Second Shortest Path in a Graph

Formalized

- Given a directed, weighted, graph.
- Single-pair shortest path problem:
  - Given a pair of vertices *u* and *v,* find a (u,v)-path, such that the sum of the weights of its edges is minimized.
- Second shortest path problem:
  - Find a (u,v)-path which is the shortest path amongst all (u-v)-paths with cost strictly greater than the shortest (u-v)-path.

Why shortest path?

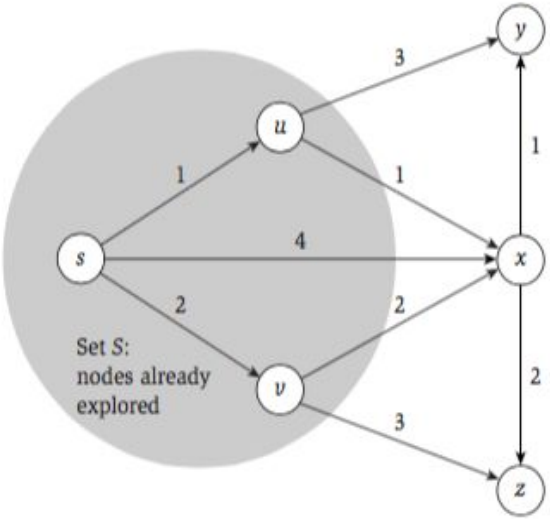
- Route planning and navigation
- Network design, packet routing
- Degree of separation in a social network
- Currency conversion (given currencies and exchange rates, what is the best way to convert US to CAD?)
- Many other optimization problems....

Why second shortest path?

- Generation of alternatives
  - Useful to look at a larger class of solutions, not just the optimal one.
- Additional constraints are ill-defined.
  - e.g. a power line should connect its endpoints reasonably directly, but there may be more or less community support for certain options.

Yen’s Algorithm

Dijkstra



Going from source node s to destination node u

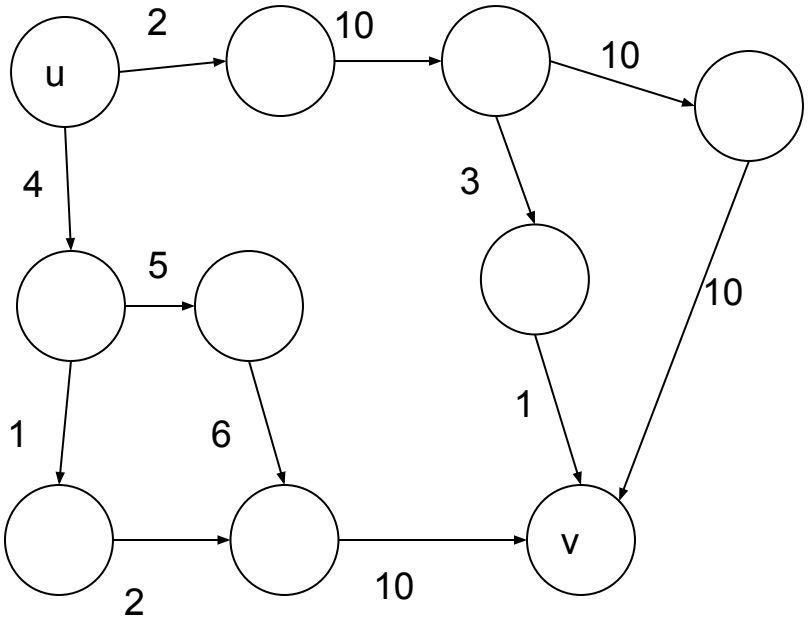
Let S be the set of explored nodes

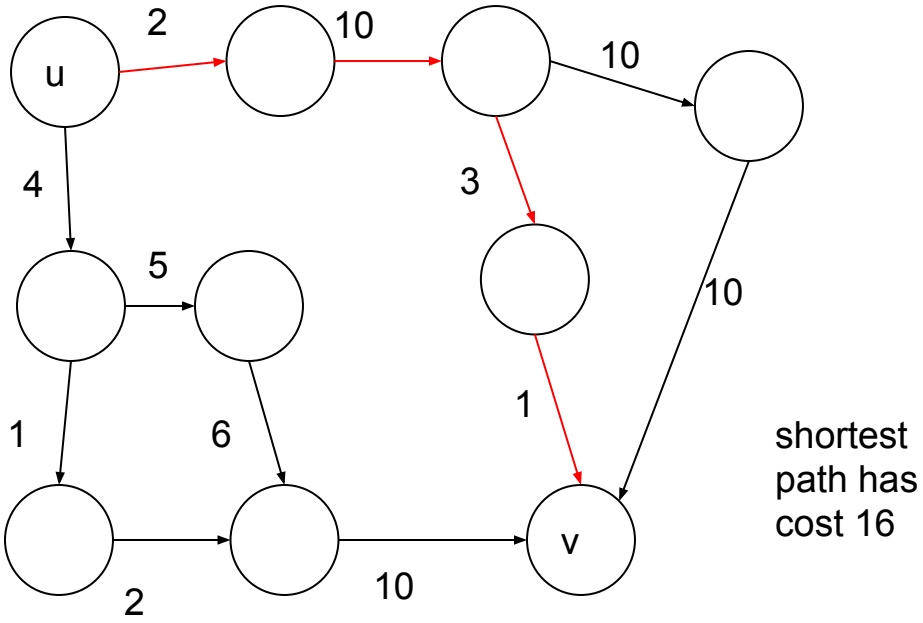
For each $u \in S$, we store a distance $d(u)$
Initially $S=\{s\}$ and $d(s)=0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S
    for which

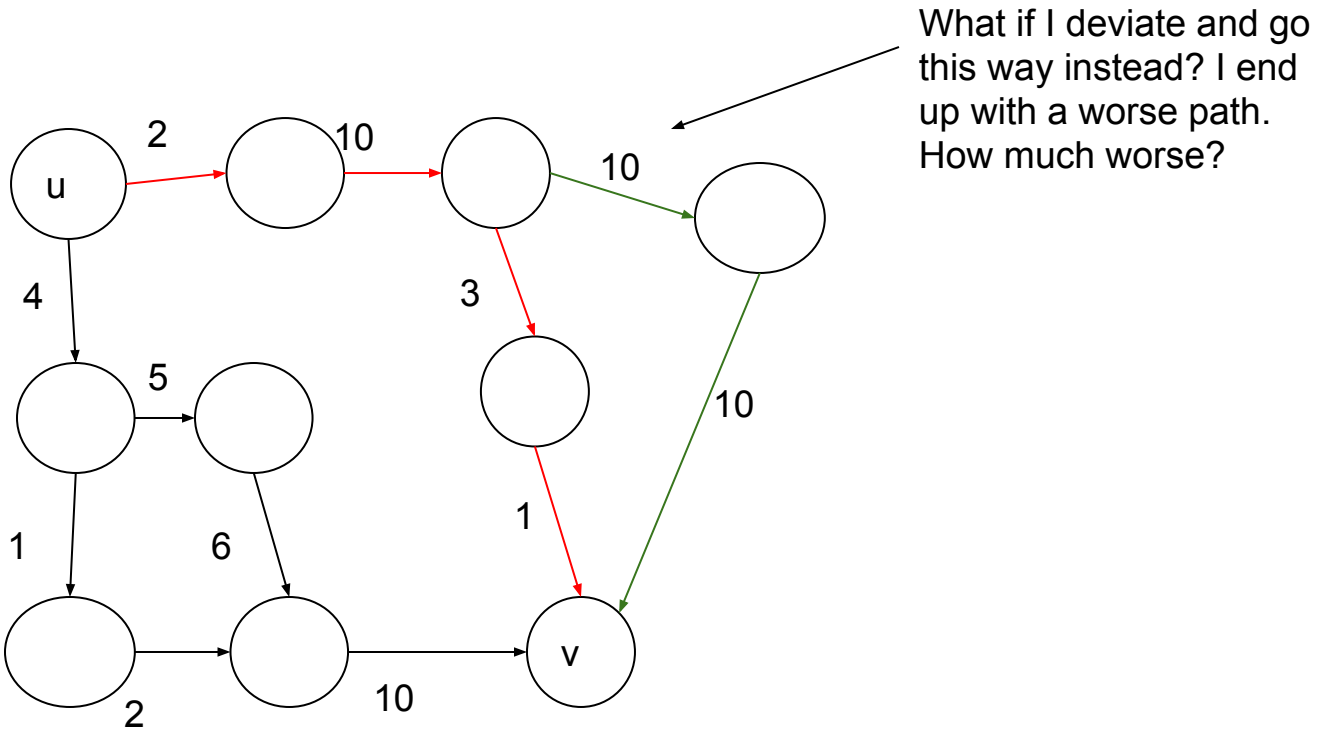
    $d'(v) = \min_{e=(u,v):u \in S} d(u) + l_e$

    is as small as possible
    Add v to S and define $d(v)=d'(v)$
EndWhile

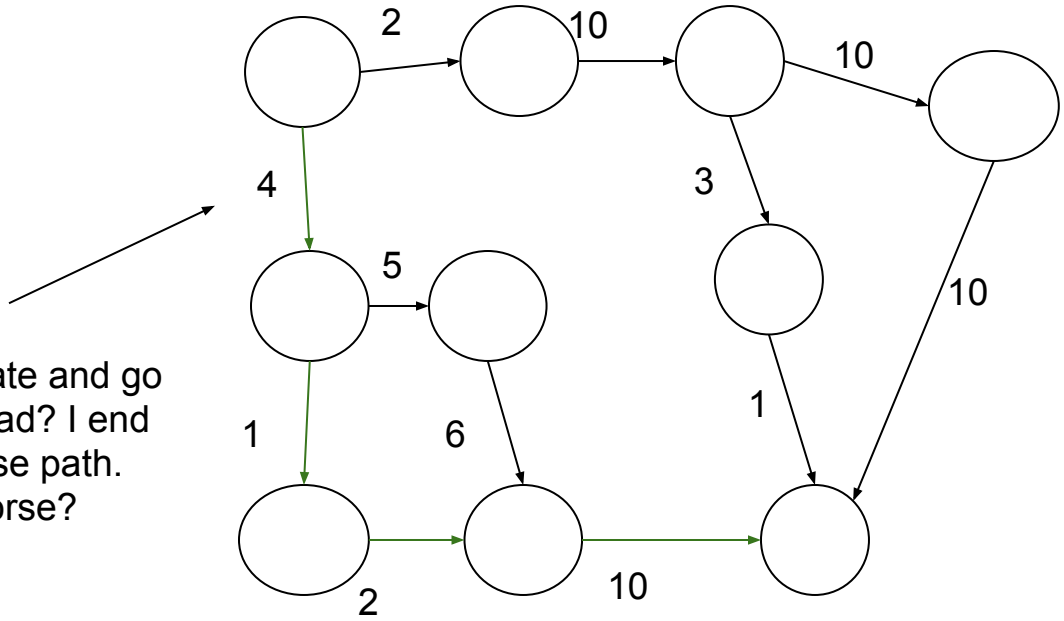
Runtime using a min priority queue with $d'(v)$ as the key:
O((E+V) log V) → simplify to O(E log V)


u
2
10
10
4
3
5
10
1
6
1
v
2
10


u
2
10
10
4
3
5
10
1
1
6
v
2
10
shortest
path has
cost 16


What if I deviate and go this way instead? I end up with a worse path. How much worse?
u
2
10
10
4
3
5
10
1
6
1
v
2
10

What if I deviate and go this way instead? I end up with a worse path. How much worse?



If, at any step along the path, we make one “mistake” or “deviation”, we won't reach our target via the shortest path.

We need to know which wrong step will make us arrive through a path that is worse than the shortest path, but still better than any other possible path.

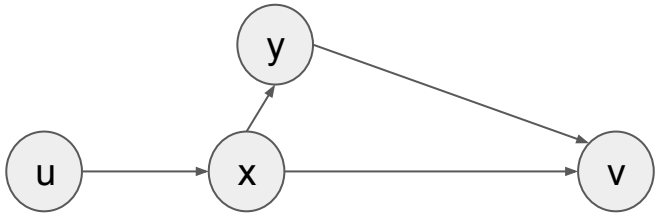
We know that we can’t make more than one mistake. That will give us a path that is worse even than the second shortest path.

So: try to find all the paths from u to v composed of a sequence of “right steps,” then one step that could be our mistake, then another sequence of “right steps.”

The shortest path can be used in order to find the second shortest path.

(And more generally, a k-1 shortest path can be used to find the k-th shortest path).

1. Calculate the shortest path P using Dijkstra.
2. Let V' be the set of vertices belonging to the shortest path.
3. For each vertex x in V' that is the not the last vertex, consider taking a single step to the side.
    - Explore all vertices y adjacent to x, where y is not on the shortest path.
4. Compute pathCost = distance(u,x) + distance(x,y) + distance(y,v).
    - distance(u,x) can be summed as you walk along the shortest path
    - distance (x,y) is the cost of a single edge
    - distance(y,v) can be calculated via Dijkstra again
5. Return the shortest of all the paths discovered in step 4.



Calculate the shortest path P using Dijkstra.
Let V' be the set of vertices belonging to the shortest path.
For each vertex x in V', consider taking a single step to the side.

Explore all vertices y adjacent to x, where y is not on the shortest path.
Compute the distance(u,x) + distance(x,y) + distance(y,v).

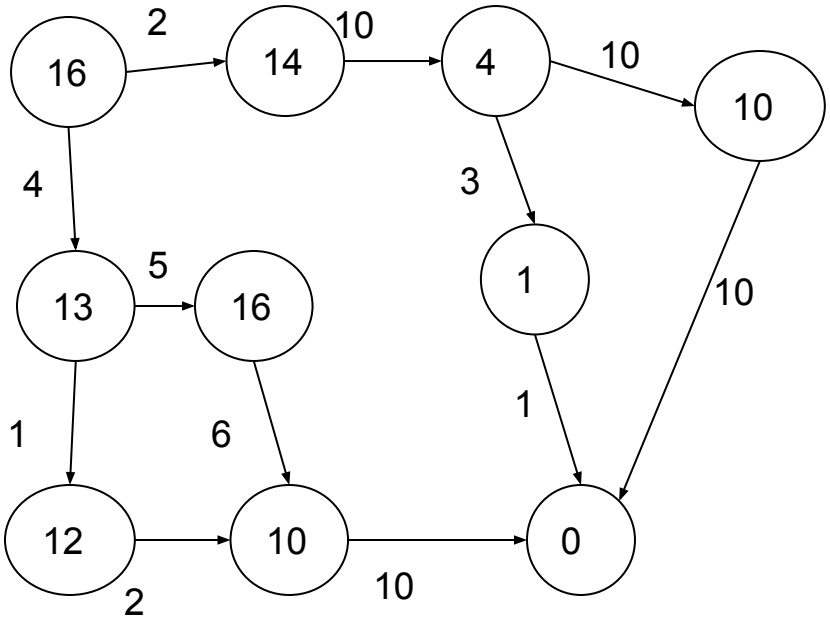
- distance(u,x) can be summed as you walk along the shortest path
- distance (x,y) is the length of a single edge
- distance(y,v) can be calculated via Dijkstra

Return the shortest of all the paths discovered in previous step.

Complexity is dominated by number of calls to Dijkstra.
First step = O(ElogV)
Any one iteration of the loop may demand the analysis of V adjacent nodes, but in aggregate only V additional calls to Dijkstra.
O(ElogV + V*ElogV) = O(VElogV)

Can we do better?

We can pre-calculate the distances from ALL vertices to the target at the start of execution.



Every node has its own shortest path to the goal.

Discussion

Depends on Dijkstra, which does not handle negative edge weights.

Generalizes to the k-shortest path problem:

- List the k (loop-free) paths connecting u and v in increasing order of length.

References

Eppstein, D. (1998). Finding the k shortest paths. SIAM Journal on computing, 28(2), 652-673.

Nagubadi, RadhaKrishna. "K Shortest Path Implementation." (2013).

Yen, Jin Y. "Finding the k shortest loopless paths in a network." *Management Science* 17.11 (1971): 712-716.