of V=6 and the graph as--> { {0, 1, 0,0,0,0},
{0,0,1,0,0,0},
{0,0,0,1,0,0},
{0,0,0,0,1,0},
{1,0,0,0,0,0},
{0,0,0,0,0,0}
};
for u=0,v=5 and k=3, the code gives segmentation fault.
Basically in the base case one more condition of k<0 must be added.
if(k<0)
return 0;

3 • Reply • Share ›

GeeksforGeeks Mod ➦ Manish Sharma • 9 months ago

Thanks for pointing this out. We have added the base case condition for this.

• Reply • Share ›

Reply • Share ›

Priyal Rathi • 9 months ago

Another approach for dynamic programming solution using following logic:
If the destination is d and source s then,
for all vertices w such that there is edge from w to d
cnt+=func(s,w,k-1); // summation of paths from s to w using k-1 edges
return cnt;

Dynamic Programming code for above: http://ideone.com/VPrK3h

Time complexity: O(V^3 * k)

• Reply • Share ›

Priyal Rathi • 9 months ago

Another approach for the recursive code is:

If the destination is d and source s then,
for all vertices w such that there is edge from w to d

cnt+=func(s,w,k-1); // summation of paths from s to w using k-1 edges
return cnt;

Link: http://ideone.com/FTLUBJ

• Reply • Share ›

GeeksforGeeks Mod • 9 months ago

Ujjwal, Hitesh and Prakhar, thanks for your input. We have added matrix multiplication based method to the above post.

• Reply • Share ›

Itachi Uchiha → GeeksforGeeks • a month ago

Like me many think that the problem can be easily solved using BFS.
If you find anything wrong in the method than why don't you people commment?

```
int getwalks(int graph[4][4],int src,int dest,int k){

int queue[5],front=0,walks,i;

queue[front]=src;

while(front>=0){

int v=queue[front--];

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

{if(graph[v][i])
```

{if(graph[v][i])

{queue[++front]=i;

graph[v][i]=graph[src][v]+1;

see more

• Reply • Share ›

Mahmoud Arafa • 9 months ago

I can't understand what is special in this topic. We can use BFS for example to solve this problem. Please help me understand.

• Reply • Share ›

Prakhar Jain • 9 months ago

Why we need to iterate through all the destinations (j). Can't we keep it fix at V, then solution complexity will be O(V^2 * K).

1 • Reply • Share ›

Hitesh Dholaria • 9 months ago

What if we calculate (graph ^ k)?

Here, graph[][] is an adjacency matrix and (graph ^ k) equals:
graph * graph * graph * ... k times

Say, result = (graph ^ k)

Then, for input vertices, u and v, return result[u][v].

Is this correct way of counting walks of length k between u and v?

• Reply • Share ›

Prakhar Jain → Hitesh Dholaria • 9 months ago

http://stackoverflow.com/quest...

• Reply • Share ›

Nilesh Mishra • 9 months ago

This should be number of walks between 2 vertices but not number of path. As we are using an edge more than 1 time so it wont be a path but a walk.

1 • Reply • Share ›

GeeksforGeeks Mod → Nilesh Mishra • 9 months ago

Nitesh, thanks for pointing this out. We have update the post.

• Reply • Share ›

gaurav • 9 months ago

gaurav • 9 months ago

This can also be a solution right? If its not please tell me why..

http://ideone.com/YcGXnO

• Reply • Share ›

Ujjwal • 9 months ago

The above problem can be solved in O(V^3*K) time and it not only finds number of k length paths between u and v but for all pair of vertices Vi and Vj in the graph. This is basically known as K length walk .

the number of k-length walks between any two vertices vi and vj in G is the i-jth entry of Ak, where A is the adjacency matrix of G.

• Reply • Share ›

Rainer Hoffmann • 9 months ago

Hello,
i wonder why in the if (e > 1) branch you are using k as index. That seems to confuse the k in the outer loop.
Thanks

• Reply • Share ›

GeeksforGeeks Mod → Rainer Hoffmann • 9 months ago

Thanks for pointing this out. We have changed the variable name to 'a'.

• Reply • Share ›

hemanth465 • 9 months ago

can we toplogical sort in reverse order? If so we get the sol in O(v*k)

• Reply • Share ›

garvit • 9 months ago

Please go through this O(V^2*k) solution.
http://ideone.com/i1ahWX

• Reply • Share ›

Ujjwal Prakash • 9 months ago

This can be achieved in O(V^3*logK) time and O(V^3) space. The idea is to find {graph[V][V]}^k using matrix exponentiation and get the value using graph[u][v]

refer: http://pastebin.com/QUcwgGyS

2 • Reply • Share ›

Ujjwal → Ujjwal Prakash • 9 months ago

OMG !! I was about to share the same Approach ! :D

• 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
-