Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm)

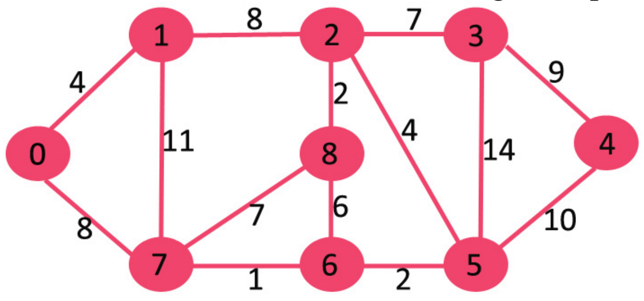
Given a graph and a source vertex in graph, find shortest paths from source to all vertices in the given graph.

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

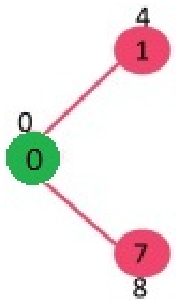
Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.
Algorithm

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
….**a)** Pick a vertex u which is not there in *sptSet*and has minimum distance value.
….**b)** Include u to *sptSet*.
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

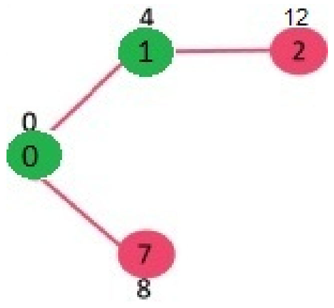
Let us understand with the following example:



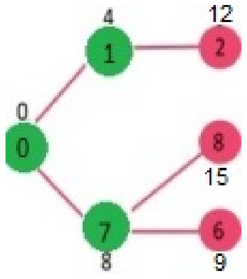
The set *sptSet*is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.



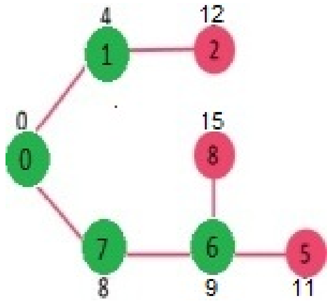
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



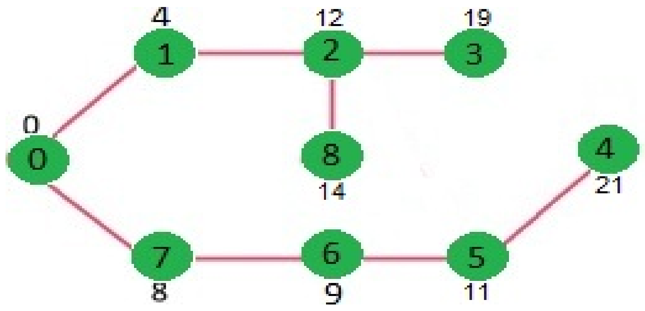
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until *sptSet* doesn't include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



How to implement the above algorithm?
We use a boolean array sptSet[] to represent the set of vertices included in SPT. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not. Array dist[] is used to store shortest distance values of all vertices.

```
// A C / C++ program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

#include <stdio.h>
#include <limits.h>

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
```

```
{
   // Initialize min value
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
     if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
   printf("Vertex   Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}

// Funtion that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
     int dist[V];     // The output array.  dist[i] will hold the shortest
                      // distance from src to i

     bool sptSet[V]; // sptSet[i] will true if vertex i is included in shorte
                     // path tree or shortest distance from src to i is final

     // Initialize all distances as INFINITE and stpSet[] as false
     for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

     // Distance of source vertex from itself is always 0
     dist[src] = 0;

     // Find shortest path for all vertices
     for (int count = 0; count < V-1; count++)
     {
       // Pick the minimum distance vertex from the set of vertices not
       // yet processed. u is always equal to src in first iteration.
       int u = minDistance(dist, sptSet);

       // Mark the picked vertex as processed
       sptSet[u] = true;

       // Update dist value of the adjacent vertices of the picked vertex.
       for (int v = 0; v < V; v++)

         // Update dist[v] only if is not in sptSet, there is an edge from
         // u to v, and total weight of path from src to  v through u is
         // smaller than current value of dist[v]
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                                           && dist[u]+graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
     }

     // print the constructed distance array
     printSolution(dist, V);
}

// driver program to test above function
int main()
{
   /* Let us create the example graph discussed above */
   int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                      {4, 0, 8, 0, 0, 0, 0, 11, 0},
                      {0, 8, 0, 7, 0, 4, 0, 0, 2},
                      {0, 0, 7, 0, 9, 14, 0, 0, 0},
                      {0, 0, 0, 9, 0, 10, 0, 0, 0},
                      {0, 0, 4, 0, 10, 0, 2, 0, 0},
                      {0, 0, 0, 14, 0, 2, 0, 1, 6},
                      {8, 11, 0, 0, 0, 0, 1, 0, 7},
                      {0, 0, 2, 0, 0, 0, 6, 7, 0}
                     };

    dijkstra(graph, 0);

    return 0;
}
```

Output:

```
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14
```

Notes:
1) The code calculates shortest distance, but doesn't calculate the path information. We can create a parent array, update the parent array when distance is updated (like prim's implementation) and use it show the shortest path from source to different vertices.

2) The code is for undirected graph, same dijekstra function can be used for directed graphs also.

3) The code finds shortest distances from source to all vertices. If we are interested only in shortest distance from source to a single target, we can break the for loop when the picked minimum distance vertex is equal to target (Step 3.a of algorithm).

4) Time Complexity of the implementation is O(V^2). If the input graph is represented using adjacency list, it can be reduced to O(E log V) with the help of binary heap. We will soon be discussing O(E Log V) algorithm as a separate post.

5) Dijkstra's algorithm doesn't work for graphs with negative weight edges. For graphs with negative weight edges, Bellman–Ford algorithm can be used, we will soon be discussing it as a separate post.
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: Dijkstra, Graph, Greedy Algorithm

Tweet 2 g+1 2

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

58 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Andy Toh • 6 days ago
Compile-time solution (with path information outputted as well):

http://ideone.com/Eu8pxI

Reply • Share ›

Andy Toh • 6 days ago
Compile-time solution (without outputting path information):

http://ideone.com/iDweb8

Reply • Share ›

Mahmud Iftekhar Zamil • 13 days ago
Hello . I am trying to solve UVa 11631 problem using dijkstra. My input set is as follow;

7 11

0 1 7

0 3 5

1 2 8

1 3 9

1 4 7

2 4 5

3 4 15

3 5 6

4 5 8

4 6 9

5 6 11

* V = 7, E = 11 << 1st line. From 1st node (0) to last node(6) what should be the MST value. I am getting 22. Can someone check?

• Reply • Share ›

Tejas • a month ago

Shouldn't count go from 0 to less than V.

// Find shortest path for all vertices
for (int count = 0; count < V-1; count++)

• Reply • Share ›

Goku • a month ago

C++ implementation using stl : http://ideone.com/o2acci

• Reply • Share ›

Raj • a month ago

void

dijkstra(vector< list<weightedgraph> >& Graph){

int* parent = new int[Graph.size()]; // Array to store constructed MST

int* key= new int[Graph.size()];; // Key values used to pick minimum weight edge in cut

bool* mstSet = new bool[Graph.size()]; // To represent set of vertices not yet included in MST list<weightedgraph>::iterator it;

list<weightedgraph>::iterator it;

// Initialize all keys as INFINITE

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

key[i] = INT_MAX,

mstSet[i] =

see more

• Reply • Share ›

Question • a month ago

Is there anyway to output the path that was taken to get to the destination?

• Reply • Share ›

haibo wang • 2 months ago

In the for loop, should the check condition be "count < V;" ?
// Find shortest path for all vertices
for (int count = 0; count < V-1; count++)
{
...

}

• Reply • Share ›

kailashagouda • 2 months ago

how to get openGL code for

Dijkstra's Algorithm

• Reply • Share ›

Aditya Goel • 3 months ago

dist[u] != INT_MAX in if condition is not contributing to anything. We can remove it

• Reply • Share ›

Guest ➦ Aditya Goel • a month ago

It has been used to avoid arithmetic overflow.

• Reply • Share ›

Pavel Kutáč ➦ Aditya Goel • 2 months ago

Agree. I just tried to find reason for that, but I wasn't successful.

• Reply • Share ›

Guest • 3 months ago

if (sptSet[v] == false && dist[v] <= min)
y dist[v] <= min? Shouldn't it be dist[v] < min?

• Reply • Share ›

Sanghwa Jung • 4 months ago

my java code is here http://javamusician.blogspot.k...

2 • Reply • Share ›

João Dias • 4 months ago

To print the shortest path for each node I used an array, prev[V], in order to keep not only the distance to it but also the previous node in the path.

I initialized all the elements of this array with an impossible value, -1, and then, as suggested, when the distance is updated, the previous node is also updated: prev[v] = u;

Then, to print the path, I used another utility function:

//An utility function to print the minimum path by recursively printing the //previous node, from the destination to the source

void printPath(int dest, int prev[]){

if (prev[dest] != -1){ //If we are not at the source node

printPath(prev[dest], prev);

printf(" - ");

}

printf("%d", dest);

}

1 • Reply • Share ›

Lan Desko • 5 months ago

would love to see the path implementation. it is not clear in the prim's example. thanks!

2 • Reply • Share ›

Victor Santana • 5 months ago

Might please tell me as I put the code if I want to display the path information? Thank you !!

1 • Reply • Share ›

Vaibhav Daga • 6 months ago

It can also be implemented using simple recursion
http://ideone.com/TsQjyS

Please check it for any flaw.

8 • Reply • Share ›

Tuan Dang • 6 months ago

Please check: The program's output is different to the example's result.

• Reply • Share ›

Vineet Sharma • 9 months ago

why we need to find the minimum distance value of the vertex not included in SPT set???

• Reply • Share ›

helper Vineet Sharma • 7 months ago

because the minimum distance from source to that vertex is still not finalized and can change...

If you are curious about why the min. dist. of that path is finalized and how we are sure that we will not find a lower cost path from src to that vertex in future using the unused vertices, you should look at this:

https://www.cs.auckland.ac.nz/...

1 • Reply • Share ›

Swati • 10 months ago

dist[u] != INT_MAX && dist[u]+graph[u][v] < dist[v]
why the first condition is required as even it its absence second condition will take care for INFINITE vertices.

• Reply • Share ›

Jon Snow Swati • 7 months ago

Because of possibility of overflow. If we are not checking first condition and simply adding dist[u]+graph[u][v] it will overflow as dist[u] is initialized to INT_MAX. overflow means value of dist[u]+graph[u][v] become negative and second condition will always evaluates to true.

1 • Reply • Share ›

skal • 10 months ago

Hi,

if I use : parent[v] = u, dist[v] = dist[u] + distArr[u][v]; as described on Notes (1) I only get the latest vertex of the path to destination.

The code is working if I iterate calling the dijkstra() from all every source I get all the minimum paths from source to dist. But parent[v] only holds the last step of the path!

Any ideas?

5 ^ | ˅ • Reply • Share ›

GaoTong • 10 months ago

The code is wrong! The test is right, because the special test data.

This is wrong:

for (int i = 0; i < V; i++)

dist[i] = INT_MAX, sptSet[i] = false;

It should be:

for (int i = 0; i < V; i++){

dist[i] = (graph[source][i] == 0 ? INT_MAX:graph[source][i]);

sptSet[i] = false;

}

7 ^ | ˅ • Reply • Share ›

JRa ➦ GaoTong • 9 months ago

Umm, actually if you were to change the code like you mentioned, your FOR loop would be redundant...because on the first iteration, the minDistance() function would return the 0-distance (source) as 'u', and set the adjacent nodes values redundantly.

^ | ˅ • Reply • Share ›

jithin • a year ago

i think the above implementation is taking 0(n^3) : outer for loop iterating n times , function minDistance : taking o(n) time to compute minimum and inner for loop taking O(n). ie n*n*n = n^3

4 ^ | ˅ • Reply • Share ›

prashant jha • a year ago

u can use min heap or rmq for reducing complexity

http://ideone.com/2gJjz2

not work for negative edges because here at each step of algorithm we make a vertex permanent that is we have found shortest path upto dat vertex but if it has negative edges there may exist a path shorter than dat length ..so greedy algorithhm dont work there

^ | ˅ • Reply • Share ›

andrei • a year ago

how should one modify Dijkstra to list all simple shortest paths to target?

5 ^ | ˅ • Reply • Share ›

James Sutton • a year ago

Can anyone convert this for use with Arduino? My attempts have failed so far. I think maybe because I haven't quite grasped what each part of the code is doing and therefore can't define them in the arduino sketch as such. Any help would be much appreciated!

^ | ˅ • Reply • Share ›

zak • a year ago

I am trying to compile the code by MVS 2012 but it does not work

^ | ˅ • Reply • Share ›

tan • a year ago

can it work for directed graph?

4 ^ | ˅ • Reply • Share ›

yossi • a year ago

hy i want to have the detail about the path if you can help me

4 ^ | ˅ • Reply • Share ›

Tomislav • a year ago

N 0 1 2 3 4 5 6 7 8
0 {{0, 4, 0, 0, 0, 0, 0, 8, 0},
1 {4, 0, 8, 0, 0, 0, 0, 11, 0},
2 {0, 8, 0, 7, 0, 4, 0, 0, 2},
3 {0, 0, 7, 0, 9, 14, 0, 0, 0},
4 {0, 0, 0, 9, 0, 10, 0, 0, 0},
5 {0, 0, 4, 14 instead of 0, 10, 0, 2, 0, 0},
6 {0, 0, 0, 14, 0, 2, 0, 1, 6},
7 {8, 11, 0, 0, 0, 0, 1, 0, 7},
8 {0, 0, 2, 0, 0, 0, 6, 7, 0}
};

If 3 is connected with 5 with weight of 14, 5 is therefore connected with 3 with weight of 14 instead of 0.

2 ^ | ˅ • Reply • Share ›

Mate ➦ Tomislav • a year ago

0 {{0, 4, 0, 0, 0, 0, 0, 8, 0},
1 {4, 0, 8, 0, 0, 0, 0, 11, 0},
2 {0, 8, 0, 7, 0, 4, 0, 0, 2},
3 {0, 0, 7, 0, 9, 14, 0, 0, 0},

4 {0, 0, 0, 9, 0, 10, 0, 0, 0},
5 {0, 0, 4, 14 instead of 0, 10, 0, 2, 0, 0},
6 {0, 0, 0, 14 (here is not 14), 0, 2, 0, 1, 6},
7 {8, 11, 0, 0, 0, 0, 1, 0, 7},
8 {0, 0, 2, 0, 0, 0, 6, 7, 0}
};

Because 6 is not connected with 3

• Reply • Share ›

Tomislav • a year ago

N 0 1 2 3 4 5 6 7 8
0 {{0, 4, 0, 0, 0, 0, 0, 8, 0},
1 {4, 0, 8, 0, 0, 0, 0, 11, 0},
2 {0, 8, 0, 7, 0, 4, 0, 0, 2},
3 {0, 0, 7, 0, 9, 14, 0, 0, 0},
4 {0, 0, 0, 9, 0, 10, 0, 0, 0},
5 {0, 0, 4, 0, 10, 0, 2, 0, 0},
6 {0, 0, 0, 14, 0, 2, 0, 1, 6},
7 {8, 11, 0, 0, 0, 0, 1, 0, 7},
8 {0, 0, 2, 0, 0, 0, 6, 7, 0}
};

• Reply • Share ›

Ankur • 2 years ago

dist[u]!=INF is a redundant check since dist[u]==INF will only be true if it was an isolated node , but than again all values of graph[u][v] will be 0 for this node and the condition

if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]+graph[u][v] < dist[v])

will never be true. But since this situation is tackled by checking graph[u][v] , so why bothering checking dist[u]!=INF ?

5 • Reply • Share ›

Nikhil Choudhary • 2 years ago

Problem Statement:-

Consider a data communication
network that must route data packets (email or MP3 files, for

example). Such a network
consists of routers connected by physical cables or links. A router can act as
a source, a destination, or a forwarder of data packets. We can model a network
as a graph with each router corresponding to a vertex and the link or physical connection

vertices.

A network that follows the OSPF
(Open Shortest Path First) protocol routes packets using

Dijkstra’s shortest path
algorithm. The criteria used to compute the weight corresponding to a

link can include the time taken

see more

1 • Reply • Share ›

Nikhil Choudhary • 2 years ago

can anyone plz upload the code for dijstraks and bellman algorithm in a single program of c++ (so that we can easily have positive and negative weight edges)

• Reply • Share ›

javaDude • 2 years ago

could you fix the formatting T_T

1 • Reply • Share ›

javaDude • 2 years ago

The article was very helpful, thank you. I changed the code so that it is clearer to me (some parts left out):

```
int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index = -1;

   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] < min)
         min = dist[v], min_index = v;

   return min_index;
}

void dijkstra(int graph[V][V], int src)
{

   //init code
```

see more

• Reply • Share ›

viki • 2 years ago

I think "&& dist[u] != INT_MAX" is redundant check.

• Reply • Share ›

Ashish Tilokani ↪ viki • a year ago

Its used because the next condition is that of dist[u]+graph[u][v] < dist[v] which can cause overflow as Int type cannot store more than INT_MAX
and the LHS can be INT_MAX + any positive integer

2 • Reply • Share ›

Kailash Gupta • 2 years ago

Should not we break the outer for loop when [sourcecode]dist[u] == INT_MAX ?Because that means no node has left which can be reached from source node

• Reply • Share ›

Tan Syh Ren • 2 years ago

the graph matrix seems to mismatch with the illustration

• Reply • Share ›

GeeksforGeeks • 2 years ago

GCC

• Reply • Share ›

Tan Syh Ren • 2 years ago

may i know what&#039s your compiler? I tried to use dev c++ but lots of erros come out. a little modification will make it compile though

• Reply • Share ›

Ray Garner • 2 years ago

Useful thanks

• Reply • Share ›

Sandeep Jain • 2 years ago

Ankit Paharia Thanks for pointing this out. We have updated the code.

• Reply • Share ›

Sarthak Mall 'shanky' • 2 years ago

Ankit bro hum to sirf padte hain is site se,tu to comment bhi karne laga ...tod bhai... :P

• Reply • Share ›

Load more comments

-
-
-
-
    - 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 · 1 hour ago

  - Piyush

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

    Detect Cycle in a Directed Graph · 1 hour ago

  - Andy Toh

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

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour 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) · 2 hours ago

  - manish

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

    Given two strings, find if first string is a subsequence of second · 2 hours ago

-