Anarchy in the APSP:
New Efficient and Incorrect APSP Algorithm

Jason (Jaehyun) Koo @ Theory Lunch ‘24
MIT EECS

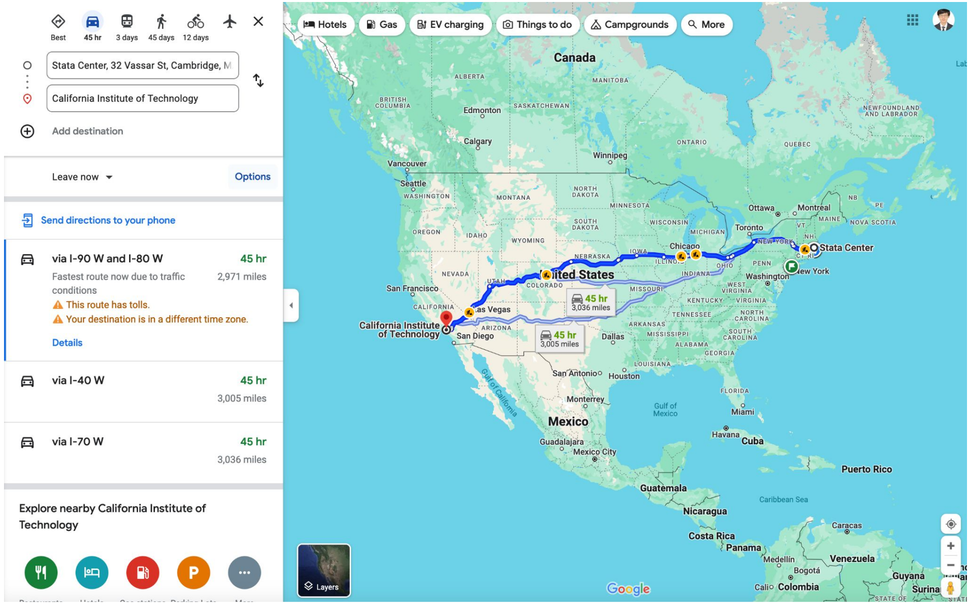
Definition: (Single-Source) Shortest Path

Input:

- A directed graph $G = (V, E)$
- Positive integer edge weights $w : E \to [n^c]$
- A designated source $s \in V$

Output:

- A size-n table of shortest distance to all vertices in V



(n: # of vertices, m: # of edges)

Definition: All-Pair Shortest Path

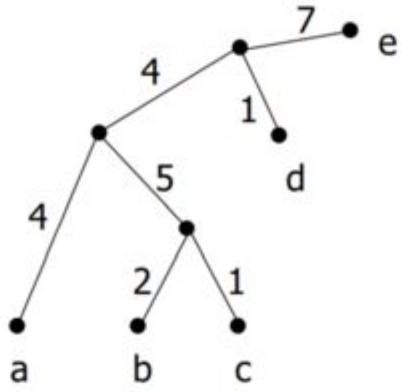
Input:

- A directed graph G = (V, E)
- Positive integer edge weights w : E -> [n^c]
- ~~A designated source s \in V~~

Output:

- An n x n table of shortest distance from each vertices, to each vertices.

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0



Two Algorithms for Shortest Paths

Dijkstra’s Algorithm

- Computes the single-source shortest path in $O(n \log n + m)$ time
- Computes the all-pair shortest path in $O(n^2 \log n + mn)$ time

Floyd-Warshall Algorithm

- Computes the all-pair shortest path in $O(n^3)$ time

Dijkstra’s algorithm is only better in sparse cases ($m \ll n^2$)

Floyd-Warshall Algorithm (The KIJ Algorithm)

D[i, j] =

- 0, if i = j
- w(i -> j) if there exists an edge i -> j in E
- infinity otherwise.

```
for k in range(0, n):
    for i in range(0, n):
        for j in range(0, n):
            D[i, j] = min(D[i, j], D[i, k] + D[k, j])
```

Freshman's dream? (The IJK Algorithm)

```
for i in range(0, n):
    for j in range(0, n):
        for k in range(0, n):
            D[i, j] = min(D[i, j], D[i, k] + D[k, j])
```

Freshman's dream? (The IJK Algorithm)

Why doesn't the Floyd-Warshall algorithm work if I put k in the innermost loop

Asked 11 years ago Modified 2 years, 11 months ago Viewed 4k times

10

The Floyd-Warshall algorithm is defined as follows:

```
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][k] + dist[k][j] < dist[i][j] then
            dist[i][j] ← dist[i][k] + dist[k][j]
```

Why doesn't it work if I simply use

```
for i from 1 to |V|
  for j from 1 to |V|
     for k from 1 to |V|
        if dist[i][k] + dist[k][j] < dist[i][j] then
           dist[i][j] ← dist[i][k] + dist[k][j]
```

In this case, the intermediate node k is iterated in the innermost loop. I expect it will make the same comparisons, but maybe different order. Why is the result different and incorrect?

algorithms graphs shortest-path

Featured on Meta

- Data Dumps Releases: Timeline Updates and Clarification
- What would you like to change a the moderator election process?
- Updates to the Acceptable Use I (AUP) – January 2024

Linked

- 3 Can the loops be in any order i Floyd-Warshall algorithm?
- 1 Can I change the order of iterat the Floyd-Warshall Algorithm?

Related

- 3 Shortest path with odd weight

What is wrong with my Floyd Warshall algorithm?

Asked 1 year, 2 months ago Modified 1 year, 2 months ago Viewed 82 times

0

Here is the `problem link`

Approach:

My approach is to simply give every other node the chance to be an in-between node for every 2 pairs of vertices `i` and `j`.

Below is my code for Floyd Warshall algorithm:

```
class Solution{
    public void shortest_distance(int[][] mat){
        int N = mat.length;

        for(int i = 0; i < N; ++i){
            for(int j = 0; j < N; ++ j){
                for(int k = 0; k < N; ++k){
                    if(mat[i][k] != -1 && mat[k][j] != -1 && (mat[i][j] == -1 || ma
                        mat[i][j] = mat[i][k] + mat[k][j];
```

Quora

Why is the order of the loops in Floyd-Warshall algorithm important to its correctness ?

All related (33) Sort Recommended

Assistant · Bot

The order of the loops in the Floyd-Warshall algorithm is important for its correctness because it ensures that the shortest ... two vertices is correctly

Continue reading

Sumedh Gupte
Researcher · 7y

The order of loops in Floyd-Warshall algorithm is important because it determines the number of times you are finding distance between two given vertices.

In the correct implementation, you need to find distance between two vertices 'n' times, since you have to consider shortest distance among all possible hops (and permutations of hops). So for each k(outermost iteration), you find the distance between a pair(i,j) considering 0 to k no of hops in between.

8.3 Some students mistakenly transpose, in the Floyd-Warshall algorithm, the lines
15 for k ← 1 to n do
16 for i ← 1 to n do
17 for j ← 1 to n do
: : :
to read,
15 for i ← 1 to n do
16 for j ← 1 to n do
17 for k ← 1 to n do
: : :

Let the transposed algorithm be run on a general graph. What paths will it consider as po vertex 1 to vertex 2? Describe this selection in words.

The ⭐🌈 *incorrect* ⭐🌈 APSP Problem

Definition. In the *Incorrect all-pair shortest path problem* (wrong-APSP):

- The input is a digraph with positive integer weights in [1, n^c],
- The output is an n x n matrix corresponding to the resulting output of the IJK Algorithm (incorrect Floyd-Warshall algo: ➡️)

```
for i in range(0, n):
    for j in range(0, n):
        for k in range(0, n):
D[i, j] = min(D[i, j], D[i, k] + D[k, j])
```

Motivation


when you hit a wrong note in classical
@CharlesBerthoud
구독중
Playing wrong notes in CLASSICAL vs JAZZ

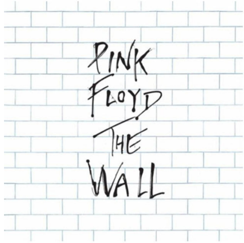
Motivation

Late 19C: The invention of Jazz

1970s: The invention of Punk rock

THE BEATLES

2020s: The invention of Punk TCS??



Algorithm 2 IJK algorithm

for $i = 1, 2, \ldots, n$ **do**
  for $j = 1, 2, \ldots, n$ **do**
    for $k = 1, 2, \ldots, n$ **do**
      $\mathtt{d}[i, j] \leftarrow \min\{\mathtt{d}[i, j], \mathtt{d}[i, k] + \mathtt{d}[k, j]\}$
    end for
  end for
end for

(less motivating) Motivation

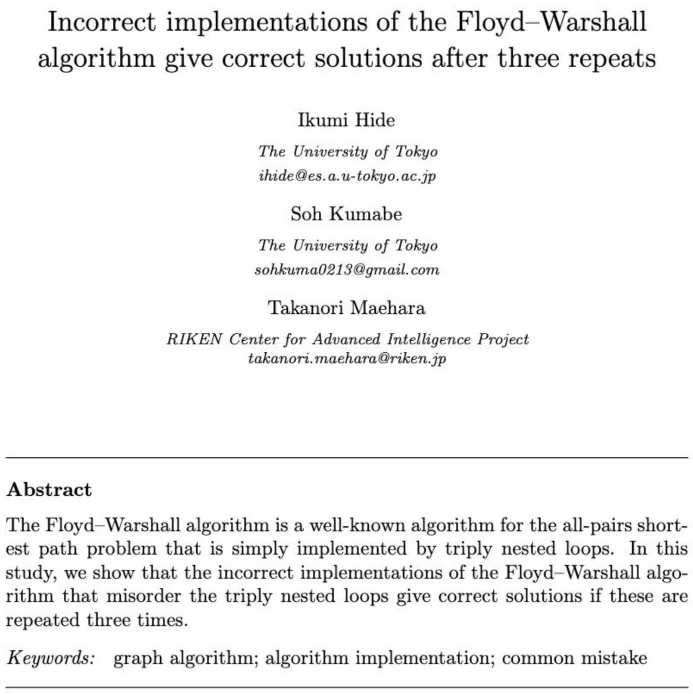
APSP Conjecture (WW10): The all-pairs shortest paths problem can not be solved in $O(n^{3 - c})$ time for any constant $c > 0$.

Theorem (HKM19): Iterating the IJK Algorithm **three times** actually gives the correct APSP distance matrix (like in ➡️)

Truly subcubic incorrect APSP - a breakthrough!

```
for _ in range(0, 3):
    for i in range(0, n):
        for j in range(0, n):
            for k in range(0, n):
D[i, j] = min(D[i, j], D[i, k] + D[k, j])
```

(less motivating) Motivation



Incorrect implementations of the Floyd–Warshall algorithm give correct solutions after three repeats

Ikumi Hide
The University of Tokyo
ihide@es.a.u-tokyo.ac.jp

Soh Kumabe
The University of Tokyo
sohkuma0213@gmail.com

Takanori Maehara
RIKEN Center for Advanced Intelligence Project
takanori.maehara@riken.jp

Abstract

The Floyd–Warshall algorithm is a well-known algorithm for the all-pairs shortest path problem that is simply implemented by triply nested loops. In this study, we show that the incorrect implementations of the Floyd–Warshall algorithm that misorder the triply nested loops give correct solutions if these are repeated three times.

Keywords: graph algorithm; algorithm implementation; common mistake

(actually this is just an arxiv preprint, but what they proved is a subset of our result, so unless we are wrong they are correct)

Our results: Thinking fast and wrong

In sparse graph APSP problem can be solved in quadratic time using Dijkstra's algorithm. The same upper bound was not known in wrong-APSP problem, until we closed the gap:

Our results. There is an $O(nm+n^2 \log n)$ algorithm for wrong-APSP problem.

This is n^3 in dense graph, so our result do not contradict APSP conj.

A comparison

	APSP problem	wrong-APSP problem
Hardness	APSP-Complete	APSP-Hard [HKM19]
Best algo for dense	n^3 [FW62]	n^3 (trivial alg. from defn)
Best algo for sparse	nm + n^2 log n [FT84]	n^3 (previous) nm + n^2 log n (this work)
Cool?		

Our strategy

We represent the algorithm A as a sequence of the 3-tuples.

Each tuples corresponds to a “relaxation”:

relax(i, j, k): D[i, j] = min(D[i, j], D[i, k] + D[k, j])

We say a path P = {p_0, p_1, …, p_l} is **realized** by A, iff:

- l = 1 and (p_0 -> p_1) ∈ E
- l > 1, there exists an entry A_i = (p_0, p_l, p_x) such that
    - {p_0, p_1, …, p_x} is realized by A_1, A_2, …, A_{i-1}
    - {p_x, p_{x+1}, …, p_l} is realized by A_1, A_2, …, A_{i-1}

Our strategy

We say a path $P = \{p_0, p_1, \ldots, p_l\}$ is **realized** by A, iff:

- $l = 1$ and $(p_0 \rightarrow p_1) \in E$
- $l > 1$, there exists an entry $A_i = (p_0, p_l, p_x)$ such that
  - $\{p_0, p_1, \ldots, p_x\}$ is realized by $A_1, A_2, \ldots, A_{i-1}$
  - $\{p_x, p_{x+1}, \ldots, p_l\}$ is realized by $A_1, A_2, \ldots, A_{i-1}$

- **What does it mean?:** These are exactly the path that is considered by algorithm A.
- **Goal**: Provide a succinct description of paths realized by IJK Algorithm, and compute it efficiently with e.g. Dijkstra, using such description.

Recitation: Floyd-Warshall

We first recall the following classical and beautiful proof, which shows a correctness of Floyd-Warshall algorithm.

Proposition. KIJ (Floyd-Warshall) Algorithm realizes all simple paths of G.

Proof.

- For a path with at most one edges, trivial.
- For a path with at least two edges, we induct on the maximum index of the middle vertices.

Recitation: Floyd-Warshall

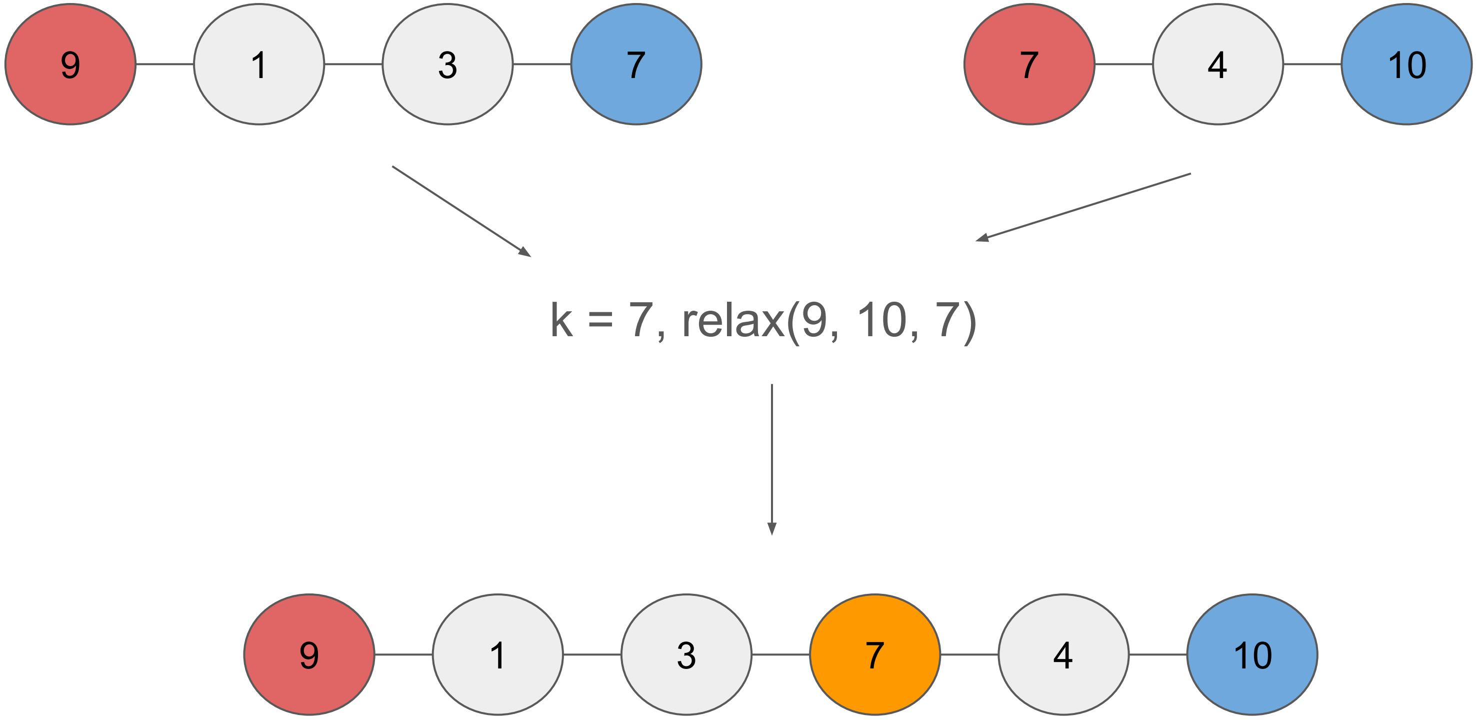
Let $P = \{p_0, p_1, \ldots, p_l\}$ for $l >= 2$:

the “middle vertices” are $p_1, p_2 \ldots, p_{l-1}$.

Induction hypothesis: After iteration $k = t$, simple paths where all middle vertices have index $<= t$ are realized.

- If all middle vertices have index $<= t - 1$, follows from I.H
- Otherwise, let $p_x = t$ be the unique middle vertex. $p_0 \ldots p_x$ are realized by I.H, and $p_x \ldots p_l$ are realized by I.H. We can certainly find an entry with $(p_0, p_l, p_x = k)$, since in iteration $k = x$ we try all (i, j).

Recitation: Floyd-Warshall

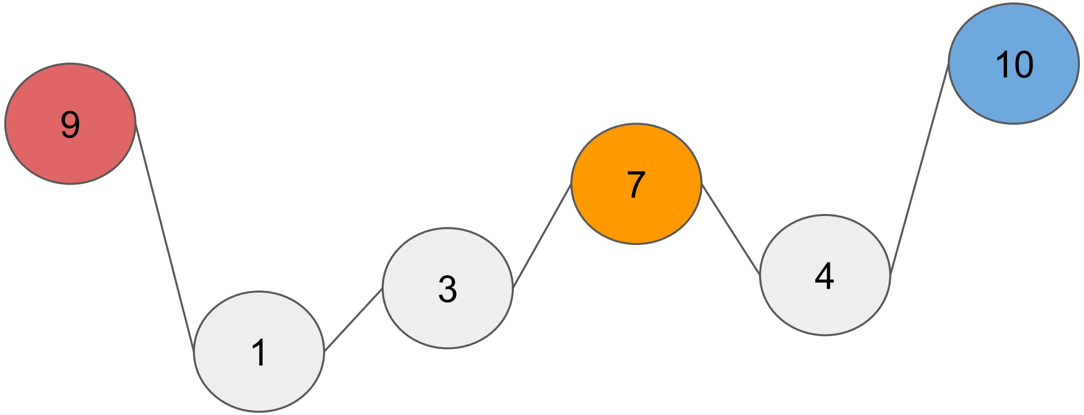


Equivalent condition

For the IJK algorithm, there are *similar* arguments that can be made.

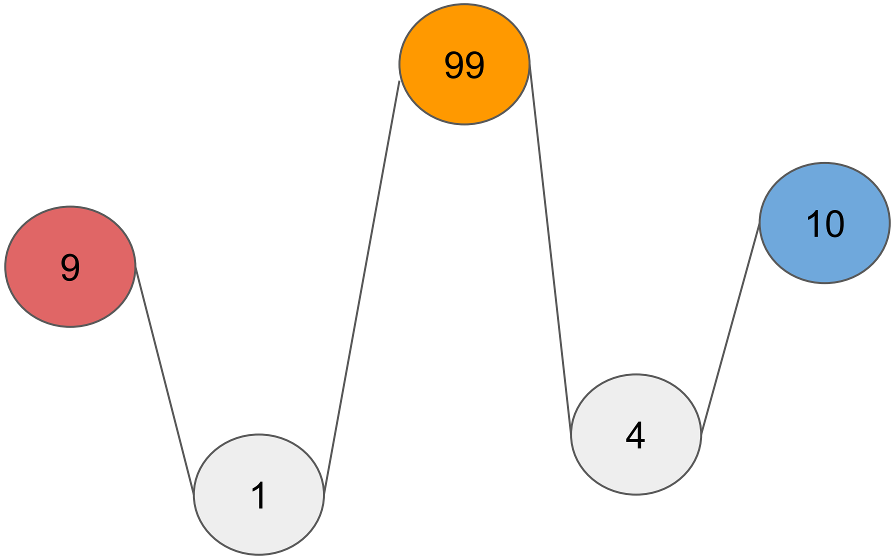
Proposition. IJK (Floyd-Warshall) Algorithm realizes all paths of G, such that all middle vertices have smaller indices than the endpoint of the path.

(Proof follows the same outline from the KIJ algorithm.)



Equivalent condition

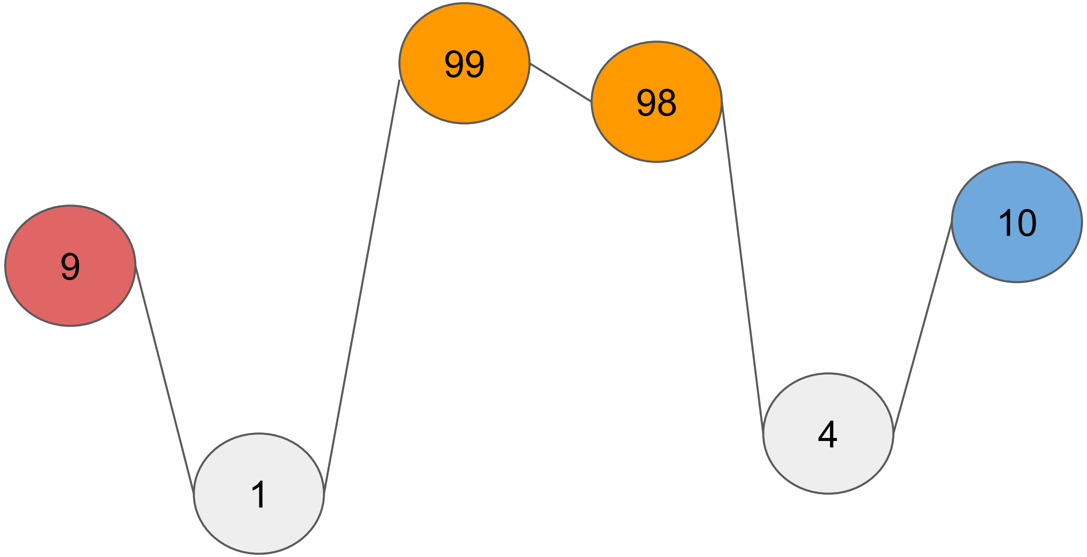
But that's not the whole characterization. For example, the IJK algorithm realizes this path: Consider the sequence (1, 4, 99), (9, 4, 1), (9, 10, 4).



Equivalent condition

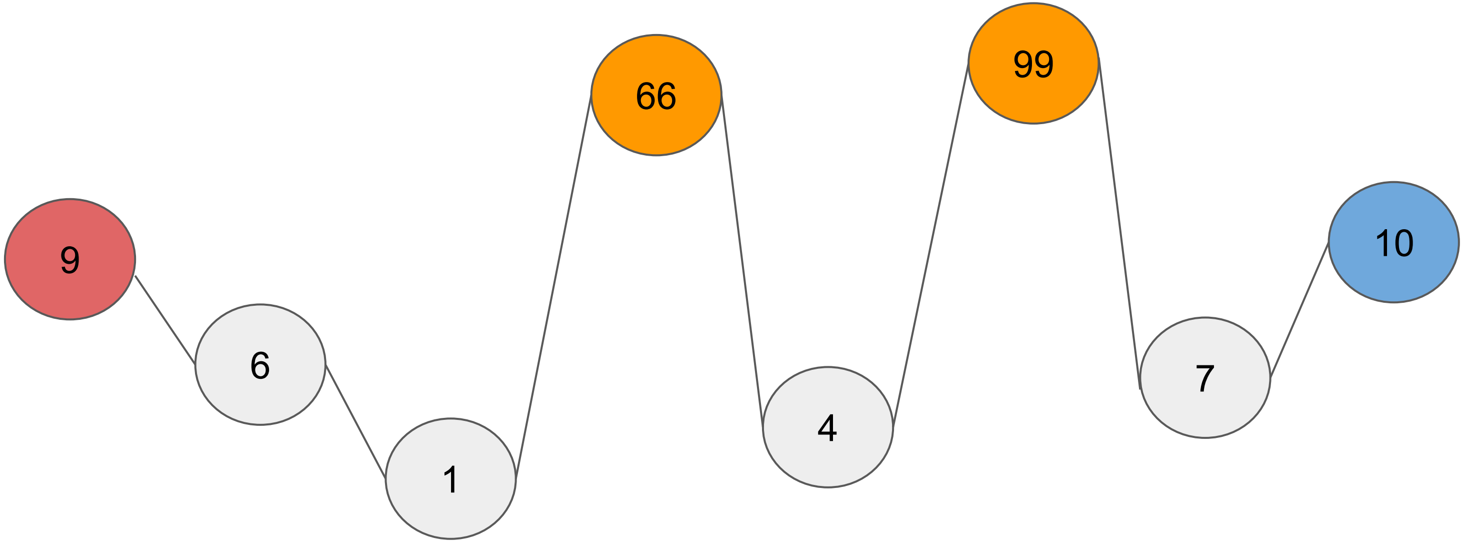
But there is a limited degree of freedom.

This path is not realized by IJK algorithm.



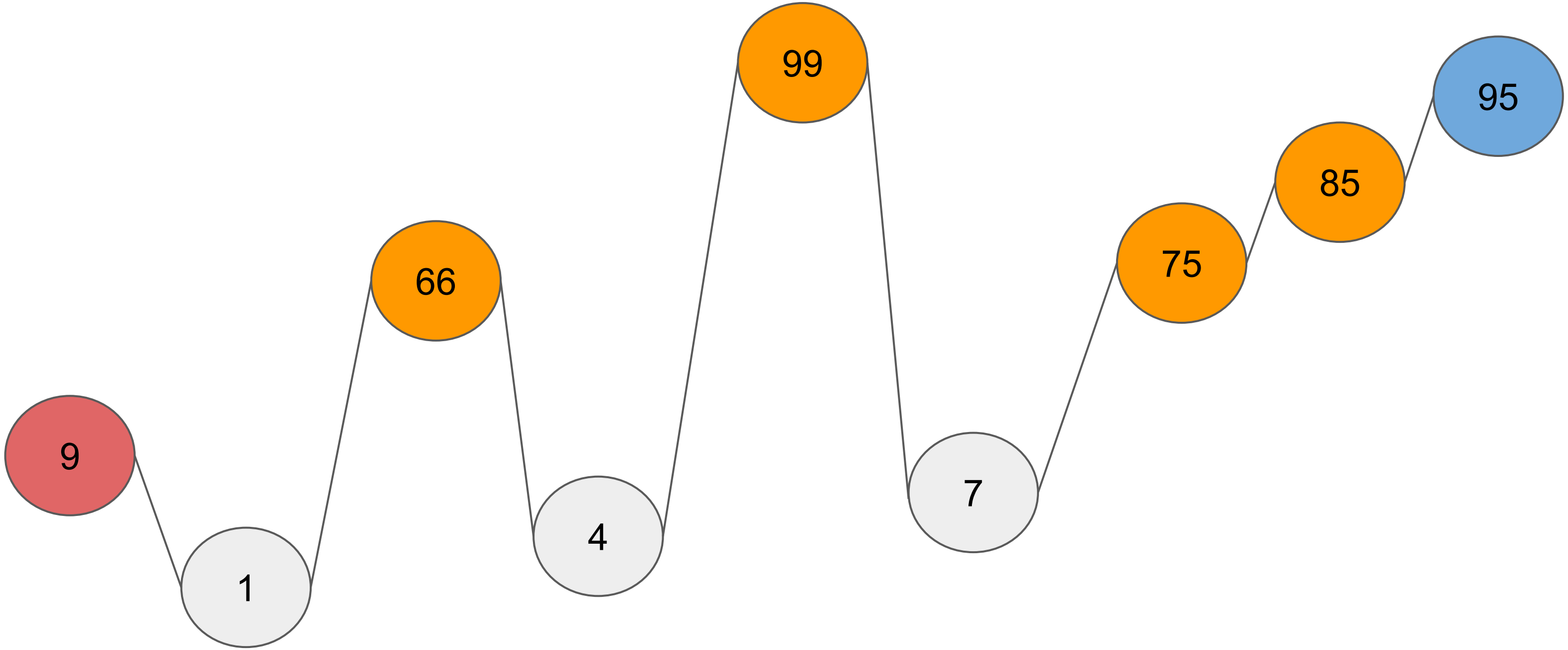
Equivalent condition

By trying out some examples, we can see that each “large” vertices (k) need sufficiently small neighbors (i, j) so that it can be soldered into the path in the lexicographically smaller part of relaxation.



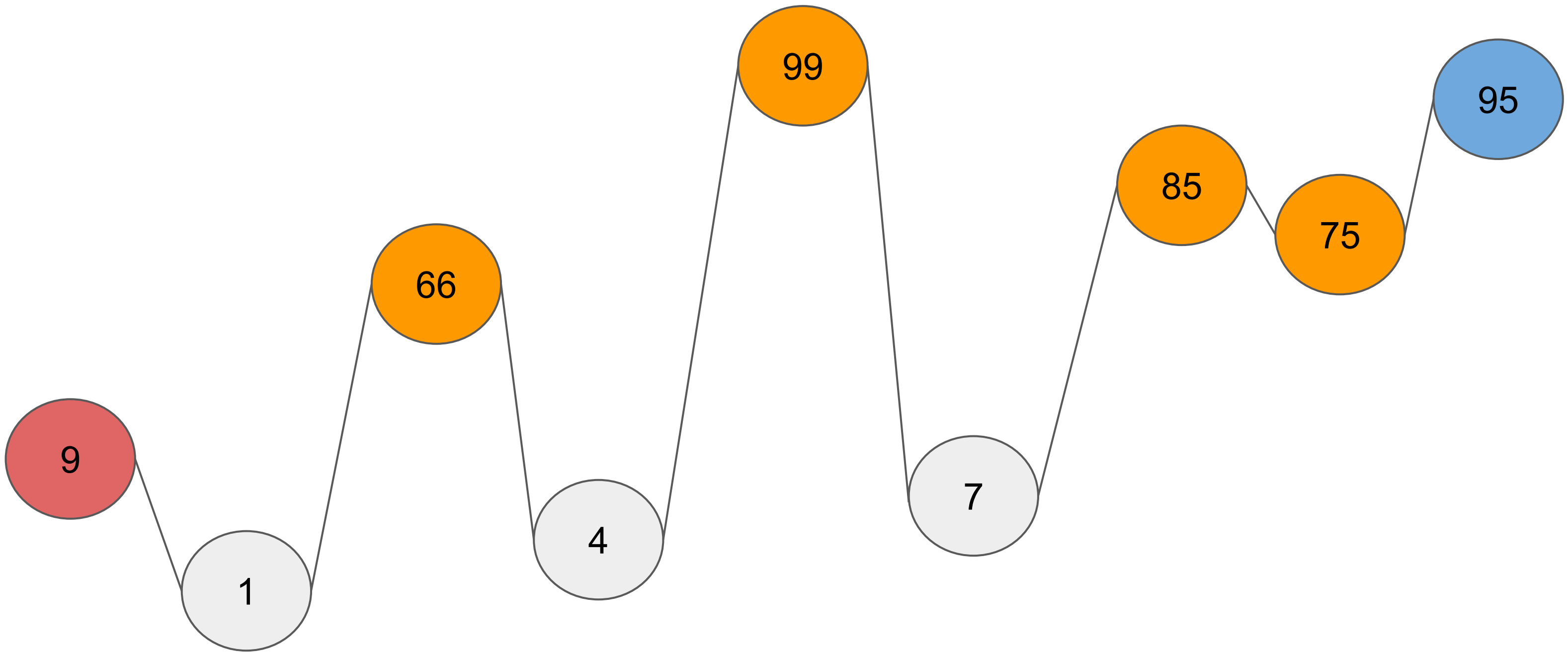
Equivalent condition

So does it mean it’s ok if they are not adjacent? But these are realizable.



Equivalent condition

These are not, on the other hand:



Equivalent condition

It’s actually quite fun to figure out what is doable and whatnot

But I will spoil your fun for sake of completeness..

Equivalent condition

Definition. An i - j path is **proper** if no middle vertex of index greater than min(i, j) is adjacent.

Definition. For all i <= j, an i - j path is **increasing** if the index of vertices are strictly increasing. We define **decreasing** path similarly for i >= j.

Theorem: A nonempty path is realized by IJK algorithm if and only if:

- i < j, and it is a concatenation of proper path and an increasing path.
- i > j, and it is a concatenation of decreasing path and a proper path.

(Proof followed by easy yet pretty satisfying induction)

Proper path

We will find, for a fixed i and all j > i, a length of proper i - j path.

Given j > i, A proper i - j path does not contain two adjacent middle vertex with index greater than min(i, j) = i.

Proper path

We will find, for a fixed i and all j > i, a length of proper i - j path.

Given j > i, A proper i - j path does not contain two adjacent middle vertex with index greater than min(i, j) = i.

In other words, if you are at vertex of index at least i, you cannot move to another vertex of index at least i

UNLESS you are ending a path (because it is only about middle vertex!)

Proper path

> In other words, if you are at vertex of index at least i, you cannot move to another vertex of index at least i

- Apply single-source shortest path algorithm, in a graph where edges connecting two vertices of index at least i are all removed.

> UNLESS you are ending a path (because it is only about middle vertex!)

- A single last move from a path can be done freely - apply linear pass

Finding a proper path can be done in $O(m + n \log n)$ time with Dijkstra's alg.

Append an increasing path

For each vertex j > i we know the shortest proper path from i ending at j. We need to append a (possibly empty) increasing path from there, to obtain an even shorter solution.

This can be done with linear-time dynamic programming:

- D[j] = (shortest path that is proper+increasing and ends at j)
  - Case 1: Take a proper path
  - Case 2: Append an edge k -> j at the back of path such that k < j. Here the cost is D[k] + w(k->j).

Future Works & Acknowledgement

- We know that wrong-APSP is APSP-Hard, but is it APSP-complete?
  - Actually an interesting question!
- Very fast solution in special graphs?
  - Graphs with bounded treewidth / interval graphs / permutation graphs?
- Can we publish this?
  - Likely harder than the problem itself!

Acknowledgement. I would like to thank several programming contest people for motivating this study, and an anonymous MIT alumni for helpful discussion on this problem in an otherwise lazy saturday afternoon of New York City.

References

- On incorrect floyd-warshalls
  - https://cs.nyu.edu/~siegel/JJ10.pdf
  - https://stackoverflow.com/questions/74507878/what-is-wrong-with-my-floyd-warshall-algorithm
  - https://www.quora.com/Why-is-the-order-of-the-loops-in-Floyd-Warshall-algorithm-important-to-its-correctness
  - https://cs.stackexchange.com/questions/9636/why-doesnt-the-floyd-warshall-algorithm-work-if-i-put-k-in-the-innermost-loop
- https://arxiv.org/abs/1904.01210
- https://www.acmicpc.net/problem/20588