CS301: Assignment #4

Muhammed Orhun Gale
morhun@sabanciuniv.edu

Sabanci University — May 22, 2022

1 (a) Tourist Problem

1.1 Recursive Definition

Tourist problem can be simplified to and considered as "Single Source Shortest Path" problem. Since travelling around different cities cannot be possible without spending time, edges of the graph which the tourist will wander do not have negative values and thus in this problem encountering with negative cycles is not possible. Also, note that the problem is simplified to find the length of the shortest paths however, one must construct the path after the length of shortest paths are found. In this sense, this problem can be formulated recursively as follows:

Define a graph G which does not have any negative edges as $G = (V, E, w)$ where V is the set of vertices, E is the set of edges and w is the function that maps an edge to a weight

Then define $\delta(s, v)$ as the shortest path from vertex s to vertex v where $s, v \epsilon V$. Then the problem can be formulated recursively as:
$\delta(s, s) = 0$ when v is same as the source vertex s
$\delta(s, v) = min(\delta(u, v) + w(s, u))$ for vertices different than the source vertex s

1.2 Optimal Substructure Property

In order to justify investing in a DP strategy to solve Tourist Problem, one must observe the Optimal Substructure Property.

According to the recursive formulation of the problem, solutions of problems are constructed via using solutions of sub-problems. This property can be proven with Cut and Paste method as follows:

Suppose that $\delta(s, v)$ is a shortest path from vertices s to v. Then according to optimal substructure property $\delta(s, u)$ should be in the shortest path $\delta(s, v)$. Now suppose otherwise, $\delta(s, v-1)$ is not a member of the shortest path $\delta(s, v)$. This means that there is a shorter path from s to u. If there exist a shorter path from s to u this means that the shortest path must be less than the $\delta(s, v)$ which contradicts with our supposition. Therefore, optimal substructure property is proved by contradiction.

1.3 Topological Ordering

After visit a vertex going back to it does not worth it since there is no negative edges, therefore no negative cycles. Topological ordering of the problems can be described as $s, u1, u2, ..., un, u1.1, ..., u1.n, u2.n, ..., u2.m, ..., v$ where un are the subproblems of the main problem s and $un.m$ are the subproblems of the un, where where $n, m = 1, ..., k$. Therefore, the problem would terminate since any subproblem does not depend on a prior subproblem.

1.4 Overlapping Subproblems

In the Tourist Problem, there exist overlapping sub-problems.

Suppose that there exist two vertices $v1$ and $v2$ whose out-degrees are high and there exist an edge from $v1$ to $v2$. Then, suppose that one tries to find shortest path from vertices s to u and vertices $v1$ and $v2$ are included in the shortest path which would be found. Then, while computing the shortest path $\delta(s, v1)$, one again would need to compute $\delta(s, v2)$ since $\delta(s, v1)$ would call the problem $\delta(s, v2)$. Therefore, it can be said that there exist many overlapping sub-problems in while computing this problem which shows that investing in a DP strategy to solve this problem would worth.

2 (b) Pseudocode of the algorithm to solve Tourist Problem

In order to solve the Tourist Problem, I designed a Bellman-Ford variant algorithm which is applicable to a "Single Source Shortest Path" algorithm so to Tourist Problem.

In order to apply my algorithm, I used adjacency list representation. I represented each train station and bus station as a vertex and routes between these stations are represented as edges. Also note that bus stations and train stations on the same city are represented separately which means that those stations also have edges between them. Since my algorithm is a Bellman-Ford variant, it detects shortest path from the source vertex to all other vertices. After calculating the shortest paths for each vertex, for the nodes in the same cities my algorithm detects the shortest one between bus and train options and then represented the shortest one. After that, via using the parental information in each vertex, algorithm constructs the path.

procedure $findShortestPath(source, Vertices, Edges)$:

distanceEstimates $= \inf$ for size of |Vertices|
piList $= -1$ for |Vertices|
distanceEstimates[source] $= 0$
loop|Vertices| - 1 times:
  for *e in Edges(u, v) Edges* **do**
    if distanceEstimates[v] > distanceEstimates[u] + GetWeight(Edges[u])
    distanceEstimates[v] = distanceEstimates[u] + GetWeight(Edges[u])
    piList[v] = u
  end

return distanceEstimates, piList

3 (c) Asymptotic time and space complexity analysis of the algorithm

$\mathbf{Space - Complexity}$:
$\mathbf{Worst - case}$: My algorithm needs to store the information of all vertices and edges. Since each station is considered separately from the cities, there can be at most $(2N) = (N)$ vertices if there exist N cities. For edges, in the worst case the graph can be dense which means that there can be $|V| * (|V| - 1)/2 = \Theta(|V|^2)$ edges. Therefore, the space complexity would be $(|V| + |E|) = (|V| + |V|^2) = (|V|^2)$ in worst case.

$\mathbf{Best - case}$: In the best case, there can be at most again (V) vertices but this time the graph can be sparse which means that there exist $\Omega(V)$ edges. Therefore in the best case the space complexity would be $\Omega(V + E) = \Omega(V + V) = \Omega(V)$

$\mathbf{Time - Complexity}$:
In the worst case outer loop iterates |V|-1 times ($\Theta(|V|)$) and inner loop would iterate all the edges ($\Theta(|E|)$) therefore the time complexity would be ($\Theta(|V| * |E|)$) in total.

$\mathbf{Worst - case}$: In the worst case the graph would be dense and therefore there can be $V*(V-1)/2 = (|V|^2)$ edges. In this sense the the time complexity would be $(|V| * |E|) = (|V| * |V|^2) = (|V|^3)$ at worst

Best − case : At best, the graph would be sparse therefore there exist $\Omega(V)$ edges. In this sense $\Omega((|V| * |E|) = \Omega((|V| * |V|) = \Omega((|V|^2)$

Note that $findShortestPath$ procedure finds the lengths of the shortest paths however, this is the simplified version of the original problem. Therefore, one must needs to construct path via using parent nodes. In order to construct the shortest path at most $\Theta(|V|)$ vertices will be visited for $\Theta(|V|)$ vertices therefore it would take $\Theta(|V|^2)$ time and would not effect the overall time complexity

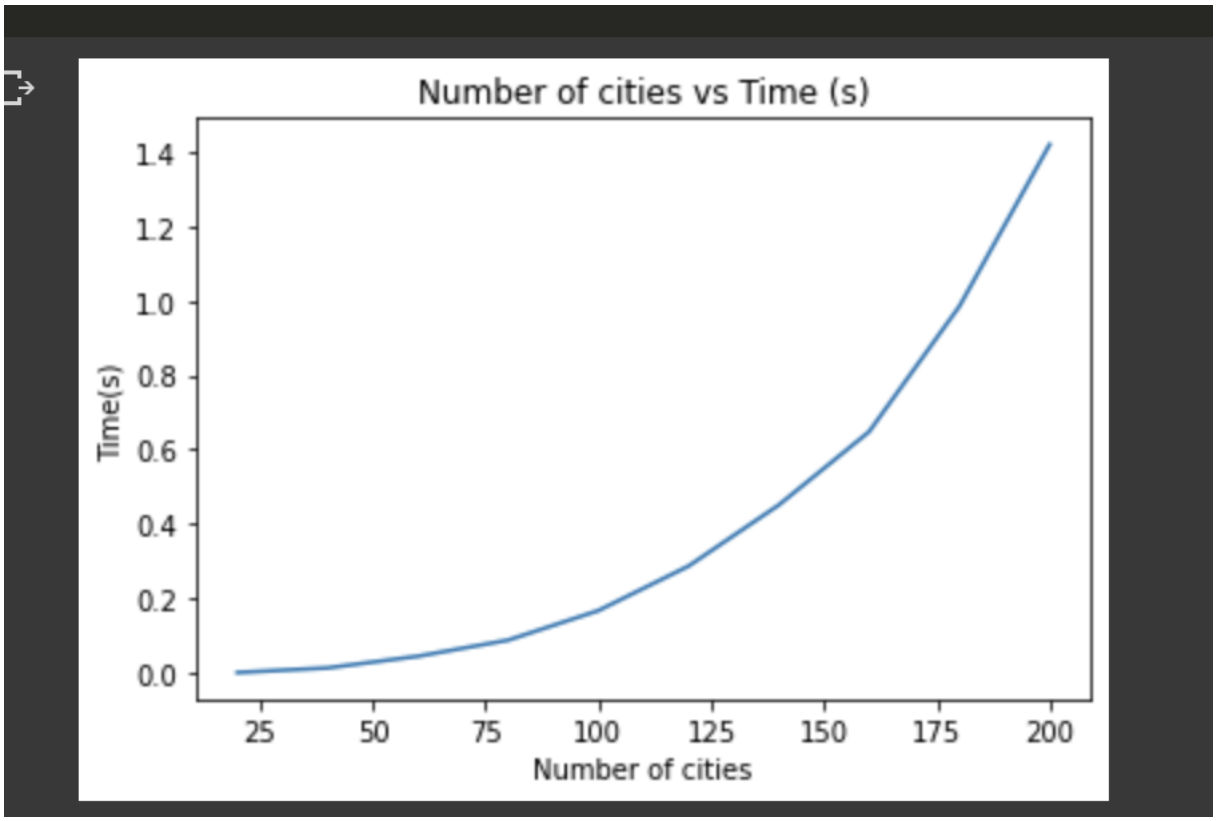
4 (d) Performance Plots

System: macOS Monterey, Apple M1 3.2 GHz, 16GB RAM

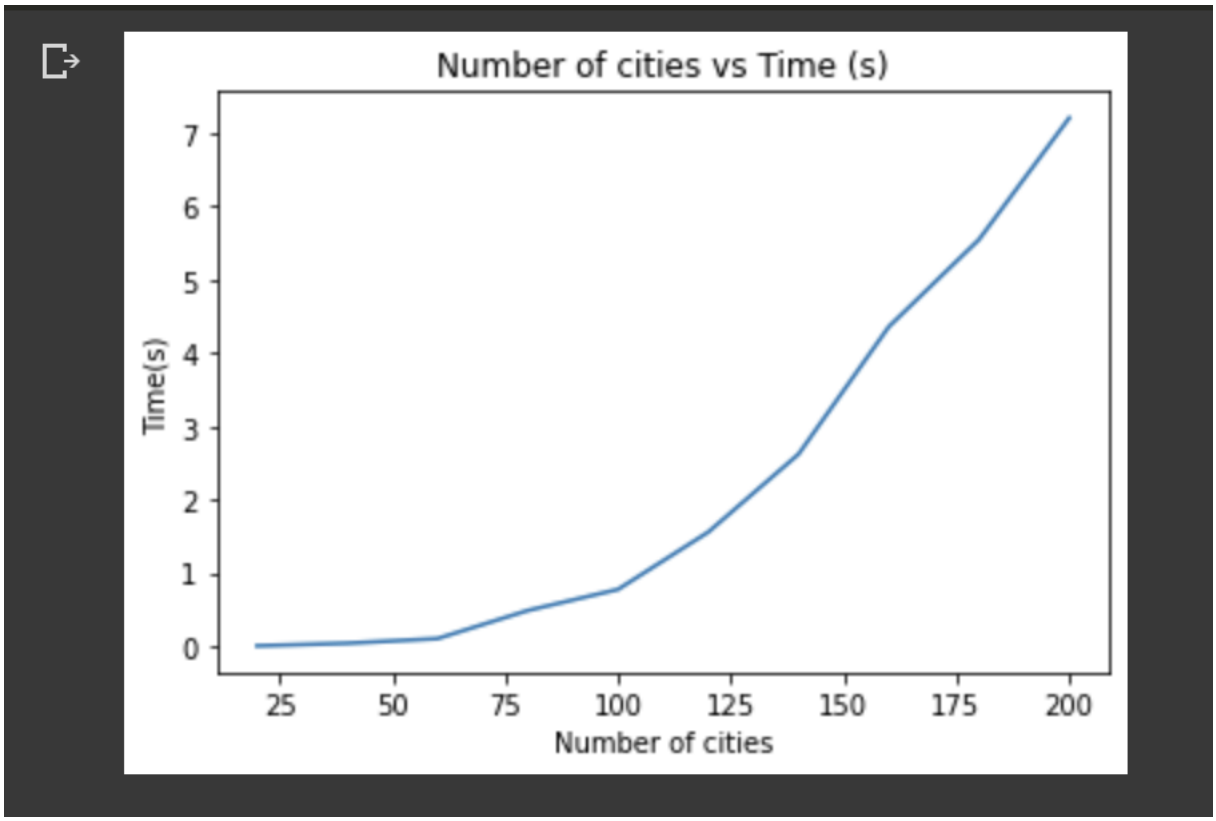
As it is expected, both average case (sparse graph) and worst case (dense graph) scenarios are seen as polynomial functions.

When those two scenarios are visualised together, it can be clearly seen that dense graph cause the algorithm to run in a higher polynomial order. Therefore, it can be said that, as expected, in worst-case (dense) the algorithm runs as $\Theta(|V|^3)$ and in average-case (sparse) it runs as $\Theta(|V|^2)$

AVERAGE CASE RUNNING TIME (SPARSE)



WORST CASE RUNNING TIME (DENSE)



AVERAGE CASE RUNNING TIME VS WORST CASE RUNNING TIME