Algorithm Design and Analysis(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Assignment 4
SPFA with Tarjan's quick disassembly trick

Mengxuan Wu

Psuedocode

Algorithm 1: SPFA with Tarjan's quick disassembly trick

Input: Graph $G = (V, E)$, s, t
Output: Shortest path from s to all other vertices and whether there is a negative cycle

Function `SPFA`(G, s, t)**:**

```
foreach v ∈ V do
    dist[v] ← ∞
    successor[v] ← null
end
dist[t] ← 0
for i = 1 to |V| − 1 do
    preprocess the preorder traversal of the Shortest Path Tree
    foreach v ∈ V do
        dormant[v] ← false
    end
    foreach w ∈ V do
        if dist[w] is updated in previous iteration and dormant[w] = false
         then
            foreach (v, w) ∈ E do
                if dist[v] > dist[w] + C_vw then
                    dist[v] ← dist[w] + C_vw
                    successor[v] ← w
                    if disassemble(v, w) then
                        return true
                    end
                    addChild(v, w)
                end
            end
        end
    end
    if no dist[v] is updated in this iteration then
        return false
    end
end
return false
```

Algorithm 2: Disassemble

Input: Subtree root v, Parent w
Output: Whether there is a negative cycle
Data: A preprocessed preorder traversal of the Shortest Path Tree of current iteration

Function `disassemble`(v, w)**:**

```
prev ← v.prev
currentLevel ← v.level
dormant[v] ← true
curr ← v.next
while curr.level > currentLevel do
    if curr = w then
        return true
    end
    dormant[curr] ← true
    curr ← curr.next
end
curr.prev ← prev
prev.next ← curr
```

Core idea

The edges we specified with *successor* are the edges that are used to represent the Shortest Path Tree (Once the algorithm finishes, we can find the shortest path following these edges). This tree's root will be t, and the edges are directed from the children to the parent. With such a setting, if we update the distance of a vertex v in the i-th iteration, then we know all of its children are going to be updated in the $i+1$-th iteration. This means updating these children in the i-th iteration is unnecessary, and we can skip them.

It also gives a convenient way to detect negative cycles. When we update the distance of a vertex v, we are adding an edge (v, w) to the Shortest Path Tree. If w is one of the children of v, then there exists a cycle in the Shortest Path Tree. The figure below shows the two cases when adding an edge (v, w) to the Shortest Path Tree and removing the previous edge (v, t).

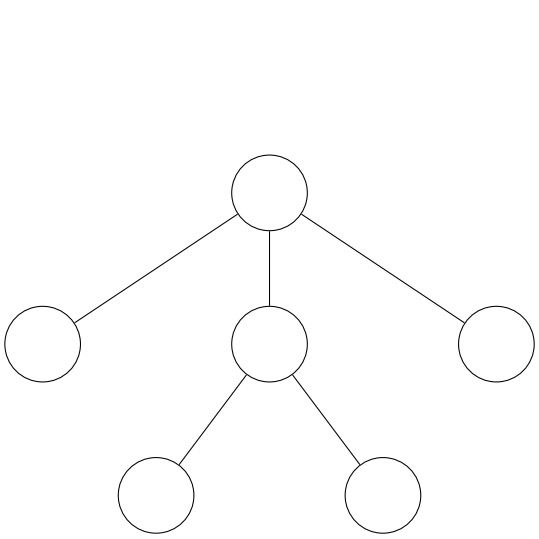
(a) Will not cause negative cycle (w is not a child of v) (b) Will cause negative cycle (w is a child of v)

Since there is a cycle, there will be an infinite loop with our algorithm and thus there must be a negative cycle. Taking the same example, since v is updated in the i-th iteration, then v will be checked in the $i+1$-th iteration and s will be updated. This will cause w to be updated in the $i+2$-th iteration, and v will be updated again in the $i+3$-th iteration. This will be an infinite loop.

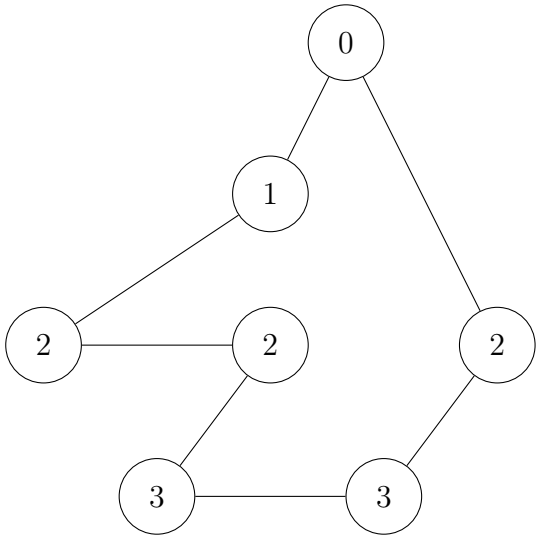
Combining these two ideas, each time we update the distance of a vertex v, we can scan all its children. If we find w, then we know there is a negative cycle. If we don't, we mark all of them as dormant, and skip them in the current iteration.

Critical Process

The critical process of the algorithm is the disassemble function. I will use an example from reference [1] to illustrate the process.



(a) Shortest Path Tree



(b) Preorder traversal

We use an extra dummy vertex v_0 to avoid the special case when all the vertices are dormant. The preorder traversal of the Shortest Path Tree help we define the subtrees by the level of the vertices. For each vertex v, all the vertices in the subtree rooted at v will have a level greater than v. Also, this traversal will help us find the children of a vertex. Since all the vertices are in a linked list, we can find the children of a vertex by scanning the linked list.

Complexity Analysis

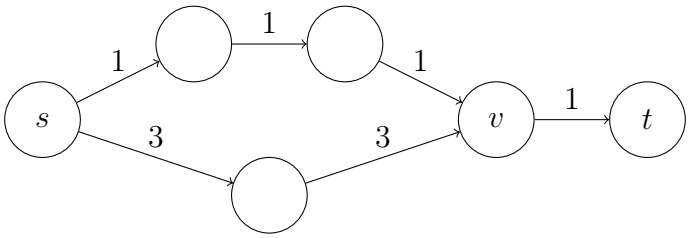
Time Complexity

Since the disassemble function is traversing the Shortest Path Tree, in the worst case it will traverse all the vertices in the tree. Since each node at most be marked as dormant once in a single iteration, the disassemble function will take $O(n + m)$ time. The rest of the algorithm is the same as the normal SPFA algorithm. Hence, the time complexity of each iteration is $O(m)$. The SPFA algorithm will run for $n - 1$ iterations. Overall, the time complexity is $O(mn)$.

Space Complexity

The space will be required to store the distance, successor, and dormant status of each vertex. Also, we need space to store the graph. Hence, the graph storage will be the dominating factor, and the overall space complexity is $O(m + n)$.

Example



In this example, the shortest path from s to t is the upper path with length 4. However, when we run the normal SPFA algorithm, we will get the lower path with length 7 first, then be corrected to the upper path in the next iteration. This is because the lower path is shorter than the upper path.

With the SPFA algorithm with Tarjan's quick disassembly trick, we will mark the node s as dormant when we find the lower path. Hence, the distance of s will not be updated in current iteration. This will skip one update operation. And in the next iteration, we will find the upper path and make the correct update in the next next iteration.

As this example shows, the SPFA algorithm with Tarjan's quick disassembly trick can avoid unnecessary updates and thus reduce the number of iterations, even when the graph contains no negative cycles.

References

[1] S. Lewandowski, "Shortest paths and negative cycle detection in graphs with negative weights - i: The bellman-ford-moore algorithm revisited," 09 2010.