CS170–Fall 2022 — Homework 3 Solutions

Deming Chen cdm@pku.edu.cn

Last Modified: October 4, 2022

Collaborators: NONE

2 Updating Labels

The algorithm is based on DFS. We do not need to record `pre` numbers and `post` numbers of all points. We record the label of points in the path from the root to the currently visiting point.

Algorithm 1. Computing new labels

```
procedure explore(T, A)
Input:  T is a tree with root r, and all its nodes have a label l(v); A
is an global array which contains s records.
Output:  all labels are updated.

Add the label of the root into the back of A
For each child tree T' of the root node:
    explore(T', A)
Update l(r) to be (s − l(v))-th record of array A (if s < l(v), the new label
is set to be the zeroth record of array A).
Remove the last record from array A.
```

In this algorithm, each node in the tree is visited once. When a node is being visited, update its label takes a constant time. As a result, the runtime of this algorithm is $\mathcal{O}(n)$.

3 Where's the Graph?

(a) The problem can be represented by the graph in Figure 1 (in fact, it is a tree). Every node has 6 branches. The method to get 2022 is BFS. From the root of x, when first getting 2022, that is just the shortest path.

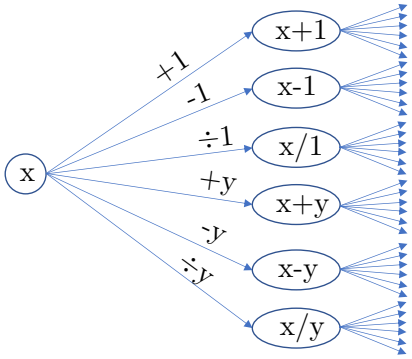

Figure 1: The graph to solve the problem to get 2022 from x using 1 and y in the shortest path.

(b) Construct a graph $G(V, E)$ s.t. V represents all species of Pokemen. $(u, v) \in E$ if and only if v directly descended from u, and Since all species are descended from the original Mew, the graph is actually a tree with root Mew. We can carry out DFS from the root Mew and stored the `pre` numbers and `post` numbers of all vertices.

For two inputs a and b, if $\texttt{pre}(a) < \texttt{pre}(b) < \texttt{post}(b) < \texttt{post}(a)$, b is descended from a. If$\texttt{pre}(b) < \texttt{pre}(a) < \texttt{post}(a) < \texttt{post}(b)$, a is descended from b. Otherwise, a and b share a common ancestor, but neither are descended from each other.

(c) Construct a graph $G(V, E)$ s.t. V represent all boxes Bob have, and there exists an edge from u to v if and only if box u can fit into box v (that is, the size of box u is less than 15 smaller than box v). For an edge $(u, v) \in E$, its weight equals to the weight of box v. Since no larger box can fit into a smaller box, the graph is a directed acyclic graph (DAG).

To find the lightest sequence of boxes that can fit in each other, we just need to find a shortest path from vertex x.

4 The Greatest Roads in America

Let $R = \{(u_1, v_1), (u_2, v_2), \cdots, (u_r, v_r)\}$. We construct a graph $G' = (V', E')$ with $(2kr + 2)$ vertices: $s, t, u_i^{(j)}, v_i^{(j)}$ $(i = 1, 2, \cdots, r, j = 1, 2, \cdots, k)$. Edges in G' are $(s, u_i^{(1)})$ $(i = 1, 2, \cdots, r)$, $(u_i^{(j)}, v_i^{(j)})(i = 1, 2, \cdots, r, j = 1, 2, \cdots, k)$, $(v_i^{(j)}, u_i^{(j+1)})$ $(i = 1, 2, \cdots, r, j = 1, 2, \cdots, k-1)$ and $(v_i^{(k)}, t)$ $(i = 1, 2, \cdots, r)$. Each edge has a weight w:

$$\begin{aligned} w(u_i^{(j)}, v_i^{(j)}) &= d(u_i, v_i) \text{ (recall } (u_i, v_i) \in R \subseteq E); \\ w(s, u_i^{(1)}) &= \text{the length of the shortest path from } a \text{ to } u_i \text{ in } G; \\ w(v_i^{(j)}, u_i^{(j+1)}) &= \text{the length of the shortest path from } v_i \text{ to } u_i \text{ in } G; \\ w(v_i^{(k)}, t) &= \text{the length of the shortest path from } v_i \text{ to } a \text{ in } G. \end{aligned} \tag{1}$$

G' is a directed acyclic graph. Any path from s to t corresponds to a path of the same length from a to itself in G containing at least k roads in R. The distance from s to t in G' is just the length of the shortest path that hits at least k of those amazing roads in G. So, the algorithm is:

Algorithm 2. Find the length of the shortest path containing k amazing roads

`procedure travel`$(G = (V, E, d), R, k)$
`Input:` $G = (V, E, d)$ `is a directed weighted graph;` $R \subseteq E$; `and` k `is an positive integer`
`Output: the length of the shortest path that containing` k `edges in` R

`Construct a new a directed weighted graph` G' `according to equation (1).`
`Find the shortest path in` G' `from` s `to` t `and return its length.`

We firstly prove that this algorithm is correct. All paths from a to itself in G containing at least k roads in R (let's call such paths amazing paths) can be divided into r^k groups labeled by $(x_1, x_2, \cdots, x_k) \in \{1, 2, \cdots, r\}^k$. If one amazing path passes through (u_{x_1}, v_{x_1}), (u_{x_2}, v_{x_2}),$\cdots$, (u_{x_k}, v_{x_k}), it will be classified in group $(x_1, x_2, \cdots, x_k)$ (if more than k amazing roads are included in a amazing path, just consider the first k roads in the path). In each group there is the shortest path. Its length equals to the length of path from s to t in G' through $(u_{x_1}^{(1)}, v_{x_1}^{(1)})$, $(u_{x_2}^{(2)}, v_{x_2}^{(2)})$, $\cdots$, $(u_{x_k}^{(k)}, v_{x_k}^{(k)})$. So, the length of the shortest amazing path in G equals to the length of the shortest path of the shortest paths in all path groups, that is the shortest path from s to t in G'.

The runtime analysis is as follows. At the first step, we need to compute the weights of all edges in G'. We can call Dijkstra's algorithm for a and u_i $(i = 1, 2, \cdots, k)$, which takes $\mathcal{O}[(r + 1)(m + n) \log n]$ time. At the second step, we need to compute the shortest path in a directed acyclic graph G', which takes $\mathcal{O}(kr^2)$. Since $r = \mathcal{O}(m)$, the total runtime is

$$\mathcal{O}[m(m + n) \log n + km^2]. \tag{2}$$

5 Detecting Cyclic Vertices

The algorithm is simple. First decompose the graph into its strongly connected components. Then return the vertices belonging to components that contain more than one vertex.

The run-time of this algorithm is $\mathcal{O}(m+n)$, as analysed in the textbook.

Now we prove that the algorithm is correct. That is, a vertex is contained by a circle in G if and only if it belongs to a strongly connected component which has more than one vertex. If a vertex v belongs to a strongly connected component which contains another vertex u, there exist a path from v to u, and a path from u back to v. That is just a circle. If a vertex v is in a circle, let u be a vertex distinct from v in the circle, then u and v are in the same strongly connected component.