Negative weights using Dijkstra's Algorithm

I am trying to understand why Dijkstra's algorithm will not work with negative weights. Reading an example on Shortest Paths, I am trying to figure out the following scenario:

```
    2
A-------B
 \     /
3 \   / -2
   \ /
    C
```

From the website:

> Assuming the edges are all directed from left to right, If we start with A, Dijkstra's algorithm will choose the edge (A,x) minimizing d(A,A)+length(edge), namely (A,B). It then sets d(A,B)=2 and chooses another edge (y,C) minimizing d(A,y)+d(y,C); the only choice is (A,C) and it sets d(A,C)=3. But it never finds the shortest path from A to B, via C, with total length 1.

I can not understand why using the following implementation of Dijkstra, d[B] will not be updated to `1` (When the algorithm reaches vertex C, it will run a relax on B, see that the d[B] equals to `2`, and therefore update its value to `1`).

```
Dijkstra(G, w, s)  {
   Initialize-Single-Source(G, s)
   S ← ∅
   Q ← V[G]//priority queue by d[v]
   while Q ≠ ∅ do
      u ← Extract-Min(Q)
      S ← S U {u}
      for each vertex v in Adj[u] do
         Relax(u, v)
}

Initialize-Single-Source(G, s) {
   for each vertex v ∈ V(G)
      d[v] ← ∞
      π[v] ← NIL
   d[s] ← 0
}

Relax(u, v) {
   //update only if we found a strictly shortest path
   if d[v] > d[u] + w(u,v)
      d[v] ← d[u] + w(u,v)
      π[v] ← u
      Update(Q, v)
}
```

Thanks,

Meir

algorithm dijkstra shortest-path graph-algorithm

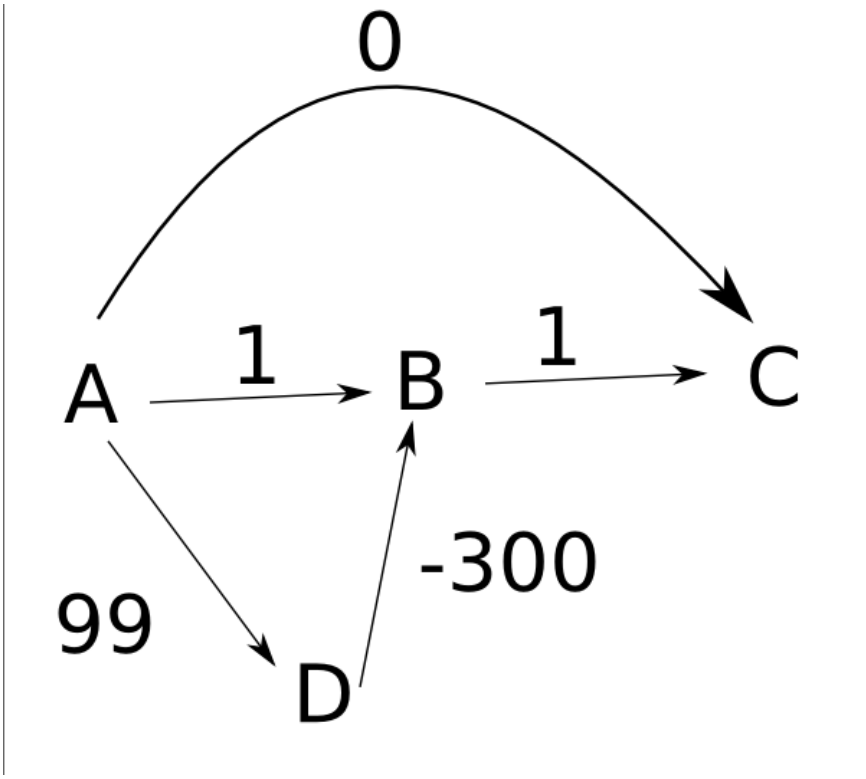
edited Feb 28 '13 at 17:14
templatetypedef
152k ● 35 ● 373 ● 605

asked Jul 23 '11 at 8:19
Meir
416 ● 1 ● 5 ● 11

Pathfinding in general with negative edge weights is extremely difficult. No matter what route you find, there's always the possibility of an arbitrarily long route with an arbitrarily large negative edge weight somewhere along it. I wouldn't be surprised if it's NP complete. – Nick Johnson Jan 16 '12 at 2:47

6 Answers

The algorithm you have suggested will indeed find the shortest path in this graph, but not all graphs in general. For example, consider this graph:



Assume the edges are directed from left to right as in your example,

Your algorithm will work as follows:

1. First, you set `d(A)` to `zero` and the other distances to `infinity`.
2. You then expand out node `A`, setting `d(B)` to `1`, `d(C)` to `zero`, and `d(D)` to `99`.
3. Next, you expand out `C`, with no net changes.
4. You then expand out `B`, which has no effect.
5. Finally, you expand `D`, which changes `d(B)` to `-201`.

Notice that at the end of this, though, that `d(C)` is still `0`, **even though the shortest path to `C` has length `-200`.** Your algorithm thus fails to accurately compute distances in some cases. Moreover, even if you were to store back pointers saying how to get from each node to the start node `A`, you'd end taking the wrong path back from `C` to `A`.

edited May 10 '14 at 8:18
wpp
2,176 ● 1 ● 9 ● 33

answered Jul 23 '11 at 8:53
templatetypedef
152k ● 35 ● 373 ● 605

13 To add to your excellent answer: Dijkstra being a **greedy algorithm** is the reason for its short-sighted choice. – blubb Jul 23 '11 at 9:15

1 I would like to point out that, technically, all paths in this graph have a cost of negative infinity courtesy of the negative cycle A,D,B,A. – Nate Nov 10 '11 at 13:16

2 @Nate- To clarify, all the edges in the graph are directed from left to right. It was kinda hard to render arrows in my high-quality ASCII art. :-) – templatetypedef Nov 10 '11 at 19:05

11 @templatetypedef I replaced your high-quality ASCII art diagramm with a plain png grafic... ;) – lumbric Jun 15 '12 at 7:37

1 For those who haven't seen graphs with negative edges before, I find a useful interpretation of this graph to be a network of toll roads, where the edge weights give the toll you pay. The -300 road is a crazy backwards toll road where they give you $300 instead. – D Coetzee Dec 14 '12 at 3:35

I understand the OP's question as *why* Dijkstra's algorithm fails in this case. Let's see.

First off, the correct statement is that "Dijkstra's algorithm fails if there are negative *directed* edges". If there are negative *undirected* edges, then every negative edge immediately constitutes a negative *cycle* (`X->Y->X`), and the graph does not *have* a shortest path and the question is moot.

So to make this non-trivial, let's suppose the graph is directed:

```
A -> B : 2
A -> C : 3
C -> B : -2
```

Suppose further our source is `A` . The true shortest path from A to B is `A-C-B` with weight 1. But Dijkstra's algorithm starts by selecting `A->B` , and then removes that edge from future consideration *because it assumes that it cannot be improved upon*.

This is crucial: Under the non-negativity hypothesis, we can be sure once and for all that the shortest path from A to B is `A -> B : 2` . Because the algorithm assumes this, it never checks again later whether there is an even shorter path, because with non-negative edges there cannot be on.

This hypothesis allows the algorithm to proceed with fewer steps, but it precludes it from correctly identifying the shortest path in the presence of negative edges. The negative edges make the problem "non-local" in the sense that you cannot tell if you're doing well just by looking at your immediate neighbours - there might always be an extremely negative edge just past this very expensive edge next to you.

answered Jul 23 '11 at 8:55

Kerrek SB
223k ● 26 ● 382 ● 602

So you're saying that his implementation works on his example because it's not strictly Dijkstra's algorithm? – Jeff Feb 23 '12 at 20:54

1 @Jeff: I wouldn't say that the algorithm "works", much like `f(x) = 100` is not the correct implementation of the "square" function although it is correct for `f(10)` . – Kerrek SB Feb 23 '12 at 21:05

Very nice explanation. I have always heard about the negative weight cycle tending toward negative infinity which to me never made sense since djikstra's algorithm only incorporates an edge once. Nice explanation and clarification on the matter – cpowel2 Dec 12 '12 at 20:41

3 It will still work for your directed graph, since at the end, when you extract C from the heap, it will relax B and change the B's weight to 1. It will not work for the top answer – Pan Dec 29 '12 at 22:12

you did not use S anywhere in your algorithm (besides modifying it). the idea of dijkstra is once a vertex is on S, it will not be modified ever again. in this case, once B is inside S, you will not reach it again via C.

this fact ensures the complexity of O(E+VlogV) [otherwise, you will repeat edges more then once, and vertices more then once]

in other words, the algorithm you posted, might not be in O(E+VlogV), as promised by dijkstra's algorithm.

edited Jul 23 '11 at 9:07

answered Jul 23 '11 at 8:42

amit
89.7k ● 12 ● 77 ● 163

Also, there is no need to modify the vertex without negative weight edges, which completely breaks the assumption that path costs can only increase with repeated edges – prusswan Jul 23 '11 at 8:50

this assumption is exactly what allows us to use S, and 'knowing' once a vertex is in S, it will never be modified again. – amit Jul 23 '11 at 8:53

Consider what happens if you go back and forth between B and C...voila

(relevant only if the graph is not directed)

Edited: I believe the problem has to do with the fact that the path with AC* can only be better than AB with the existence of negative weight edges, so it doesn't matter where you go after AC, with the assumption of non-negative weight edges it is impossible to find a path better than AB once you chose to reach B after going AC.

edited Jul 23 '11 at 9:04

Henk Holterman
164k ● 12 ● 135 ● 279

answered Jul 23 '11 at 8:31

prusswan
4,411 ● 3 ● 19 ● 38

this is not possible, the graph is directed. – amit Jul 23 '11 at 8:32

@amit: good point, I missed that. Time to reconsider the problem – prusswan Jul 23 '11 at 8:39

Note, that Dijkstra works even for negative weights, if the Graph has no negative cycles, i.e. cycles whose summed up weight is less than zero.

Of course one might ask, why in the example made by templatetypedef Dijkstra fails even though there are no negative cycles, infact not even cycles. That is because he is using another stop criterion, that holds the algorithm as soon as the target node is reached (or all nodes have been settled once, he did not specify that exactly). In a graph without negative weights this works fine.

If one is using the alternative stop criterion, which stops the algorithm when the priority-queue (heap) runs empty (this stop criterion was also used in the question), then dijkstra will find the correct distance even for graphs with negative weights but without negative cycles.

However, in this case, the asymptotic time bound of dijkstra for graphs without negative cycles is lost. This is because a previously settled node can be reinserted into the heap when a better distance is found due to negative weights. This property is called label correcting.

edited Aug 6 '14 at 13:06

answered Aug 6 '14 at 10:54

infty10000101
11 ● 2

1. That is not Dijkstra's algorithm anymore. – Gassa Aug 6 '14 at 11:11

2. It is not clear why you think the time would me "more like Bellman-Ford" and not exponential (which is worse than Bellman-Ford). Do you have a concrete algorithm and a proof in mind? – Gassa Aug 6 '14 at 11:12

To 1.: as you can use exactly the same implementation of dijkstra with the mentioned stop criterion, that stops when a the queue runs empty (see pseudocode in original question), it is still dijkstras algorithm for shortest paths, even though it behaves differently settling nodes several times (label correcting). – infty10000101 Aug 6 '14 at 11:19

To 2.: That was just a guess so I'm going to delete that. I' think you're right with the exponential time, as there are exponentially many paths, which have to be explored. – infty10000101 Aug 6 '14 at 11:20

Since Dijkstra is a Greedy approach, once a vertice is marked as visited for this loop, it would never be reevaluated again even if there's another path with less cost to reach it later on. And such issue could only happen when negative edges exist in the graph.

answered Oct 25 '14 at 9:36

punchoyeah
28 ● 5