{Z, Zero, Zero'}-Path Reachability

Jae Tak Kim

1. **Thought Process:** It's clear that every Z-path must be of the form m number of -1 followed by k number of $+1$, where $m \in \mathbb{N}$. Thus, every Z-path of length $2m$ contains within it Z-paths of length $2(m-1), \ldots, 2$. My initial idea is to consecutively shrink the Z-paths from smallest to the largest. That is, to get every pair of consecutive -1 and $+1$ weight and combine them into one edge. If you have nodes a, b, c, d, e and edges $(a, b), (b, c), (c, d), (d, e)$ with $w(a, b) = w(b, c) = -1$ and $w(c, d) = w(d, e) = +1$, you would first combine edges (b, c) and (c, d) into one edge (b, d). However, you have to be sure not to remove the combined edges because there could be another Z-path if edge (f, c) was in the graph with $w(f, c) = -1$, and if you removed (c, d), the algorithm would incorrectly miss the Z-path from f to d. Thus, the idea became instead that I would create new edges instead of shrinking two entirely. The natural weight for the newly created edge is 0 since the weight of the path $p = b, c, d$ has weight 0. We could merely record this new edge in a separate list or adjacency matrix but since we know that this Z-path is a part of a longer Z-path, we want to use this information somehow to create a new edge (a, e) of weight 0. Since we added the weights of the two edges to create (b, d), we can do the same with the other edges. We can add the weights of edges (a, b) and (b, d) to create an edge (a, d) of weight $-1+0 = -1$, which would then be combined with the last remaining edge (d, e). We could have likewise first combined (b, d) with (d, e) instead and obtained the same result. When deciding the rule for creating new edges, we have to be careful not to combine cases where $(i, j), (j, k) \in E$ and $w(i, j) = +1, w(j, k) = -1$ as this is not a Z-path.

   The allowable cases out of the 9 possible are: $(-1 \to 0), (-1 \to +1), (0 \to +1)$. The last case is redundant though so we will only use the first case. Thus, every new -1 edge indicates a possible Z-path but with an additional -1. The idea of the following algorithm is that we start with the pair of -1 and $+1$ edges to create the first Z-path, and then build up from there by adding another -1 weighted edge prior to the pair. If a longer Z-path created with the 2-edge Z-path exists, then there must be a negative edge connected at the begining. These triple consequtive edges are used to create another -1 weight negative edge representing this possible Z-path, then iterates the problem again. We note that since every single Z-path must come from these original two edge pairs, we merely need to keep a worklist containing them and any new edge created until we run out of edges.

ZPaths($G = (V, E)$) :

```
1  W = list of edges in E        // worklist W
2  Initialize E' as adjacency matrix with E
3  while W ≠ ∅:
4      (i, j) = W.pop()
5      for k = v_1, ..., v_|V|:
6          // Case (−1 → +1)
7          if w(i, j) == −1 and +1 ∈ E[j][k] and 0 ∉ E'[i][k]:
8              Add 0 to E'[i][k]
9              Add (i, k) with w(i, k) = 0 to W
10
11         // Case (−1 → 0)
12         if w(i, j) == 0 and −1 ∈ E[k][i] and −1 ∉ E'[k][j]:
13             Add −1 to E'[k][j]
14             Add (k, j) with w(k, j) = −1 to W
```

The following implemenation notes and runtime analysis also equally applies to the algorithms for Zero and Zero' path reachability.

Implementation Notes: Assume that the vertices are labeled $v_1, \ldots, v_{|V|}$. Initialize a new graph represented by the "adjacency matrix" E' where each index $E'[i][j]$ is a list of edges from i to j. An empty list denotes that no edge exists between i and j. The list is a set of unique elements from the set $\{-1, 0, +1\}$ so the list has size at most 3. Each number in the list denotes an edge from i to j of that integer weight. E' is initialized with all of the edges E from the original graph which can be done in $O(|V|^2)$-time as E' is a $|V| \times |V|$ matrix where each index contains a list of constant size. Constant size lists support constant-time insertion and membership/containment operations. I will assume that E in the above algorithm is also implemented in this way without any issue. The worklist W can be implemented as a queue supporting $O(1)$ insert and pop operations. I assume that the weight function can also be computed in constant time.

Runtime Analysis: The for-loop clearly iterates $O(|V|)$ time. All of the operations within the for-loop all run in constant-time as explained in the implementation notes. For each for-loop iteration, at most two edges are added if both if-statements are true.

The outer while-loop runs as long as the worklist is nonempty. It starts with $|E|$ elements. Note that an edge is added to W only if the edge doesn't already exist in E'. Thus, at most $3 \cdot |V|^2$ edges are added to W, so the while loop runs in $O(|V|^2)$ time. For checking Z-path-reachability, you don't actually need to keep track of newly created edges with non-zero weights, but it is kept here to make the analysis simpler. This will actually be required for Zero and Zero' reachability. Overall, the algorithm has overal time complexity $O(|V|^3)$, which is equivalent to $O(|E| \cdot |V|)$ only if $E = \Omega(|V|^2)$. It may be possible to make it $O(|E| \cdot |V|)$-time for a sparse graph by using a different implemenation of edges instead of the adjacency matrix I used.

Claim: There exists a Z-path in G from node i to node j if and only if $0 \in E'[i][j]$.

Proof. The idea of the correctness proof goes as follows. Every Z-path between two nodes i and j must be of length $2m$ where $m \in \mathbb{N}$. We prove by induction on m that if a Z-path exists from i to j, 0 will be in $E'[i][j]$. Let $m = 1$. This path must be a single -1 weight

edge followed by a single $+1$ weight edge. The -1 edge will be initialized into the worklist W, and thus the case on line 6 will run. Then the algorithm will add an edge of weight 0 that connects the first node to the last to the new matrix E' so the base case runs correctly.

Now assume that a Z-path exists from i to j of length $2m$ for some $m > 1$ and that $0 \in E'[i][j]$ after the algorithm executes. We prove the induction step that this implies that if there is a Z-path of length $2(m+1)$ that contains this Z-path of length $2m$, then this will also be found by the algorithm. If $0 \in E'[i][j]$, then edge (i, j) with $w(i, j) = 0$ must have been added to the worklist as the two always go together on lines 8 and 9. Then when the worklist gets around to this element and pops it, the second if-statement must be true as edge (k, i) with $w(k, i) = -1$ will be an edge in the original graph. Then a new edge (k, j) will be added to the worklist that connects edge (k, i) with the Z-path (i, j). Then this (k, j) will be popped from the worklist and the rightmost $+1$ edge required to form the Z-path of length $2(m+1)$ will be added by the first if-statement on line 7, adding the full Z-path to E'. This proves the induction.

Now suppose that $0 \in E'[i][j]$. Then there must be a Z-path from i to j in G. The intuition is that every edge (both the original edge and the new edges) is a path with the weight of that edge that can still be used to create potential Z-paths, that is, still has the property that any positive edge appears after all negative edges. Clearly, this is true for all of the original edges as they are paths of length 1 and therefore also vacuously have the above property.

Consider the new edges with weight -1. The very first edges of this kind is created from the very first 0 weight edge created with an original edge of weight -1. The very first 0 weight edges are likewise created from the original edges (since no other edges exist at that time). Thus, the very first 0 weight edges must be of the form (i, k) where $(i, j), (j, k) \in E$ and $w(i, j) = -1, w(j, k) = +1$, and therefore have the property that any positive edge appears after all negative edges. Then it follows from this that the first new edges of weight -1 must also have this property since the paths have edge weights $-1, -1, +1$. These new -1 edges act exactly like the single original edge of weight -1 when being combined with a $+1$ edge. Note that these new -1 edges are never added to a 0 weight edge as per the if-statements as they would not hold the property $(-1, -1, +1 \to -1, +1)$. □

The key to getting this runtime is that you're limiting the number of potential new edges by keeping the weights of the edges to a constant number $\{-1, 0, +1\}$. I tried keeping track of all of the possible weight paths but that's too expensive even if there's no cycles (if there is, then this is not possible in polynomial time).

Negative Cycles. Lastly, I wanted to briefly address negative cycles as it was specifically mentioned in the problems. Unlike with shortest path algorithms or similar algorithms, negative cycles do not cause issues as new edges are not created from two consecutive -1 edges. Also, if a Z-path includes the use of a negative cycle, this is correctly caught by the algorithm as it iterates between new 0 weight and -1 weight edges as it goes around the cycle until it exhausts the $+1$ edges to the right (this is a vague statement but I'm pretty sure negative cycles aren't a problem).

2. The algorithm for solving Zero-path reachability is very similar to Z-path reachability above. There's less restrictions on the paths for Zero-paths so instead of only adding a single edge to a path at a time, we can also combine two paths together into one. Further, we don't need to look at the edge before and after the edge (i, j) from W as positive edges can appear before

negative ones. Thus, if two edges are to be combined at some point, we merely only need to look at one of those to find the other.

ZeroPaths($G = (V, E)$) :

```
1  W = list of edges in E        // worklist W
2  Initialize E' as adjacency matrix with E
3  while W ≠ ∅:
4      (i, j) = W.pop()
5      for k = v_1, ..., v_|V|:
6          // Cases (0 → +1) and (−1 → +1)
7          if +1 ∈ E'[j][k] and w(i, j) ∈ {−1, 0} and w(i, j) + 1 ∉ E'[i][k]:
8              Add w(i, j) + 1 to E'[i][k]
9              Add (i, k) with w(i, k) = w(i, j) + 1 to W
10
11         // Cases (−1 → 0), (0 → 0) and (+1 → 0)
12         if 0 ∈ E'[j][k] and w(i, j) ∈ {−1, 0, +1} and w(i, j) ∉ E'[i][k]:
13             Add w(i, j) to E'[i][k]
14             Add (i, k) with w(i, k) = w(i, j) to W
15
16         // Cases (0 → −1) and (+1 → −1)
17         if −1 ∈ E'[j][k] and w(i, j) ∈ {0, +1} and w(i, j) − 1 ∉ E'[i][k]:
18             Add w(i, j) − 1 to E'[i][k]
19             Add (i, k) with w(i, k) = w(i, j) − 1 to W
```

Proof Idea The proof for Zero-path reachability is very similar to the one for Z-path but with more cases to think through. Note that unlike in the Z-path algorithm, we check whether the new edge we're adding onto the edge from W is in E', not E. This is because a Zero-path with an additional $+1$ or -1 followed by a Zero-path is still another potential Zero-path with an additional $+1$ or -1. That is, we can safely add paths together that create new viable paths. This was not the case in Z-paths as connecting two Z-paths together does not result in a Z-path.

Zero-path algorithm allows for combining every pair of edges except for two -1 or two $+1$ as the new integer weight would increase time and space complexity. □

3. Zero'-path reachability is also similar to both Z and Zero path reachability problems. The logic is similar to the Zero-path reachability except that there's a little more restriction on what intermediate paths are allowed. A Zero'-path must always have every prefix have a non-negative weight. This already forces the first edge in a Zero'-path to be a $+1$.

```
ZERO'PATHS(G = (V, E)) :
 1  W = list of edges in E        // worklist W
 2  Initialize E' as adjacency matrix with E
 3  while W ≠ ∅:
 4      (i, j) = W.pop()
 5      for k = v_1, ..., v_|V|:
 6          // Case (0 → +1)
 7          if +1 ∈ E'[j][k] and w(i, j) == 0 and +1 ∉ E'[i][k]:
 8              Add +1 to E'[i][k]
 9              Add (i, k) with w(i, k) = +1 to W
10
11          // Cases (0 → 0) and (+1 → 0)
12          if 0 ∈ E'[j][k] and w(i, j) ∈ {0, +1} and w(i, j) ∉ E'[i][k]:
13              Add w(i, j) to E'[i][k]
14              Add (i, k) with w(i, k) = w(i, j) to W
15
16          // Case (+1 → −1)
17          if −1 ∈ E'[j][k] and w(i, j) == +1 and 0 ∉ E'[i][k]:
18              Add 0 to E'[i][k]
19              Add (i, k) with w(i, k) = 0 to W
```

Proof Idea Again, the proof argues along very similar lines to the ones above so I'll just talk about the logic of the different cases for adding new edges. First, no -1 edge followed by another edge can be combined together because any path that starts with a -1 edge is not a Zero'-path (the first edge is a prefix and doesn't meet the requirements). Second, a 0 path edge followed by a -1 edge can't be combined for the obvious reason that this path has weight below 0. All of the other cases are allowed as the prefix sum never falls below 0. The proof makes the similar argument that all of the new edges denote paths having the property that all prefix of each path is always non-negative, so clearly any two paths with this property can be combined to create another path with the same property. □