COMP 182: Homework 3

Quan Le

Monday, February 17, 2020

Problem 1

Growth of Functions [25 pts]

1. Show that $1^k + 2^k + ... + n^k = O(n^{k+1})$ for $k, n \in \mathbb{N}$ and $k, n \geq 1$.

$$\begin{aligned} 1^k + 2^k + ... + n^k &\leq n * n^k = n^{k+1} \\ n^{k+1} &= O(n^{k+1}) \\ \therefore \quad 1^k + 2^k + ... + n^k &= O(n^{k+1}) \end{aligned}$$

2. Show that for all real numbers a and b with $a > 1$ and $b > 1$, if $f(x) = O(\log_b x)$, then $f(x) = O(\log_a x)$.

$$\begin{aligned} & f(x) = O(\log_b x) \\ \exists C \quad & f(x) \leq C \log_b x \\ & = C \frac{\log_a x}{\log_a b} \end{aligned}$$

$$\frac{C}{\log_a b} \text{ is constant} \qquad \therefore f(x) = O(\log_a x)$$

3. Show that the two functions $\log(x^2 + 1)$ and $\log(x)$ are of the same order (recall that "same order" is established using Θ).

$$\begin{aligned} \log(x^2 + 1) &< 4 \log(x) \quad \forall x > 2 \\ \log(x^2 + 1) &> \log(x) \quad \forall x > 0 \end{aligned}$$

The first statement is true since

$$\lim_{x \to \infty} \frac{\log(x^2 + 1)}{4 \log(x)} = \lim_{x \to \infty} \frac{\frac{2x}{x^2+1}}{\frac{4}{x}} = \lim_{x \to \infty} \frac{x^2}{2x^2 + 2} = \lim_{x \to \infty} \frac{2x}{4x} = 1/2 \neq 0, \infty$$

The second is true since $x^2 + 1 > x \ \forall x > 0$.

$$\begin{aligned} \log(x^2 + 1) &= O(\log(x)) \\ \log(x^2 + 1) &= \Omega(\log(x)) \\ \therefore \log(x^2 + 1) &= \Theta(\log(x)) \end{aligned}$$

4. Let $f(x), g(x)$, and $h(x)$ be three functions such that $f(x) = \Theta(g(x))$ and $g(x) = \Theta(h(x))$. Prove that $f(x) = \Theta(h(x))$

$$f(x) = \Theta(g(x)) \wedge g(x) = \Theta(h(x))$$
$$f(x) = O(g(x)) \wedge g(x) = O(h(x))$$
$$f(x) \leq C_1 g(x) \wedge g(x) \leq C_1 h(x)$$
$$f(x) \leq C_1 C_2 h(x)$$
$$f(x) = O(h(x))$$

$$f(x) = \Theta(g(x)) \wedge g(x) = \Theta(h(x))$$
$$f(x) = \Omega(g(x)) \wedge g(x) = \Omega(h(x))$$
$$f(x) \geq C_1 g(x) \wedge g(x) \geq C_1 h(x)$$
$$f(x) \geq C_1 C_2 h(x)$$
$$f(x) = \Omega(h(x))$$

Therefore, $f(x) = \Theta(h(x))$

5. Arrange the functions $2^{100n}, 2^{n^2}, 2^{n!}, 2^{2n}, n^{\log n}, n \log n \log\log n, n^{3/2}, n(\log n)^{3/2}$, and $n^{4/3}(\log n)^2$ in a list so that each function is big-O of the next function. Briefly justify your answer.

$n \log n \log\log n, n(\log n)^{3/2}, n^{3/2}, n^{4/3}(\log n)^2, n^{\log n}, 2^{100n}, 2^{n^2}, 2^{2^n}, 2^{n!}$. This order was chosen based on $\exists k \forall n > k (1 < \log n < n < n \log n < n^2 < 2^n < n!)$, as shown in the slides, and the following arguments.

$$\begin{aligned} n \log n \log\log n &\quad n(\log n)^{3/2} \\ \log\log n &\quad (\log n)^{1/2} \\ \log\log\log n &\quad (1/2)\log\log n \\ \log x = O(x/2) &\implies n \log n \log\log n = O(n(\log n)^{3/2}) \end{aligned}$$

$$\begin{aligned} n(\log n)^{3/2} &\quad n^{4/3}(\log n)^2 \\ 1 &\quad n^{1/3}(\log n)^{1/2} \\ 1 = O(n^{1/3}(\log n)^{1/2}) &\implies n(\log n)^{3/2} = O(n^{4/3}(\log n)^2) \end{aligned}$$

$$\begin{aligned} n^{4/3}(\log n)^2 &\quad n^{3/2} \\ (\log n)^2 &\quad n^{1/6} \\ 2\log\log n &\quad (1/6)\log n \\ 2\log(x) = O(x/6) &\implies n^{4/3}(\log n)^2 = O(n^{3/2}) \end{aligned}$$

$$\begin{aligned} n^{3/2} &\quad n^{\log n} \\ (3/2)\log n &\quad (\log n)^2 \\ 3/2 = O(\log n) &\implies n^{3/2} = O(n^{\log n}) \end{aligned}$$

$$\begin{aligned} n^{\log n} &\quad 2^{100n} \\ (\log n)^2 &\quad (100n)\log 2 \\ \log\log n &\quad \log n + \log(100 \log 2) \\ \log x &\quad x + \log(100 \log 2) \\ \log x = O(x) &\implies n^{\log n} = O(2^{100n}) \end{aligned}$$

Algorithms and Their Complexity [10 pts]

In class, we saw Algorithm **MatrixMultiplication** for multiplying two matrices and discussed its running time. Section 10.4 of your textbook establishes a theorem about the number of paths of any given length between a pair of nodes in a graph using the adjacency matrix of the graph and raising it to the appropriate power.

1. Using this theorem, write the pseudo-code of Algorithm **ComputeNumberOfShortestPaths** that takes as input the adjacency matrix A of a graph g (assume its nodes are numbered $0, 1, ..., n-1$), an edge e in the graph, and two nodes i, j in the graph, and computes/returns the number of shortest paths (recall: the length of a path is the number of edges on it) between i and j that go through e in g. *Do not use Algorithm **BFS** here. Your algorithm must make use of Algorithm **MatrixMultiplication** and be based on the Theorem in Section 10.4 of the textbook.*

Algorithm 1: ComputeNumberOfShortestPaths

Input: A, an adjacency matrix of a graph g, whose nodes are numbered $0, 1, ...n-1$
edge $e = \{k, l\}$ in graph g
nodes i, j in graph g
Output: the number of shortest paths between i, j that go through e

```
1  M ← I_{n×n}
2  for r = 1, 2, ..., n − 1 do
3  |  M = M * A
4  |  if M_{i,j} ≠ 0 then
5  |  |  A' ← A
6  |  |  A'_{k,l}, A'_{l,k} ← 0
7  |  |  M' ← I_{n×n}
8  |  |  for a = 1, 2, ...r do
9  |  |  |  M' ← M' * A'
10 |  |  return M_{i,j} − M'_{i,j}
11 return 0
```

2. Using big-O notation, what is the worst-case running time of your Algorithm **ComputeNumberOfShortestPaths** on a graph with n nodes? Clearly state what the worst case corresponds to in this case, assuming the distance between pairs of nodes could be as large as $n-1$.

   Since matrix multiplication is the vector-wise dot-product of each column of the second matrix with each row of the first, and since the dot product of two vectors is $O(n)$ (element-wise multiplication and addition), we can conclude that the multiplication of two square matricies is $O(n^3)$.

   From here, it is clear that the worst case is when the shortest paths between two nodes intersect all nodes in g. Therefore, the initial for-loop is computed $n-1$ times, and the matrix multiplication is done with $O(n^3)$. The conditional statement is also computed in the worst case, where $M' * A'$ is computed $n-1$ times. Therefore, the worst case is $O(n^4)$

Sequences and Summations [10 pts]

Several algorithms have a loop structure that iterates n times and for the k-th iteration, they perform on the order of k^2 operations. In this case, the running time of the loop can be obtained by $\sum_{k=1}^{n} k^2$. In this problem, we will derive a closed formula for this sum and show that the sum is $O(n^3)$.

1. A telescoping sum is a sum of the form $\sum_{j=1}^{n}(a_j - a_{j-1})$. It is easy to see that this sum equals $a_n - a_0$ (try it for example for $j = 4$). In class, we stated that $\sum_{k=1}^{n} k^2 = n(n+1)(2n+1)/6$. Derive this formula using the telescoping sum. *Hint:* Take $a_k = k^3$ in the telescoping sum.

$$\sum_{k=1}^{n}(k^3-(k-1)^3)=n^3-0$$

$$\sum_{k=1}^{n}(k^3-(k-1)(k^2-2k+1))=n^3$$

$$\sum_{k=1}^{n}(k^3-(k^3-3k^2+3k-1))=n^3$$

$$\sum_{k=1}^{n}(3k^2-3k+1)=n^3$$

$$3\sum_{k=1}^{n}k^2-3\sum_{k=1}^{n}k+n=n^3$$

$$3\sum_{k=1}^{n}k^2-\frac{3(n+1)n}{2}=n^3-n$$

$$\sum_{k=1}^{n}k^2=\frac{1}{3}\left[\frac{3(n+1)n}{2}+n^3-n\right]$$

$$\sum_{k=1}^{n}k^2=\frac{1}{6}\left[3(n+1)n+2n^3-2n\right]$$

$$\sum_{k=1}^{n}k^2=\frac{1}{6}\left[3n^2+3n+2n^3-2n\right]$$

$$\sum_{k=1}^{n}k^2=\frac{n}{6}\left[3n+2n^2+1\right]$$

$$\sum_{k=1}^{n}k^2=\frac{n}{6}(2n+1)(n+1)$$

2. Show that $\sum_{k=1}^{n}k^2=O(n^3)$ by explicitly finding the values of the constants k and C and demonstrating the result.

$$C=1,k=1$$

$$\sum_{k=1}^{n}k^2=n(n+1)(2n+1)/6\overset{?}{\leq}n^3$$

$$2n^3+3n^2+n\overset{?}{\leq}6n^3$$

$$3n^2+n\overset{?}{\leq}4n^3$$

$$3n+1\overset{?}{\leq}4n^2$$

$$0\overset{?}{\leq}4n^2-3n+1$$

$$0\overset{?}{\leq}(4n+1)(n-1)$$

$$n>k=1\implies(4n+1),(n-1)>0.$$

Therefore, $\sum_{k=1}^{n}k^2=O(n^3)$

Problem 2

Breadth-first Search and Its Applications

We saw in class the pseudo-code of Algorithm **BFSPreview** the document. Furthermore, as discussed in class, this algorithm performs on the order of $m + n$ operations on a graph with n nodes and m edges. In this problem, assume the graph $g = (V, E)$ has n nodes and m edges and is represented by its adjacency list.

1. [10*pts*] Give the pseudo-code of Algorithm **ComputeDistances** that, given a graph $g = (V, E)$ and a source node $i \in V$, computes the distance d_j from node i to every other node $j \in V$ and performs on the order of $m + n$ operations. In particular, $d_i = 0$ and $d_j = \text{inf}$ if node j is not reachable from node i. Discuss that your algorithm indeed performs on the order of $m + n$ operations. *Hint: Think about replacing v_j by d_j throughout the algorithm.*

Algorithm 2: Compute Distances

Input: Graph $g = (V, E)$; source node $i \in V$
Output: $d_j \in \mathbb{N} \cup \{0\}$ $\forall j \in V$, the distance from i to each node in g

```
1  foreach j ∈ V do
2      d_j ← ∞
3  d_i ← 0
4  Initialize Q to an empty queue
5  enqueue(Q, i)
6  while Q is not empty do
7      j ← dequeue(Q)
8      foreach neighbor h of j do
9          if d_h = ∞ then
10             d_h ← d_j + 1
11             enqueue(Q, h)
12 return d
```

Consider the number of operations each step takes: the initialization of d takes n operations, one for each node; and the while loop looks at each edge twice, thus the algorithm performs another $O(2m)$ operations. Therefore, the algorithm operates on $O(m + n)$.

2. [10*pts*] Give the pseudo-code of Algorithm **IsBipartite** that, given a graph $g = (V, E)$, determines whether the graph is bipartite and takes on the order of m+n operations. You may assume in this part that the input graph g is connected. Discuss that your algorithm indeed performs on the order of $m + n$ operations. *Hint: Think of slightly modifying Algorithm **BFS** to color the nodes of the graph.*

Algorithm 3: IsBipartite

Input: connected graph $g = (V, E)$
Output: True, if g is bipartite, False otherwise

```
1  foreach j ∈ V do
2      v_j ← ∞
3  i ← an arbitrary node in V
4  v_i ← 1
5  Initialize Q to an empty queue
6  enqueue(Q, i)
7  while Q is not empty do
8      j ← dequeue(Q)
9      foreach neighbor h of j do
10         if v_h ≠ ∞ then
11             if v_h = v_j then
12                 return False
13         else
14             v_h ← −v_j
15             enqueue(Q, h)
16 return True
```

The argument for ComputeDistances applies to IsBipartite; the initialization of $v_j = O(n)$, and the while loop observes each edge twice ($O(m)$). Finally, note that the choice of an arbitrary node in V scans through V but selects the first element it encounters and is thus $O(1)$.

Therefore, it can be concluded that IsBipartite is $O(n + m)$

3. [10*pts*] Give the pseudo-code of Algorithm **ComputeLargestCCSize** that, given a graph $g = (V, E)$, computes the size (in terms of the number of nodes) of the largest connected component of g. Your algorithm must perform on the order of $m+n$ operations in the worst case. Discuss that your algorithm indeed performs on the order of $m + n$ operations. If you decide to use sets and set operations, you have to be clear on how long set operations take.

Algorithm 4: ComputeLargestCCSize

Input: Graph $g = (V, E)$
Output: $n \in \mathbb{N}$, the size of the largest connected component

```
1  foreach j ∈ V do
2      v_j ← ∞
3  Initialize Q to an empty queue
4  n ← 0
5  foreach i ∈ V do
6      if v_i = ∞ then
7          n' ← 1
8          v_i ← 1
9          enqueue(Q, i)
10         while Q is not empty do
11             j ← dequeue(Q)
12             foreach neighbor h of j do
13                 if v_h = ∞ then
14                     n' ← n' + 1
15                     v_h ← 1
16                     enqueue(Q, h)
17     if n < n' then
18         n ← n'
19 return n
```

In the worst case, the algorithm initializes v_j in $O(n)$ operations. Then, for each component, the algorithm computes a modified version of BFS. Clearly, an edge cannot be in more than one component. As such, the algorithm still only covers each edge twice. Thus, over all of the components, the algorithms runs for another $O(m)$ operations. Therefore, ComputeLargestCCSize has running time $O(m + n)$.

Problem 3

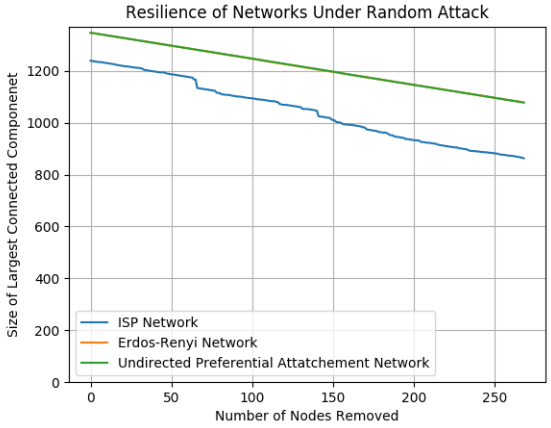
Network Resilience

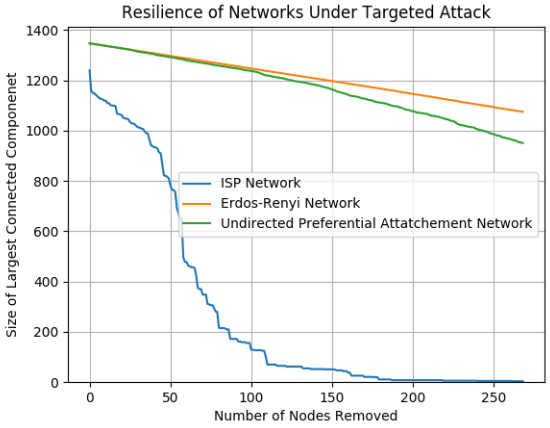
Graph Resilience

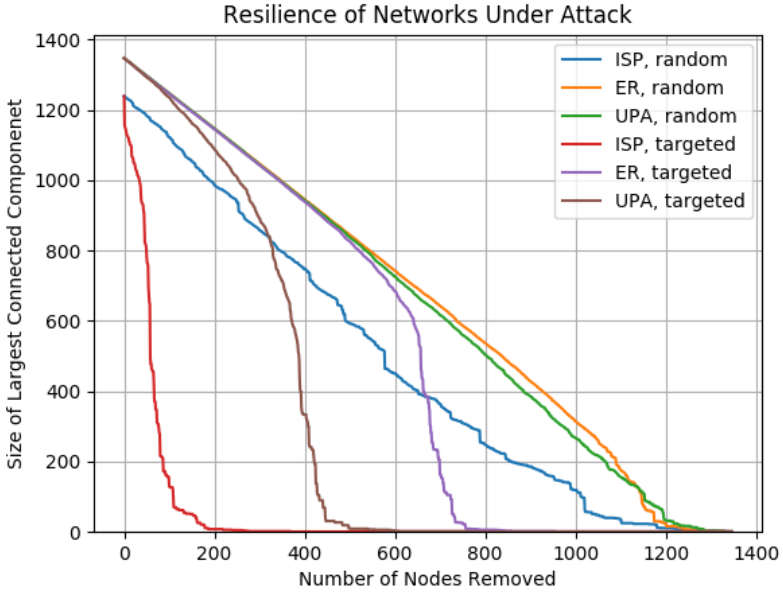
Within 20% deletion of nodes, it seems the the Erdos-Renyi (ER) and Undirected Preferential Attachment (UPA) networks seem equally resilient under random attack, with both being more resilient than the given ISP network.

Considering the same networks under targeted attack, the size of the largest connected component of the ER Network diverges from that of the UPA network, with the UPA network lagging behind. In addition, the ISP network is thoroughly destroyed, with the size of the max connected component decreasing rapidly between the 75-100 nodes removed mark.

When the experiment is considered over all of the nodes, the ER Network demonstrates ever so slightly more resilience than the UPA Network under random attack. Under targeted attack, the decline curve for the ISP Network once again appears in the declines of the USP and then the ER Networks over the number of nodes removed.







Differences Between ER and UPA

The UPA network is that of several very strongly connected nodes, as well as other nodes of varying degree (who maintain the average edges per node). In comparison, the ER network is that where the nodes are connected with a given probability.

As such, we expect the ER network to fare better than the UPA network, as the ER network distributes the edges better, whereas the UPA network has edges that are more concentrated around the same nodes.

If my main concern was targeted attacks, then I would use the ER network; it most strongly protects from them out of the other options.

In comparison to the other methods of constructing a graph, the provided network had worse resilience than either of the provided random networks. However, it is understood that the provided network has influences from the real world.

Parameter Values

The parameter m was chosen as the number of edges per node: the number of nodes and edges in the ISP network were found, the ratio was found and used to generate the UPA network.

The parameter p was chosen as the probability of having an edge between two nodes: the number of edges was divided by the number of possible edges, $n(n-1)/2$, where n is the number of nodes.