Overview of Single Source Shortest Path

Tuesday, February 8, 2022 8:02 PM

- For unweighted graphs we can BFS as the shortest path is basically no of edges between vertices.
- But for weighted graphs it isn't the case.

- Previously, we used the "breadth-first search" algorithm to find the "shortest path" between two vertices. However, the "breadth-first search" algorithm can only solve the "shortest path" problem in "unweighted graphs". But in real life, we often need to find the "shortest path" in a "weighted graph".
- For example, there may be many routes from your home to a target location, such as a bus station, and the time needed for each route may be different. The route with the shortest distance may not be the one that requires the least amount of time because of the speed limit and traffic jams. So, if we want to find the route that takes the least time from home to a certain bus station, then the weights should be time instead of distance. With that in mind, how can we solve the "shortest path" problem given two vertices in a "weighted graph"?
- The main focus of this chapter is to solve such "single source shortest path" problems. Given the starting vertex, find the "shortest path" to any of the vertices in a weighted graph. Once we solve this, we can easily acquire the shortest paths between the starting vertex and a given target vertex.

Edge Relaxation

- An alternative way to understand why this process is called 'relaxation' is to imagine that each path is a rubber band of length 1. The original path

from A to D is of length 3, so the rubber band was stretched to 3 times its original length. When we relax the path to length 2, by visiting C first, the rubber band is now only stretched to twice its length, so you can imagine the rubber band being relaxed, hence the term edge relaxation.

- In this chapter, we will learn two "single source shortest path" algorithms:
  - Dijkstra's algorithm
  - Bellman-Ford algorithm
- "Dijkstra's algorithm" can only be used to solve the "single source shortest path" problem in a weighted directed graph with non-negative weights.
- "Bellman-Ford algorithm", on the other hand, can solve the "single-source shortest path" in a weighted directed graph with any weights, including, of course, negative weights.

Limitation

Dijkstra's Algorithm

Tuesday, February 8, 2022 8:12 PM

- "Dijkstra's algorithm" solves the "single-source shortest path" problem in a weighted directed graph with non-negative weights.

The Main Idea

- We take the starting point u as the center and gradually expand outward while updating the "shortest path" to reach other vertices.
- "Dijkstra's Algorithm" uses a "greedy approach".
- Each step selects the "minimum weight" from the currently reached vertices to find the "shortest path" to other vertices.

Imp We always take the min wt. from the currently reached vertices.
we can use "set" for that (See Code)

vector<int> d; → dist from source
vector<int> p; → to store path
{ a → b p[b] = a. }

Limitation of the Algorithm

"Dijkstra's Algorithm" can only be used on graphs that satisfy the following condition:

- Weights of all edges are non-negative.

Complexity Analysis

V represents the number of vertices, and **E** represents the number of edges.

- Time Complexity: **O(E+VlogV)** when a Fibonacci heap is used, or **O(V + E log V)** for a Binary heap.
  - If you use a Fibonacci heap to implement the "min-heap", extracting minimum element will take **O(log V)** time while key decreasing operation will take

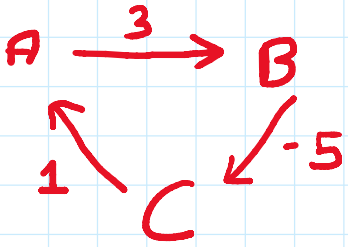
amortized **O(1)** time, therefore, the total time complexity would be **O(E + V log V)**.
    - If you use a Binary heap, then the time complexity would be **O(V + E log V)**.
- Space Complexity: **O(V)**. We need to store **V** vertices in our data structure.

Bellman Ford Algorithm

Tuesday, February 8, 2022 8:35 PM

If we got problem like use at most (k) edges use DP, bellman ford wont give the correct answer.

- As discussed previously, the "Dijkstra algorithm" is restricted to solving the "single source shortest path" problem in graphs without negative weights. So, how could we solve the "single source shortest path" problem in graphs with negative weights? In this chapter, we will introduce the Bellman-Ford algorithm.
- Works only for graphs with Positive weight cycles because with negative weight cycles we can't reach on to a result.



1st iter
A → 0
B → 3 A-B
C → -2 B-C
A → -1 C-A
Earlier it was 0
if we keep on going in cycles we will always get ans decreasing

∴ we need +ve weighted cycle

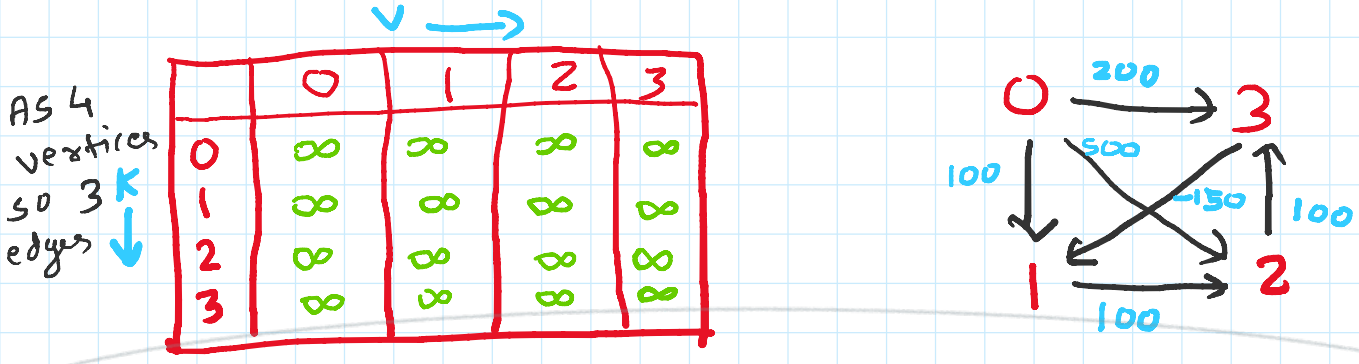
-ve wt. cycle
✗ not gonna work

- We do N-1 iterations of bellman ford because we need N-1 edges.
- In a "graph with no negative-weight cycles" with N vertices, the shortest path between any two vertices has at most N-1 edges.

Using Dynamic Programming to Find the Shortest Path

[At most k edges]

- It will dp[v][k]
  Where v - is the vertex
  K - at most edges to reach v
  Dp[v][k] - the min distance from 0(start) vertex to v using at most k edges.
- It will be something like:



$$dp[v][k] = \min\left[\, dp[v][k],\ dp[u][k-1] + W(u,v) \,\right]$$

- Complexity Analysis

DP

V represents the number of vertices in the graph, and E represents the number of edges.

  - Time Complexity: O(V·E). In the worst-case scenario, when all the vertices are connected with each other, we need to check every path from every vertex; this results in O(V·E) time complexity.
  - Space Complexity: O(V^2). We need to store a 2-dimensional DP matrix with the size of VxV.

- Limitation of the algorithm

"Bellman-Ford algorithm" is only applicable to "graphs" with no "negative weight cycles".

- How does the Bellman-Ford algorithm detect "negative weight cycles"?

Do n^{th} iter if d[u] (u∈V) decreases, NWC exists.

Although the "Bellman-Ford algorithm" cannot find the shortest path in a graph with "negative weight cycles", it can detect whether there exists a "negative weight cycle" in the "graph".

Detection method: After relaxing each edge N-1 times, perform the Nth relaxation. According to the "Bellman-Ford algorithm", all distances must be the shortest after relaxing each edge N-1 times. However, after the Nth relaxation, if there exists distances[u] + weight(u, v) < distances(v) for any edge(u, v), it means there is a shorter path . At this point, we can conclude that there exists a "negative weight cycle".

- Complexity Analysis

V represents the number of vertices, and E represents the number of edges.

- Time Complexity: we iterate through all the vertices, and in each iteration, we'll perform a relaxation operation for each appropriate edge. Therefore, the time complexity would be $O(V \cdot E)$

- Space Complexity: $O(V)$. We use two arrays of length V. One to store the shortest distance from the source vertex using at most $k-1$ edges. The other is to store the shortest distance from the source vertex using at most k edges.