An Improved Thorup Shortest Paths Algorithm with a Modified Component Tree

Yusi Wei
Information Systems Course, Interdisciplinary Graduate School of Science and Engineering
Shimane University
Matsue, Japan

Shojiro Tanaka
Information Systems Division, Interdisciplinary Graduate School of Science and Engineering
Shimane University
Matsue, Japan

Abstract—**This paper provides an improved Thorup algorithm which modified the component tree of the original Thorup algorithm to make it able to maintain the tentative distance of each vertex without the unvisited structure. According to the experimental result, our algorithm showed a better result than the original Thorup algorithm and Fibonacci-based Dijkstra algorithm in practice. By comparison, the time cost of query is reduced by 75.04% and 58.26%, respectively.**

Keywords: Dijkstra; Single-Source Shortest Path; Undirected Weights

I. INTRODUCTION

The single source shortest path (SSSP) is the problem of finding the shortest path from a source vertex to every other vertex. It has been applied in many fields such as navigation [1], keyword searching [2], computer network [3], and it is widely used for finding the optimal route in a road network. SSSP problem is described as the following throughout this research. Given a graph $G = (V, E)$ and a source vertex $s \in V$, suppose s can reach each vertex of the graph, then find the shortest path from s to every vertex $v \in V$, in which V and E represent the vertices and edges of G [4]. The m and n mentioned in the rest of this paper represent $|E|$ and $|V|$, respectively. Use $\mathrm{D}(v)$ to represent the tentative distance from the source vertex to v, and use $\mathrm{d}(v)$ to represent the ensured shortest distance from the source vertex to v, and let $\mathrm{L}(v, w)$ represent the positive integer weights of edge(v, w). At the beginning, $\mathrm{D}(v) = \infty$ for every vertex except the source vertex, $\mathrm{d}(s) = 0$. The length of a shortest path should be the sum of the weights of each edge of the shortest path.

One of the most influential shortest path algorithms is Dijkstra [5][4][6] which is proposed in 1959. Yen's paper [7] is considered to be the first one which implements Dijkstra with an array. In detail, an array is used to record the tentative distance of each vertex which is adjacent to visited vertices. Every time when a vertex is visited, the distances of the vertices which are adjacent to the vertex will be recorded in an array, and then, the vertex which has the shortest distance will be taken to try to relax the vertices which are adjacent to this vertex. The word relax is described as follows. A vertex, say A, can relax another vertex, say B, means the distance from the source vertex to B can be shortened through A. It takes $\mathrm{O}(1)$ time to insert a relaxed vertex and $\mathrm{O}(n)$ time to delete a vertex from an array. To relax all vertices, Dijkstra algorithm runs in $\mathrm{O}(m)$ plus the time of maintaining the array, overall, it takes $\mathrm{O}(m + n^2)$.

Thorup [8] is an algorithm which theoretically proved that solving the SSSP problem in linear time with pre-processed index. The paper proposed a hierarchy- and buckets-based algorithm to pre-process indices for performing queries in undirected graphs with non-negative weights. Theoretically, this algorithm constructs the minimum spanning tree in $\mathrm{O}(m)$, constructs the component tree in $\mathrm{O}(n)$, constructs unvisited data structure in $\mathrm{O}(n)$, and calculates distance of all vertices based on constructed structures in $\mathrm{O}(m + n)$. But in practice, due to the difficulty of implementation, Thorup algorithm occasionally does not perform as expected according to the experimental result provided by Asano and Imai [9], and Pruehs [10]. This paper proposes an alternative to accelerate Thorup algorithm by reducing the structures used to maintain the tentative distance of each vertex.

II. RELATED WORKS

Asano and Imai [9], and Pruehs [10] implemented Thorup algorithm with their modifications. Asano and Imai [9] uses an array to realize the function of atomic heaps, which is used in the Thorup algorithm; uses the union with size and find with path compression algorithm instead of the union and find algorithm [11] to construct the component tree, and designs a three-levels tree as an unvisited data structure to maintain the tentative distance of each vertex.

Pruehs [10] uses Kruskal's algorithm [12] to generate minimum spanning tree, and uses Tarjan's union-find algorithm [14] to construct the component tree. Then it uses Gabow's Split-findmin data structure [11] instead of atomic heaps to maintain the tentative distance of each vertex.

Hagerup [16] proposed an improved hierarchy-based algorithm which theoretically solves SSSP problem in $\mathrm{O}(m \log\log C + n \log\log n)$, and also solves all-path shortest path (APSP) problem in $\mathrm{O}(mn + n^2 \log\log n)$, where C represents the maximum edge weights. Pettie [17] proposed a

978-1-4673-4713-6/10/$26.00 ©2013 IEEE

hierarchy-based algorithm which theoretically solves APSP problems in O ($mn + n^2 \log\log n$), and proved that no SSSP algorithm can solve the problem in $\Omega(m + n \log n)$.

III. THORUP ALGORITHM

Thorup [8] is a hierarchy- and buckets-based algorithm which preprocesses indices for performing queries in undirected graphs with non-negative weights. It consists of two phases: construction and visiting. In the first phase, it constructs a minimum spanning tree [13], and then constructs a component tree in linear time based on the constructed minimum spanning tree. Each node of the component tree is a component that includes vertices. Leaf-nodes always contain only one vertex. A component is created by the connected vertices, in which their weights are smaller than 2^i, where i represents an integer that increases from 0 to $2^i \geq$ the largest weights of G. In the second phase, the tentative distance of each reached component will be mapped to its parent's buckets, for deciding the order to visit vertices in different components in a hierarchy structure. This method will overcome the sorting bottleneck of priority queue-based algorithms. That is, when trying to get the vertex which has the smallest tentative distance, no method can sort the distances in linear time currently. This problem reduces the performance of Dijkstra algorithm every time when getting the vertex from a priority queue. An unvisited data structure is used to maintain the tentative distance of each vertex. It is not limited to a specific structure, and any structure that can retrieve the smallest value from numerous values efficiently can be used. Atomic heaps and Split-findmin structure are two candidates given in Thorup algorithm for the calculation.

Fig.1 to Fig.5 gives an example of the component hierarchy and component tree which are invented by Thorup. Fig. 1 shows a graph with five vertices. The component hierarchy of the graph is given in Fig. 2. In the figure, the identification of a component is set to the smallest ID among all vertices in the same component plus the level of the component. Please note the level is counted from the bottom. For instance, "[1]2" represents the component on level 2 of the component hierarchy in Fig. 2. It includes vertices with ID 1 and ID 2, because 1 is the smallest ID, so use "[1]" to represent this component. As the example shown in Fig. 3, for enhancing the performance, the component hierarchy is compressed to the component tree by pruning component $[v]i = [v](i\text{-}1)$, where i represents the level of component v. Fig. 4 and Fig. 5 show the components of the graph on level 1 and level 2 in the Fig. 2, respectively.

Since Thorup algorithm does not construct indices based on any source vertex, the constructed indices can be used to calculate with different source vertex without reconstruction. The atomic heaps used by Thorup's algorithm is defined for the number of vertices more than $2^{12^{20}}$. Thorup [8] gives an alternative to avoid using atomic heaps by using split-findmin algorithm [11]. This changing reduces the algorithm's cost to $O(\log C + \alpha(m,n)m)$, where C represents the maximum edge weights, and α is an inverse function of Ackermann which grows very slowly. If the number of vertices is less than 10^{80}, $\alpha(m,n)$ should be equal to, or less than 4 [18].

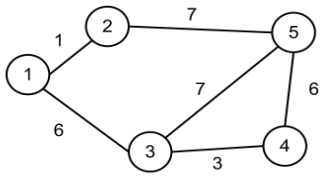

Figure 1. A graph with five vertices.

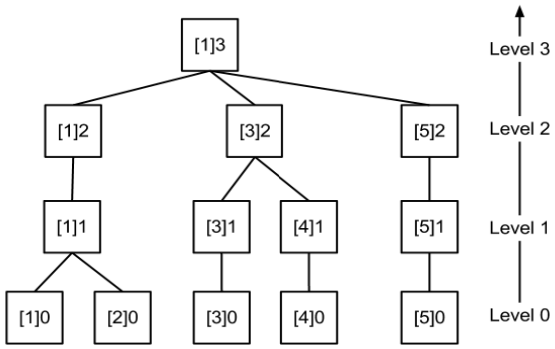

Figure 2. The component hierarchy of Fig. 1.

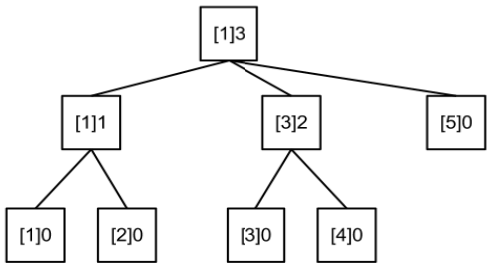

Figure 3. The component tree of Fig. 1.

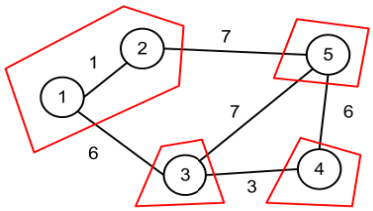

Figure 4. The components of graph on level 1.

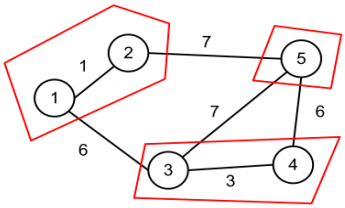

Figure 5. The components of graph on level 2.

IV. IMPROVED THORUP ALGORITHM

In this Section, an improved Thorup algorithm is proposed which can maintain the tentative distance of each vertex with the component tree itself, so as to avoid using the unvisited data structure. This change has two benefits as follows.

1. Save the time cost to construct unvisited structures.

2. Accelerate the process of calculating shortest paths.

Two variables should be added to each node of a component tree.

1. *distance*, which is used to record the tentative distance of each component.
2. *deleted*, which is used to record that if the tentative of a component is not needed to be updated. It happens when we start bucketing the component's children.

The followings are the improved algorithms starting from the one named Visit, which were in the original Thorup paper.

```
1.  Visit(v)
2.  if v is a leaf node of component tree then
3.      VisitLeaf(v)
4.      Remove v from the bucket of v's parent
5.      return
6.  end if
7.  if v has not been visited previously then
8.      Expand(v)
9.      v.ix = v.ix0
10. end if
11. repeat until v has no child or v.ix >> j-i is increased
12.     while the bucket B[v.ix] is not empty
13.         let wh equal to the node in bucket B[v.ix]
14.         Visit(wh)
15.     end while
16.     v.ix = v.ix+1
17. end repeat
18. if v has any child then
19.     move v to bucket B[v.ix>>j-i] of v's parent
20. end if
21. if v do not has child and v is not the root of the component tree then
22.     remove v from bucket of v's parent
23. end if
```

Figure 6. Algorithm of visit.

```
1.  Expand (v)
2.  v.ix0 = v.distance >>i-1
3.  v.deleted = TRUE
4.  for each child wh of v
5.      store wh in bucket B[wh.distance >>i-1)]
6.  end for
```

Figure 7. Algorithm of Expand.

```
1.  VisitLeaf(v)
2.  for each vertex w connected with v, if v.distance + L(v, w) < w.distance
3.      Let wh be the unvisited root of leaf w
4.      Let wi be the unvisited parent of wh
5.      Decrease(w, v.distance +L(v,w))
6.      if this decreases wh.distance >> i-1 then
7.          Move wh to bucket B[wh.distance >> i-1] of wi
8.      end if
9.  end for
```

Figure 8. Algorithm of VisitLeaf.

```
1.  Decrease (v, newValue)
2.  if v.distance > newValue and v.deleted !=TRUE then
3.      v.distance = newValue
4.      let n to be the parent of v
5.      while n.deleted !=TRUE and n.distance >newValue
6.          n.distance = newValue
7.          let n to be the parent of n
8.      end while
9.  end if
```

Figure 9. Algorithm of Decrease.

V. EXPERIMENT

The performance of the improved Thorup algorithm is evaluated through our modified experiment which originally provided by Pruehs [10]. The experiment compared the performance of Dijkstra and Thorup algorithm with synthetic datasets. The time cost of finding the distance of every vertex to a given source vertex is compared among the following three: Dijkstra with array-based primary queue, Dijkstra with Fibonacci-based primary queue, and Thorup. Since some algorithms mentioned in the Thorup algorithm [8] are difficult

to implement [9][10], Pruehs [10] replaced some algorithms with other algorithms. The detail information is as follows.

1. Use Kruskal's algorithm [12] to generate a minimum spanning tree so as to avoid using Fredman and Willard's union-find algorithm [13], since their algorithm uses atomic heaps as a priority queue, which requires $n > 2^{12^{20}}$. It is the same as the solution given by Thorup [8]. Moreover, in Kruskal's algorithm, Tarjan's union-find algorithm [14] is used to make set. The time cost of constructing a minimum spanning tree is bounded in $O(\alpha(m,n)m)$. Here α is an inverse function of Ackermann, which grows very slowly. If the number of vertices is less than 10^{80}, $\alpha(m,n)$ should be equal to, or less than 4[18].
2. Use Tarjan's union-find algorithm [14] instead of the tabulation-based algorithm [15] to construct components of the component tree.
3. Use Gabow's Split-findmin data structure [11] instead of atomic heaps to maintain the tentative distance of each vertex. It is called unvisited data structure in [8]. It is the same as the solution given by Thorup [8].

We modified the experiment in two phases,

1. Added the code of the improved Thorup algorithm.
2. Made the experiment possible to run with real datasets.

Because we avoid using unvisited structure in our algorithm, so the third change made by Pruehs[10] is not suitable for our algorithm. The dataset we used originally comes from the Geospatial Information Authority of Japan, which can be found from the link below: http://www1.gsi.go.jp/geowww/globalmap-gsi/download/data/gm-japan/gm-jpn-trans_u_2.zip.

For making it suitable for the experiment, the dataset is cut into five parts. All the datasets are pruned to delete single lines which both sides of this kind of line do not connect to other lines. Otherwise the minimum spanning tree cannot be constructed. An instance of a single line is shown in Fig.10. The algorithm of the prune single line is given in Fig. 11.

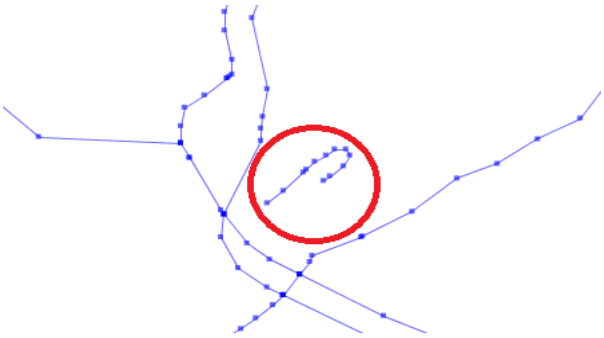

Figure 10. An instance of single line, which is need to be pruned.

```
    Delete_single_line (dataset)
1   for each line x in dataset
2    start = x.StartPoint
3    end = x.EndPoint
4    for each line y in dataset
5     if (start.intersection(y) && x! = y) then
6      start_intersect = TURE
7     end if
8     if (end.intersection(y) && x! = y)  then
9      end_intersect = TURE
10    end if
11    if (start_intersect == TURE &&
12     end_intersect == TURE)
13    end if
14    newdataset.add(x)
15    break
16    end for
17  end for
18  return newdataset
```

Figure 11. Algorithm of prune single line.

Detailed information of datasets is given by Table I. The pruned datasets can be found:

http://weiyusi.com/resource/gm-jpn-trans_u_2.7z

The source code of this project can be found:

http://weiyusi.com/resource/ShortestPaths.7z

TABLE I. INFORMATION OF DATASETS.

Info\Data	1	2	3	4	5
Vertices	1889	6913	11478	14295	16670
Edges	5920	21798	36262	44904	53318

The experiment proposed by Pruehs [10] originally can generate datasets, but does not support import of external dataset. We modified the project to make it able to transform Line Shapfiles to experimental datasets. Since the amount of the code is big, the algorithm is given in Appendix. It uses hash tables to allocate a unique vertex ID to each endpoint of each line in a Shapfile, and gets the length of a line as the weights of an edge. So that it turns endpoints to vertices, and turns lines to edges, which will be used in the experiment. In the situation that many lines have the same endpoints, the one has the shortest length will be reserved, and the others will be ignored.

TABLE II. RESULT OF EXPERIMENT.

Algorithms\Datasets	**1**	**2**	**3**	**4**	**5**
Dijkstra (MS) (Array heap)	0.5319	3.1180	5.0408	6.3472	7.3435
Dijkstra (MS) (Fibonacci heap)	3.4136	13.3113	16.8408	19.6897	23.2135
Thorup Construct Structures (MS)	4.1039	12.4451	20.4150	25.9321	32.5762
Thorup Visiting (MS)	3.4568	17.5888	28.1970	36.5393	42.0897
Improved Thorup Construct Structures (MS)	3.2122	10.9237	16.2118	21.8653	27.6312
Improved Thorup Visiting (MS)	0.8410	3.8663	6.8227	10.1153	10.2714

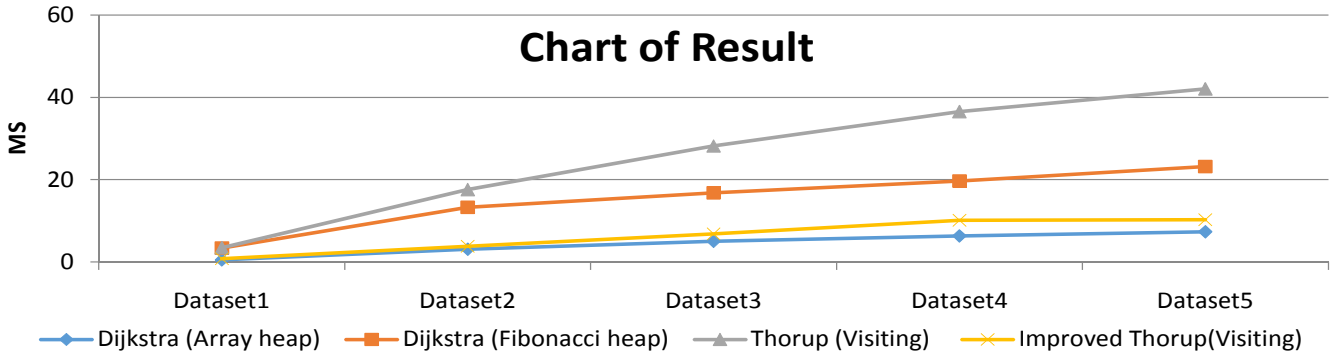

Figure 13. Chart of result.

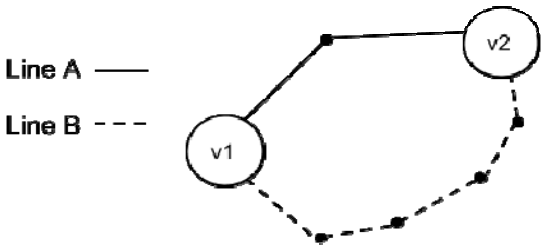

Figure 12. An example of allocating vertex IDs to the endpoints of lines.

In the algorithm introduced in the Appendix, from the line 5 to 7, we got the information of each line including weights and the coordinates of both of its endpoints. From the line 8 to line 21, we attempt to allocate a unique ID to each endpoint which was not allocated before. Each allocated endpoint will be stored in a hash table. Before allocating an ID, firstly, the program checks whether the endpoint is already in the hash table, which means it was allocated an ID before. This is used to make sure the spatial intersection point of two or more lines will not be allocated as two different IDs. When there are more than one line having the same endpoints, line 22 to 30 make sure that only the edge which has the shortest length can be kept in the dataset. As the example shown in Fig.12, there are two lines *A* and *B*, they are composed of many points, and they share the same endpoints, vertex *v1* and *v2*. Assume the path's length of *A* is shorter than *B*, and then *A* will be reserved in the dataset. Line 32 to the end of this algorithm creates an adjacent list with all transformed data which are used as an experimental dataset.

The result of comparison between Dijkstra, Thorup and our algorithm is given in Table II and Fig. 13. Because of Thorup algorithm can respond in arbitrary time of shortest path query from any vertex by constructing an index only one time, here we focus on the comparison of visiting part. Comparing to the Fibonacci-based Dijkstra and the original Thorup algorithm, our algorithm reduced 58.26% and 75.04% of time cost with all the five datasets, respectively. Comparing to the array-based Dijkstra algorithm, however, our algorithm takes 29.87% more time to finish query.

VI. CONCLUSION

We have proposed a practically-improved Thorup-based algorithm. It maintains the tentative distance of each vertex by a modified component tree of the Thorup algorithm so as to avoid using unvisited data structure. It costs 58.26% and 75.04% less time than Fibonacci-based Dijkstra and the original Thorup algorithm, respectively, but still slower than the array-based Dijkstra algorithm. In the future work, we shall try to find the structure that performs better than a component tree in maintaining the tentative distance of each vertex, and try to use the component tree to visit vertices in a more efficient way.

APPENDIX

The algorithm of transforming a Shapfile to experimental dataset is listed as follows.

```
1.  Hashtable hb, hbw
2.  List edges, graph
3.  Int id = 0
4.  for each line l of the Shapfile do
5.      sp = l.StartPoint
6.      ep = l.EndPoint
7.      weight = l.Length
8.      s = hb.get(sp)
9.      e = hb.get(ep)
10.     if s ==null then
11.         hb.put(sp, id)
12.         sEdge = id
13.         id++
14.     else sEdge = s
15.     end if
16.     if e ==null then
17.         hb.put(ep, id)
18.         eEdge = id
19.         id++
20.     else eEdge = e
21.     end if
22.     we = sEdge + "-" + eEdge
23.     w = hbw.get(we)
24.     if w == null then
25.         hbw.put(we, weight)
26.         edges.add(WeightedEdge(sEdge, eEdge))
27.     else if w > weight then
28.         hbw.remove(we)
29.         hbw.put(we, weight)
30.     end if
31. end for
32. for each edge e in edges do
33.     sEdge = e.Source
34.     eEdge = e.Target
35.     weight = hbw.get(sEdge+ "-" + eEdge)
36.     graph.add(WeightedEdge(sEdge, eEdge, weight))
37.     graph.add(WeightedEdge(eEdge, sEdge, weight))
38. end for
```

REFERENCES

[1] En, D., Wei, H., Yang, J., Wei, N., Chen, X., Liu, Y., "Analysis of the Shortest Path of GPS Vehicle Navigation System Based on Genetic Algorithm," Electrical, Information Engineering and Mechatronics 2011, Springer London, 2012, pp.413-418.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, "Keyword Searching and Browsing in Databases using BANKS," ICDE Conf, 2002, pp.431-440.

[3] R. Sivakumar, P. Sinha, V. Bharghavan, "CEDAR: a core-extraction distributed ad hoc routing algorithm," IEEE Journal on Selected Areas in Communications, 17, 1999, pp.1454-1465.

[4] R. Sedgewick, "Algorithms in Java Part 5, Graph 3rd Edition," Addison-Wesley Professional, 2003.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik 1, 1959, pp.269-271.

[6] D. Erik, R. Rivest, S. Devadas, "Introduction to Algorithms," Massachusetts Institute of Technology: MIT OpenCourseWare, 2008.

[7] J. Y. Yen, "A Shortest Path Algorithm," Ph.D. dissertation, University of California, Berkeley, 1970.

[8] M. Thorup, "Undirected single source shortest paths in linear time," Proceedings of the 38th Symposium on Foundations of Computer Science, 1997, pp.12-21.

[9] Y. Asano, H. Imai, "Practical Efficiency of the Linear Time Algorithm for the Single Source Shortest Path problem," Journal of the Operations Research, Society of Japan, Vol. 43, No. 4, 2000, pp.431-447.

[10] N. Pruehs, "Implementation of Thorup's Linear Time Algorithm for Undirected Single-Source Shortest Paths with Positive Integer Weights," Bachelor thesis, University of Kiel, 2009.

[11] H. N. Gabow, "A scaling algorithm for weighted matching on general graphs," In Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science.IEEE Computer Society Press, Los Alamitos, Calif., 1985, pp.90-100.

[12] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proc. Am. Math. Soc. 7, 1956, pp.48-50.

[13] M. L. Fredman, E. D. Andwillard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," J. Comput. Syst. Sci. 48, 1994, pp.533-551.

[14] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," J. ACM 22 , 2 (Apr.), 1975, pp.215-225.

[15] H. N. Gabow, E. R. Tarjan, "A linear-time algorithm for a special case of disjoint set union," J. Comput. Syst. Sci. 30, 1985, pp.209-221.

[16] T. Hagerup, "Improved shortest paths on the word RAM," Automata, Languages and Programming, 2000, pp.61-72.

[17] S. Pettie, "A new approach to all-pairs shortest paths on real-weighted graphs," Theoretical Computer Science, 312(1), 2004, pp.47-74.

[18] H. Thomas, Cormen, E. Charles, Leiserson, L. Ronald, Rivest, Stein, Clifford, "Introduction to Algorithms (3rd ed.)," MIT Press and McGraw-Hill, 2009.