Dijkstra's algorithm for edge weights in range 0, ..., W

Suppose I want to run Dijkstra's algorithm on a graph whose edge weights are integers in the range 0, ..., W, where W is a relatively small number. How can I modify that algorithm so that it takes time just O((|V| + |E|) logW) and relatively easy implement that in C/C++?

algorithms algorithm-analysis data-structures shortest-path weighted-graphs

asked Dec 24 '13 at 16:15
Lucy
16 1 2

1 I know of a fairly simple algorithm that runs in O(|E| + |V| * W). A paper describing a more complex algorithm in O(|E| + |V| * logW) can be found here: scidok.sulb.uni-saarland.de/volltexte/2011/4196/pdf/... – sebii Dec 24 '13 at 17:02

@sebii - Do you have an implementation of an algorithm that runs in O(|E|+|V|*W)? It would be very useful as a starting point. – Lucy Dec 24 '13 at 17:11

@Lucy This isn't a programming site; unless somebody knows of an implementation, you'll have to do that part yourself. – David Richerby Dec 24 '13 at 19:01

2 Answers

This is the algorithm I know of that is $\mathcal{O}(|E| + W|V|)$. Some things to note:

- C is defined as the maximum cost of any shortest path that leaves s (the source). It is at most $W|V|$.
- Each $L[i]$ is a doubly-linked list. The $L[i]$ keep track of all vertices that are currently at distance i from the source.
- $d[v]$ is the shortest distance (so far) from the source to vertex v

Each edge and vertex of the graph will be visited only once, and the outer loop runs in time $\mathcal{O}(C)$, therefore the complexity of the algorithm is $\mathcal{O}(|E| + |V| + C)$. Since $C \leq W|V|$, this reduces to $\mathcal{O}(|E| + W|V|)$.

```
L[i] <- {}, 0 <= i <= C
L[inf] <- {}
L[0] <- {s}

for v in V do
    push v into L[inf]
    d[v] <- inf
end

d[s] <- 0
p <- 0
while p < C do
    while L[p] != {} do
        pop v from L[p]
        for (v, u) with d[u] > d[v] + w(v, u) do
            remove u from L[d[u]]
            d[u] <- d[v] + w(v, u)
            push u in L[d[u]]
        end
    end
    p <- p + 1
end
```

answered Dec 24 '13 at 18:44
sebii
131 3

The standard Dijkstra's algorithm uses a priority queue whose elements are vertices, and where the priority of a vertex v is the current estimate of the distance $d[v]$. The priority queue might contain a maximum of $|V|$ vertices at any point in time. If you use a heap as your priority queue, the running time of each operation on a priority queue is logarithmic in the number of items in the queue, so each priority queue operation might take $O(\lg |V|)$ time and the total running time is $O((|V| + |E|) \lg |V|)$.

In your situation where the weights are integers from $1..W$, we can get the running time down to $O((|V| + |E|) \lg W)$. The basic approach will be to ensure that the priority queue contains no more than W elements at any point in time. Thus, the running time of each operation on the priority queue will be $O(\lg W)$, and we will attain the desired running time.

How do we keep the size of the priority queue down to $\leq W$ at all points in time? Easy: instead of a queue of vertices, we will have a queue of buckets. Each bucket contains a set of vertices that all have the same distance estimate, and the priority of a bucket is that distance estimate. In particular, for each vertex v that would be in the queue in standard Dijkstra's algorithm, our modified algorithm will have a bucket for the integer $d[v]$. The bucket for the integer i will contain a linked list of all vertices v such that $d[v] = i$ and such that v would be in the queue right now in standard Dijkstra's algorithm.

Now I claim that, in standard Dijkstra's algorithm, the set of values of $d[v]$ (taken over the vertices v in the queue at any point of time) contains at most W distinct values. Why? Well, at any point in time, there is some integer α such that we have assigned a final distance value to all vertices of distance at most α from the source. The queue contains vertices that are one edge away from one of those finalized vertices. Consequently, the distance estimate for any vertex in the queue has to be somewhere in the range $\alpha + 1 \ldots \alpha + W$. Since all distances are integers, this means there are at most W possible distance estimates in the queue at any point in time. Therefore, our modified algorithm will have at most W different buckets in its queue, and each operation on the queue will take at most $O(\lg W)$ time.

To make this work, we'll have to make one other change: instead of implementing the priority queue as a heap, we'll implement it as a balanced binary search tree, using the distance as the key. A balanced binary search tree supports all of the standard priority queue operations (including extract-min); it is useful here because when we want to insert a node into the queue, we must first find its corresponding bucket, which is something that can be done in $O(\lg W)$ time using a balanced binary search tree.

This algorithm is almost easier to implement than to describe.....

edited Dec 25 '13 at 17:06

answered Dec 24 '13 at 21:52
D.W. ♦
97.7k 11 114 269

Question: how do you implement the queue? You have to find the bucket for a given distance, which seems more search tree than priority like? – Hendrik Jan Dec 25 '13 at 10:16

@HendrikJan, yes, good point, you need to use a balanced binary search tree instead of a heap. I've edited my answer accordingly. With that modification, I think it now works. Thank you for catching that. – D.W. ♦ Dec 25 '13 at 17:07

"The queue contains vertices that are one edge away from one of those finalized vertices." - that's not true for standard Dijkstra's algorithm. The queue contains all the vertices that have not yet been visited. What you said is true if you keep only the adjacent vertices in the queue, which is then contradictory to the following statement in the opening paragraph *"The priority queue might contain a maximum of* $|V|$ *vertices at any point in time."* – Abhijit Sarkar 1 hour ago

@AbhijitSarkar, good point. But all other vertices in the queue will have distance estimate $d[v] = \infty$, so they add at most one to the total number of distinct values for $d[v]$ in the queue, so it doesn't change the bottom line conclusions in my answer. – D.W. ♦ 12 mins ago