

CS216 Assignment4

Description of the subtree disassembly trick proposed by Tarjan

匡亮 (12111012)

May 1, 2023

Abstract

This is my report of CS216 Assignment4, including a description of the subtree disassembly trick proposed by Tarjan:

1. Its core idea and critical procedures.
2. Running time and space complexity analysis.
3. Its pseudocode.
4. How it speeds up SPFA.

Contents

1	Core idea and critical procedures of Tarjan’s trick	3
2	Running time and space complexity analysis	3
3	Pseudocode	4
4	How it speeds up SPFA	4

1 Core idea and critical procedures of Tarjan's trick

For convenience, in the whole report we assume the terminus vertex t is reachable from every vertex in G , and our algorithm will immediately terminate if we find a negative ring (though some vertices cannot reach it, which means they still have a valid non-infinite minimum distance to the terminus vertex). Under these conditions, Tarjan's trick can significantly speed up SPFA without change the algorithm a lot.

Here I don't repeat what we have learned in our courses (like how SPFA works) and begin from SPFA.

In SPFA, we maintain an array *first* for each vertex, and check whether there is a ring after n iterations to determine whether there is a negative ring. However, if we have found a ring during the iterations, we can immediately terminate the algorithm and report there is a negative ring. Therefore, the basic idea of this trick is try to detect whether there is a ring when modifying *first* with an acceptable extra cost.

Obviously, before we find a ring, $(v, first[v]), \forall v \neq t$ is a tree with root t . When we set $first[v] \leftarrow w$, we can do DFS in v 's subtree and check whether w is in v 's subtree, if so, we can terminate our algorithm. However, this DFS will cost an extra $O(n)$ time, so we should expect it to do more for us. We can further observe that before the modification, the minimum distance of all vertices in v 's subtree (call them x) is evaluated based on the minimum distance of v , which has been just modified. Therefore, the minimum distance of all of them are out of date. We can set them **dormant**, then they will not be used to update others' minimum distance before their own minimum distance are updated. At the same time we can remove them from the tree (set $first[x] \leftarrow null$), so that they will not be set dormant by other ancestors again. They will add themselves back to the tree automatically when their minimum distance are updated (because they will update $first[x]$ at that time).

2 Running time and space complexity analysis

Based on SPFA, we just store one more tree with $O(1)$ space per vertex, so the space complexity is still $O(n)$.

Add/Remove a vertex to/from the tree can be easily done in $O(1)$ time with linked-list. Adding only happens when $first[v]$ are updated for some v , and removing happens no more than adding because vertices will not be removed twice, so the time complexity is still $O(nm)$.

3 Pseudocode

	Algorithm 1: SPFA Improved by Tarjan's Trick
	Data: G, t
1	$n = \text{number of nodes in } G;$
2	Array $M[V];$
3	Initialize $M[t] = 0$ and $M[v] = \infty$ for all other $v \in V;$
4	Initialize $first[t] = t$ and $first[v] = \text{null}$ for all other $v \in V;$
5	Initialize a tree with only root vertex $t;$
6	$foundNegativeRing = \text{False};$
7	for $i = 1, \dots, n - 1$ do
8	for $w \in V$ where w is not dormant do
9	if $M[w]$ has been updated in the previous iteration then
10	for all edges (v, w) do
11	$M[v] = \min(M[v], c_{vw} + W[w]);$
12	if this change the value of $M[v]$ then
13	$first[v] = w;$
14	for all vertex x in v 's subtree do
15	if $x == w$ then
16	$foundNegativeRing = \text{True};$
17	End the algorithm;
18	end
19	Set x dormant;
20	Remove x from the tree;
21	end
22	Set v not dormant;
23	Add v to the tree as w 's child;
24	end
25	end
26	end
27	end
28	if no value changed in this iteration then
29	End the algorithm;
30	end
	/* If $foundNegativeRing$ is true, then users should ignore the returned $M, first.$ */
31	end
	Result: $foundNegativeRing, M, first$

4 How it speeds up SPFA

If there is a negative ring, the algorithm will terminate in advance, so it is speeded up without a doubt. However, if there is no negative rings, this trick can still speed SPFA up.

References

[Jon Kleinberg / Éva Tardos(2005)] Algorithm Design (P.304 - 307)

[Stefan Lewandowski(2010)] Shortest Paths and Negative Cycle Detection in Graphs with Negative Weights - I: The Bellman-Ford-Moore Algorithm Revisited <https://d-nb.info/1014960916/34>