Modifying Dijkstra's algorithm for edge weights drawn from range $[1, \ldots, K]$

Suppose I have a directed graph with edge weights drawn from range $[1, \ldots, K]$ where K is constant. If I'm trying to find the shortest path using Dijkstra's algorithm, how can I modify the algorithm / data structure and improve the time complexity to $O(|V| + |E|)$?

algorithms data-structures shortest-path weighted-graphs

edited Nov 21 '12 at 8:01
A.Schulz
11.1k 1 28 54

asked Nov 21 '12 at 3:08
user1675999
537 3 11 20

You should more specific, What is your data structure? And You can not get less $O(V + E)$. Review lectures. – jonaprieto Nov 21 '12 at 3:36

Just because the possibilities for distinct edge weights are small doesn't mean that the number of distances is small. – Joe Nov 21 '12 at 3:56

3 First color vertices of your graph with blue, then subdivide each edge of size t into t edges (by adding $t-1$ uncolored vertices), then run the BFS on this new graph, to find shortest paths from start node to blue nodes, it's $O(|V| + |E|)$ if you have constant k. – user742 Nov 21 '12 at 17:16

@SaeedAmiri why not write this up as an answer? – Joe Nov 22 '12 at 0:35

@Joe because it's not modifying dijkstra (at least directly is not related to dijkstra). – user742 Nov 22 '12 at 8:52

3 Answers

If edge weights are integers in $\{0, 1, \ldots, K\}$, you can implement Dijkstra's to run in $O(K|V| + |E|)$ time, following @rrenaud's suggestion. Here is a more explicit explanation.

At any time, the (finite) keys in the priority queue are in some range $\{D, D+1, \ldots, D+K\}$, where D is the value of the last key removed from the priority queue. (Every key is at least D, because the sequence of keys removed by Dijkstra's algorithm is non-decreasing, and every key is at most $D+K$, because every key has value $d[u] + wt(u,w)$ for some edge (u,w) where $d[u]$ is the distance from the source to some vertex u that has already been removed, so $d[u] \le D$.)

Because of this, you can implement the priority queue with a circular array $A[0..K]$ of size $K+1$, with each cell containing a bucket. Store each vertex with key k in the bucket in cell $A[h(k)]$ where $h(k) = k \bmod (K+1)$. Keep track of D. Do operations as follows:

- **delete-min**: While $A[h(D)]$ is empty, increment D. Then delete and return a vertex from $A[h(D)]$.
- **insert** with key k: Add the vertex to the bucket of $A[h(k)]$.
- **decrease-key** k to k': Move the vertex from $A[h(k)]$ to $A[h(k')]$.

Insert and decrease-key are constant-time operations, so the total time spent in those operations will be $O(|V| + |E|)$. The total time spent in delete-min will be $O(|V|)$ plus the final value of D. The final value of D is the maximum (finite) distance from the source to any vertex (because a delete-min that takes i iterations increases D by i). The maximum distance is at most $K(|V|-1)$ because each path has at most $|V|-1$ edges. Thus, the total time spent by the algorithm is $O(K|V| + |E|)$.

edited Nov 21 '12 at 16:42

answered Nov 21 '12 at 16:34
Neal Young
338 2 6

I like the circular queue, that's way better than my idea of basically having a K*v size array where only a v sized slice is used at any given time. – rrenaud Nov 21 '12 at 16:43

I implemented it using a double linked list, does that still mean it is O(1) for finding the min key? – user1675999 Nov 25 '12 at 18:52

@user1675999, I'm not sure. if your list is sorted by key, how you do insert and decrease-key efficiently? if your list is not sorted by key, how to you do delete-min efficiently? – Neal Young Nov 25 '12 at 21:34

I assume here that K is an integer and the edge weights are integral. Otherwise it doesn't really buy you anything, you can always rescale weights so that the min edge has cost 1 and the max has cost K, so the problem is identical to the standard shortest path problem.

Algorithm/proof sketch: Implement the priority queue in this kind of crazy way as an array of $K \times |V|$ lists keyed by cost and otherwise use the standard Dijkstra's algorithm. Keep a counter that tracks the cost of the minimum item in the heap. Resolve the dequeue call after items are deleted by *linear scanning*. Yes, this sort of sounds insane, but constant K let's you cheat and fool your algorithmic intuition against linear scans. You only need to scan up from the last min marker because Disjkstra's algorithm is nice to your queue implementation. By the time it requests a dequeue, the items inserted into queue are always greater or equal to the previous minimum. The longest shortest path possible has length $K \times |V|$, so your amortized scanning cost is $K \times |V| = O(|V|)$ if K is constant.

edited Nov 21 '12 at 20:43
Merbs
1,083 1 9 20

answered Nov 21 '12 at 5:03

rrenaud
308 1 4

you can use topological sort to find the solution, let the source to have degree 0 then, go from each edge from the source, if another vertex has 0 degree then put it into the queue and keep doing that. in this case(without cycle inside the graph) it can achieve V + E as it would go through every vertex and edges once and only once.

answered May 6 at 9:01

TIANYANG ZHANG
1

Seems unrelated to the question? The question does not assume the graph is acyclic, and your solution does not make use of the fact that weights are drawn from a constant range. – xskxzr May 6 at 14:18