CS301 - HW3

Tahir Turgut - 27035

May 2, 2021

(a) $\delta(s, v)$ is the formula to go from s to v.

$\delta(s, s) = 0$

$\delta(s, v) = \min(\delta(u, v) + w(s, u))$ where there is and edge between s and u

Subproblems: Starting from source, program will return the minimum road from neighbours of the source to target. Hence, subproblems are adjacents of the caller.

Optimal Substructure: For each subproblem, program will ensure that from source to any intermediary node is the optimal solution. Because, it is not possible to go longer path from u (an intermediary node between source and target) to target, since it will break the optimality of general problem, as well.

Overlapping Computations: When there is edge between at least two of the parents (u1 and u2, lets say) (closer to source from target), one of them will call the other one (u1 will call u2) and subproblem $\delta(s, u2)$ will be solved more than once.

Topological order: source, $u_1, u_2, ..., u_n, v_{1.1}, ..., v_{1.n}, v_{2.n}, ..., v_{2.n}, ..., target$ ($u_{1...n}$ are children of *source* and $v_{2.1...n}$ are children of u_2)

Hence, the problem is the find the shortest path from source to target, dividing the problem into subproblems which are children of source.

(b) **procedure** shortestRoute(*x, visited, target*):

    if x equals *target*:

        return 0

    if no neighbour left:

        return inf

    distlist = [ ]

    for each *adjacent* and not in *visited*:

        add *dist(x-adjacent) + shortestRoute(adjacent, visited, target)* to *dislist*

        add *adjacent* to *visited* //to avoid cycles (going back to the parent calls)

    return *min(distlist)*

    for every *city* except the *Istanbul*:

        shortestRoute(*Istanbul*, [ ], *city*)

Time complexity:

Target will be every other city than Istanbul = $\Theta(V)$

From source to target at most $|V| - 1$ vertices may exist (height of the tree) = $O(|V|)$

Let n be the number of calls in each step: In total there will be $n^{|V|}$ calls.

Thus, time complexity will be $O(|V| \cdot n^{|V|})$

Space complexity:

Every call will have its own distlist which may at most consist of $|V| - 1$ elements.

Since there is $n^{|V|}$ calls, complexity will be $O(|V| \cdot n^{|V|})$

(c) **procedure** shortestRouteDP(*vertices, edges, source*):

distlist = *inf* **for** size of $|V|$

parent = *null* **for** size of $|V|$

distlist[source] = 0

loop $|V| - 1$ times:

  for each *edge(u,v)* in *edges*:

    if distlist[u] + weight of edge < distlist[v]:

      distlist[v] = distlist[u] + weight of edge

      parent[v] = u

return distlist, parent

Time complexity:

Greater loop iterates $|V| - 1$ times = $\Theta(|V|)$

Inner loop iterates over each edge = $\Theta(|E|)$

Thus in total $\Theta(|V| \cdot |E|)$

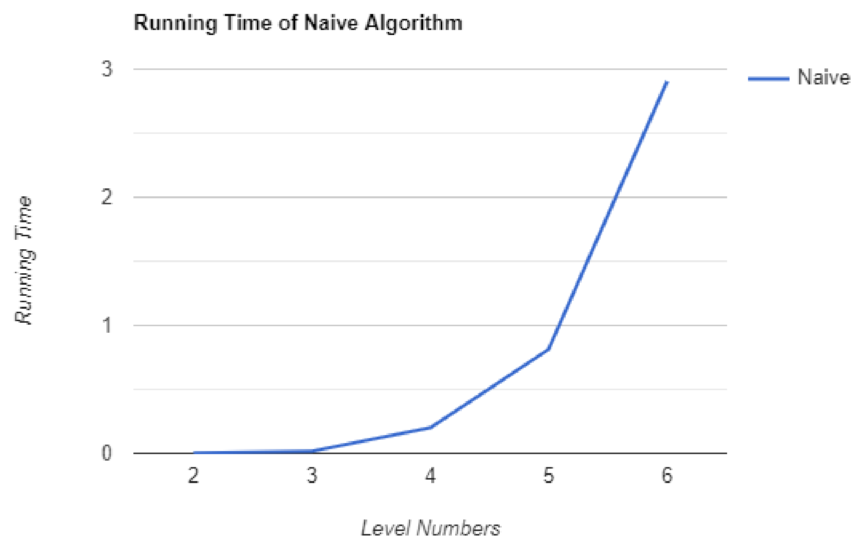
Space complexity:

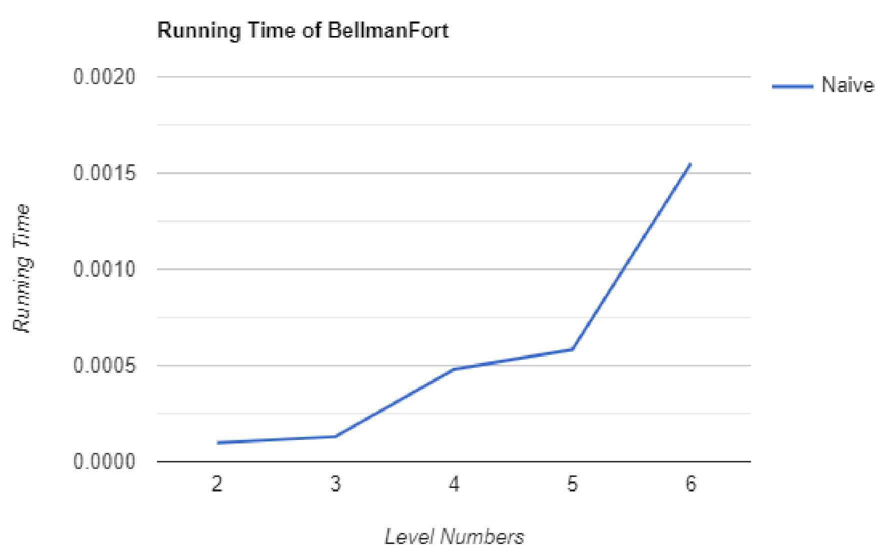
Both distlist and parent has $|V|$ elements = $\Theta(|V|)$

(d)

Computer properties: Windows10 OS, 16GB RAM, intel i7-770HQ CPU (2.8 gHz)

Algorithm	level = 2	level = 3	level = 4	level = 5	level = 6
Naive	0.0031006	0.0157071	0.199261	0.812308	2.9061138
BellmanFort	0.0000974	0.0001302	0.0004793	0.0005832	0.0015508





As expected, naive algorithm grows exponentially, even though the input size is small, the exponential trend can be observed.

On the other hand, BellmanFort is expected to show a trend between linearly and quadratic, however its data plots are quite messy. It can be steemed from the small input sizes. Even though, it is scattered (huge jump) from 5 to 6, if a trend is drawn, it will be near to a quadratic line.