Algorithms and Data Structures (ECE 345)

Issued on: Mar 4th, 2021
Due on: Mar 18th, 2021

ASSIGNMENT 3

1. Assignments must be submitted by 5:00PM EST on the due date through the Quercus submission system as a single PDF file.
2. Assignments must be completed **individually except the programming exercise** for which you can work in group of up to **two** students. Report for the programming exercise must be submitted separately at the same time. Only one report is needed for each group.
3. All pages must be numbered and no more than a single answer for any question.
4. Use LaTeX or Microsoft Word for your assignment writeup. You can find a LaTeX and a Word template for writing your assignment on Quercus.
5. Any problem encountered with submission must be reported to the head TA as soon as possible.
6. Unless otherwise stated, you need to justify the correctness and complexity of algorithms you designed for the problems.

EXERCISE 1 Amortized Analysis, 10 points

Consider the priority queue data structure as discussed in CLRS(Chapter 6.5). We implement a new operation $DELETE(k)$ which removes the k largest elements from the priority queue. Use the accounting method to show that $DELETE(k)$ can be done in amortized time $O(k)$ while maintaining an amortized time $O(logn)$ for $HEAP\_INSERT$. Describe your implementation for $DELETE(k)$. Do not change the implementation of the other priority queue routines.

Hint: what amortized time bound can you obtain for $HEAP\_EXTRACT\_MAX$?

Exercise 2 Graph, 15 points

Given a set V of objects with a "distance" function $d : V \times V \to R^+$, with $d(v,v) = 0$ and a parameter k, we seek to group V into k "clusters" (that is, partition V into k disjoint sets): $V_1, V_2, ..., V_k$ such that the minimum distance between any pair of objects in two different clusters is maximized. Describe an algorithm to solve this problem and justify the correctness and complexity of this algorithm.

Hint: Construct a complete graph with the distance function.

Exercise 3 Topological Sort, 15 points

Alien has came to Earth and learned the English alphabet in order to communicate with humans! However, they took the order of letters differently and it is unknown to you. Given a set of strings consisting of *unique words* from this Alien language that are sorted *lexicographically*, describe an algorithm that uses topological sort to find the order of all unique letters appeared in the set.

Example:

words = ["wrt","wrf","er","ett","rftt"]

order of letters: w < e < r < t < f

Hint: A string s is *lexicographically* smaller than a string s' if at the first letter where they differ, the letter in s comes before the letter in s' in the alien vocabulary. If the first $MIN(LEN(s), LEN(s'))$ letters are the same, then s is smaller if and only if $LEN(s) < LEN(s')$.

Exercise 4 Shortest Path, 20 points

Assume there is a directed graph $G = (V,E)$ with edge weights $w_i \in \{1,...,M\}$ and M is a constant in this case. Describe an algorithm with time complexity of $O((V+E)\log M)$ using Dijkstra's algorithm as basis. In other words, making the Dijkstra's algorithm more efficient by leveraging the fact that M is a constant through changing its underlying data structure without changing its outline.

Exercise 5 Programming Exercise, 30 points + 20 bonus points

Motivated by the ever growing size of social networks, and the abundance of data describing the underlying interactions and relationships in these networks, researchers in academia and industry have been studying various problems in this domain. Examples of such problems include network diffusion and influence maximization. Network diffusion is the process through which information is propagated over a network, where information can be in the form of a virus spreading across a population, an opinion emerging over a social network, or the adoption of a recently deployed product. The literature is rich with models that capture the dynamics of information propagation over networks [1].

In this exercise we will discuss the problem of influence maximization, which has received great interest over the last decade. In its simplest form, influence maximization is the problem of identifying those few individuals that play a fundamental role in maximizing the spread of information among users of a network. In past work, Domingos and Richardson [2] were the first to pose influence maximization as one of the quintessential algorithmic problems in network diffusion systems. Given a social graph along with estimates on how individuals influence each other, the goal is to find these individuals that should be initially targeted by a marketing campaign so that a new product receives the largest possible adoption rate in the network. To identify these so called “seeds", the authors in [2] propose heuristic algorithms and apply them on a probabilistic model of member interactions. In their seminal paper, Kempe et al. [3] formulated -for the first time- the influence maximization problem as a constrained discrete maximization problem, and they proposed two basic diffusion models, namely, the independent cascade (IC) model and the linear threshold (LT) model.

In this exercise we will experiment with a simplified version of the IC model in the continuous-time domain. To do so, we first need to understand how influence spreads across nodes in a network.

Without loss of generality we will assume that networks in this study correspond to social networks. A social network (network) is modelled as a finite directed graph $G = (V,E)$, where each node $u \in V$ corresponds to a user in the network, and each edge $(u,v) \in E$ implies a social connection (dependence) between users u and v. For example, a directed edge (u,v) may correspond to the fact that user u is followed by (or is a friend of) user v. If there exists an edge $(u,v) \in E$ then we also say that v is a neighbour of u. Along these lines, the set of all neighbours of u is denoted as $N(u)$.

The IC Model (simplified) In this model every edge $(u,v) \in E$ is assigned with some weight $t_{uv} \in \mathbb{R}_+$. We say that node u influences node $v \in N(u)$ after time t_{uv}. Once node v becomes influenced by u the spread of influence continues by v influencing its neighbours, then v's neighbours influencing their neighbours and so on. Once a node is influenced it remains in that state and cannot be influenced again. It becomes apparent that the further apart two nodes are in the graph the longer it will take for one to influence another. Also, if node v is unreachable from u in G, then u can never influence v through the process above.

Often, in real-world marketing scenarios we are interested in computing the spread of influence within specific "deadlines". A marketer usually wants to estimate the adoption that a campaign achieves within a few hours or days rather than in a few years. As such we have an external positive constraint, referred to as the "deadline", which is denoted as $T \in \mathbb{R}_+$. This deadline essentially tells us up to what point in time we are interested to measure the spread of influence, and it gives rise to a very useful property of the IC model:

The Shortest Path Property The shortest path property says the following: if $s(u,v)$ is the length of the shortest path from node u to node v, then u influences node v after time $s(u,v)$. Given deadline T we say that u influences v within time T if and only if $s(u,v) \leq T$.

Now we can define the **spread** of a node u given deadline T as the **number of nodes** that u can influence within deadline T. We denote the spread of u as $\sigma(u)$, and we formally have:

$$\sigma(u) = |\{v \in V : s(u,v) \leq T\}| \tag{5.1}$$

Note that node u by definition influences itself, since $s(u,u) = 0$ and $T \geq 0$ always.

With this setting, one interesting task is to identify, for a given T, which node in G is the most influential. That is, which node has maximum spread over all other nodes. We usually refer to this node as the Top-1 influencer.

Computing the Top-1 Influencer (30 points)

The task is simple:

1. Pick a node $u \in V$ and run shortest paths with u as the source
2. Enumerate how many nodes in V have $s(u,v) \leq T$. This is the spread of u.
3. Initialize all distances back to ∞ and repeat Step 1 and 2 for every other node in V
4. Return the node with maximum spread. If there are ties, break them randomly.

Note that in Step 3 above we first initialize every distance back to ∞ and then run shortest paths again.

Identifying the Top-1 influencer is important for a campaign but is rarely enough to achieve a good spread across the network. The more influencers we identify the better the spread of influence usually becomes. That is why we usually talk about the Top-K influencers (if our marketing budget allows us to initially target these K individuals with ads, personal contact etc). Here we will just focus on finding the Top-2 influencer. You might think that the Top-2 influencer is the one with the second best spread in the process we described above, but that is not the case. Imagine that the node with second best spread achieves 99% of the spread of the Top-1 influencer, but all the nodes that it influences are already nodes that the Top-1 influencer can reach by itself. There wouldn't be any reason to use the node with second best spread in that first round of computations due to this redundancy. This is where the notion of **marginal spread gain** comes into play. To understand it we first need to define the spread of two nodes.

The **spread** of two nodes u and w given deadline T is the **number of nodes** that u OR w can influence within deadline T. We denote the spread of u and w as $\sigma(u,w)$, and we formally have:

$$\sigma(u,w) = |\{v \in V : [s(u,v) \leq T] \vee [s(w,v) \leq T]\}| \tag{5.2}$$

Now that we know how to compute the spread of two nodes, we can define the marginal spread gain of node w as $\sigma(u,w) - \sigma(u)$. The marginal spread gain essentially gives us "the number of extra nodes that we can influence if we use w along with u". In our case, the Top-2 influencer is the one that will maximize the marginal gain when added to the Top-1 influencer. Finally note that $\sigma(u,w) - \sigma(u) \geq 0$, since you cannot influence fewer nodes if you use an additional influencer.

Computing the Top-2 Influencer (20 points, bonus)

1. Run shortest paths with the Top-1 influencer as the source, and mark all nodes influenced within T as `influenced`
2. Pick a node $w \in V$ other than the Top-1 Influencer and run shortest paths from w
3. Enumerate how many nodes in V have $s(w,v) \leq T$ and are not already marked as `influenced`. This is the marginal spread gain of w.
4. Initialize all distances back to ∞ and repeat Step 2 and 3 for every other node in V other than the Top-1 Influencer
5. Return the node with maximum marginal spread gain. If there are ties, break them randomly.

In this exercise you will implement the above two processes for finding the Top-1 and Top-2 influencers in a directed graph that represents a social network.

Input:

Your executable will be named `influence`. It will be given an input file named `graph` which represents edges and edge weights between nodes. It will be a 3-column file and each row will be of the format:
`node` u `node` v `weight`
The row above indicates the existence of a directed edge (u,v) in G with weight t_{uv} = `weight`. Entries are separated by "space". For simplicity node names will be a $0,\ldots,|V|-1$ enumeration, if $|V|$ is the number of nodes, and indexing will start at 0 always. Finally, you may assume that weights will be randomly and uniformly chosen from the range $(0,5]$. There can be cycles in the graph but there will be no zero or negative edge weights. For example consider the following file:

```
0 2 1.5
1 2 0.4
1 3 2.9
2 3 3.3
3 5 0.1
3 4 2.4
4 1 0.7
5 6 0.7
```

The above file corresponds to the graph in Figure 1. Your program will also take as input a deadline constraint T, which you may assume will be a positive real number in the range $[1,10]$.

To summarize, your executable should be callable from command line as:

```
influence graph T
```

Output:

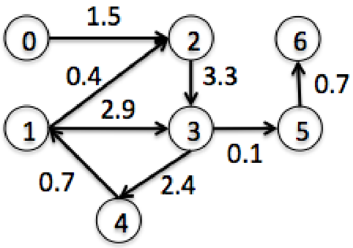

Figure 5.1: Example Graph

Your program will compute the Top-1 and Top-2 Influencers by following the process described above. Specifically you will have to compute and output the following:

1. The Top-1 Influencer, its spread, and the time taken to find the Top-1 Influencer (you will measure time for all computations including creating the graph, running shortest paths and finding the max spread)
2. The Top-2 Influencer, its marginal spread gain, and the time taken to find the Top-2 Influencer (here you will measure the time for all computations that take place AFTER the Top-1 Influencer is found)

Your program will print these results to standard output as follows:

```
TOP-1 INFLUENCER: (node name), SPREAD: (spread), TIME: (time)
TOP-2 INFLUENCER: (node name), MARGINAL SPREAD: (spread), TIME: (time)
```

In the example of Figure 1, if the program is run with $T = 3$ the output should be:

```
TOP-1 INFLUENCER: 1, SPREAD: 4, TIME: 2.5 sec
TOP-2 INFLUENCER: 3, MARGINAL SPREAD: 2, TIME: 1.2 sec
```

You can verify this by checking that there are 4 nodes with their shortest path from node 1 being $\leq T$. Particularly, nodes $1, 2, 3$ and 5. It is also the case that there are 4 nodes with their shortest path from node 3 being $\leq T$ (nodes 3, 4, 5 and 6). Remember you always include the influencers in their spread unless they are already marked as influenced by a previous one. Thus, nodes 1 and 3 have spread 4 which is actually the max spread. Randomly break the tie and pick node 1 as the Top-1 Influencer. Now note that node 3 has marginal spread 2 because adding it to node 1, allows nodes 4 and 6 to be influenced as well. Strictly speaking, $\sigma(1,3) - \sigma(1) = 6 - 4 = 2$. You can verify that the marginal spread of the other nodes is smaller than 2. Therefore, node 3 is the Top-2 Influencer. Note that the times in this example are fictional.

Deliverables:

- Your source code, including a full build environment and instructions how to build in the configuration `TOML` file. Please follow a similar style as provided with the A2 coding problem.
- You are encouraged to use external libraries but you need to make sure to include any libraries in your submission so that we can compile the code.
- A written report as part of your Assignment 3 submission, where you address the following points:
  - Implementation details: (a) how are you representing the graph, using an adjacency matrix or an adjacency list, and why? (b) Which shortest path algorithm did you use and why?
  - You will need to use multiple graph files of your making (you will be provided a large file as well) to run the following experiment: create 10 different files such that all have 100 nodes, but with different density = (# edges)/(# nodes). For example, a graph (file) with 100 nodes and 300 edges (rows) has density 3. Density should range between 2 and 10. Run your code and show the results in a **time vs. density** plot. When we refer to time we mean the total time taken to compute the two top influencers (i.e., 3.7 sec in the example above).

- Discuss the plot above. What do you observe and why?

References

[1] M. Jackson. “Social and Economic Networks" Princeton University Press, Princeton and Oxford, 2008

[2] P. Domingos, M. Richardson. “Mining the Network Value of Customers," in International Conference on Knowledge Discovery and Data Mining (KDD), 2001

[3] D. Kempe, J. M. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network," in International Conference on Knowledge Discovery and Data Mining KDD, 2003