Network Optimization, Homework 0

Nedialko B. Dimitrov

1. (a) Please tell me a little about yourself. This will help us get to know each other and structure the class so that it is helpful. Answer the following (just a paragraph or so total):
   - What is your degree program, and how far into it are you?
   - Where were you before coming to UT?
   - Where would you like to go after UT? What are your goals after you are done with your education?
   - What do you hope to get out of this class specifically?
   - How much optimization experience do you have? Have you used a linear program or a mixed integer program in another class? Give an example project you've done?
   - Have you solved an LP or a MIP using some software before? Which software?

   An acceptable solution:
   Previously to coming to UT, I was faculty in the Operations Research Department at the Naval Postgraduate School for four years. There, I worked on a lot of military problems, and trained many military students. My specialty is in algorithms, so I primarily work on: 1) describing a real world decision using math and 2) creating the algorithms to compute a decision quickly. Besides working in academia, I've also worked at Amazon.com, and D.E. Shaw – the world's largest hedge fund by market capitalization – and I've tried to start my own mathematical modeling company with friends from graduate school.

   In this HW question, I used to ask people for something they can model as a graph. I don't ask that any more, but here is a fun story about something that can be modeled as a graph: Academic papers can be modeled as a graph. Each author would be a node, and two authors would be connected if they have published a paper together. You may have heard of something called the "Erdős" number. Paul Erdős was a famous traveling mathematician who had published many papers with people all over the world. The Erdős number of a person is the number of publications separating the person from Paul Erdős in the graph of publications described at the beginning of the paragraph. My Erdős number is 4.

---

2. What is the difference between a graph and a multigraph? In an undirected graph with self-loops, are edges described using a tuple, a set, or a multiset?

   An acceptable solution:
   A graph $G = (V, E)$ allows only one edge for each pair of nodes (a, b), and E is a set. A multigraph $G = (V, E)$ allows multiple copies of edges, and E is a multiset. In an undirected graph with self-loops, an edge is a multiset since the order of the edge does not matter and repetition is allowed.

---

3. In class we defined several graph-related terms in words and with mathematical symbols. For a graph $G = (V, E)$ and node $a \in V$, define predecessors(a) and successors(a) with mathematical symbols.

   An acceptable solution:
   predecessors(a) = $\{b \in V \mid (b, a) \in E\}$, successors($a$) = $\{b \in V \mid (a, b) \in E\}$

---

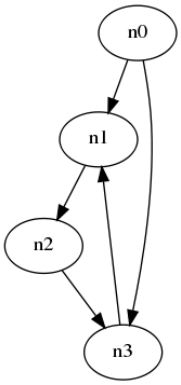
4. (a) Write the adjacency matrix of the graph in Figure 1, lets name the matrix A.
   (b) Write the transpose of A (denoted A^T).
   (c) Draw the graph whose adjacency matrix is A^T.
   (d) What is the relationship between the graph in Figure 1 and the graph you drew in problem 4c?

   An acceptable solution:

   (a) $A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$

   (b) $A^T = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

   (c)



   (d) The edges in the two pictures have reversed direction. *Punchline:* For a directed graph, if the adjacency matrix is A, then A^T is the adjacency matrix of the same graph with the edges reversed.

---

5. Write the matrix $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 12 \\ 0 & 13 & 0 & 0 \\ 14 & 0 & 15 & 0 \end{bmatrix}$ in:
   (a) Dictionary of keys (DOK) format
   (b) Row based linked list (RBLL) format
   (c) Compressed sparse row (CSR) format

(d) Compressed sparse column (CSC) format

An acceptable solution:

(a)

key	value
(1,0)	11
(1,3)	12
(2,1)	13
(3,0)	14
(3,2)	15

(b)

row	column list	data list
0	[]	[]
1	[0,3]	[11,12]
2	[1]	[13]
3	[0,2]	[14, 15]

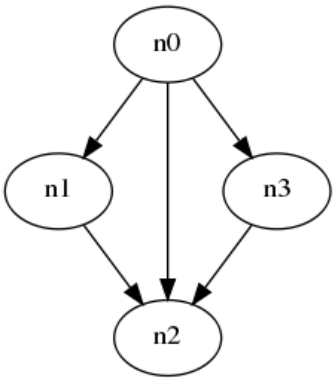
(c)
index pointer: 0,0,2,3,5
columns: 0,3,1,0,2
data: 11,12,13,14,15

(d)
index pointer: 0,2,3,4,5
rows: 1,3,2,3,1
data: 11,14,13,15,12

---

6. An adjacency matrix stored in CSR format has the following values:
   index pointer: 0,3,4,4,5
   columns: 1,2,3,2,2
   data: 1,1,1,1,1
   Draw the corresponding graph.

   An acceptable solution:



---

7. We are given a matrix A stored in CSR format.

   (a) What is the run time (in big-Oh) of finding A^T in CSC format? (Hint: For the A in problem 5, try writing A^T in CSC format.)

   (b) What is the run time (in big-Oh) for finding A in CSC format?

An acceptable solution:

(a) $O(1)$. CSR format for A is exactly the same as CSC format for A^T.

(b) $O(n+m)$. We can achieve that run time by iterating over all edges to build A in CBLL format, and then constructing A in CSC format. *Punchline:* Combining the answer with this problem to the answer of problem 5, we can conclude the following. Given graph G in CSR format, we can find the revesed graph, G^T, in CSC format very quickly... $O(1)$ time. But, our algorithms often require graphs to be in CSR format, since we need to be able to compute the successors of nodes quickly. Finding the reversed graph in CSR format takes $O(n+m)$ time.

---

8. (a) For an $n \times n$ sparse matrix with m non-zeros stored in CSR format, what is the length of "index pointer," "columns," and "data"?

   (b) We have an $n \times n$ sparse matrix with m non-zeros stored as a sparse matrix. We want to change a zero in position (i, j) to a non-zero. This is called the "insert (i, j)" operation. What is the run time (in big-Oh) of the "insert (i, j)" operation if the matrix is stored in DOK format? RBLL format? CSR format?

An acceptable solution:

(a) "index pointer" has length $n+1$, "columns" has length m, "data" has length m.

(b) DOK: $O(1)$, RBLL: $O(1)$, CSR: $O(n+m)$. The following table summarizes the properties of the three formats:

Format	Memory	Access (i, j)	successors(i)	insert (i, j)
DOK	$O(m)$	$O(1)$	$O(m)$	$O(1)$
RBLL	$O(n+m)$	$O(d_{\max})$	$O(1)$	$O(1)$
CSR	$O(n+m)$	$O(\log d_{\max})$	$O(1)$	$O(n+m)$

---

9. We have an $n \times n$ sparse matrix A with k non-zero entries in each row. We also have a dense vector v of length n with n non-zeros. What is the run time (run time is always in big-Oh) of computing $A \cdot v$ if A is stored in any of our sparse matrix formats? Write pseudo-code for computing $A \cdot v$ if A is stored in DOK format. (Hint: You may iterate over the hash table, receiving the key (i, j) and the data value d in each iteration. However, it is impossible to control the order in which the iteration is done.)

An acceptable solution:
All sparse matrix formats give a running time of $O(nk)$ for computing $A \cdot v$. The pseudo-code for computing $A \cdot v$ if A is in DOK format is as follows:

- Let a be a dense vector of n zeroes.
- Iterate over entries in the hash table. Let the current entry be (i, j) with data value d.
  - Let $a[i] = a[i] + d \cdot v[j]$
- Return a as the answer.

---

10. Give an example of something that can be modeled as a graph, where the resulting graph has over a billion nodes. "Devices on the internet" and "the person-to-person social network of the world" are good examples, but you should try to think of something else. What are the nodes of the graph? How do you know there is over a billion nodes? (any explanation, even Wikipedia is acceptable) What are the edges? What is your guess on how many edges are incident on each node? (any guess is acceptable)

    Can you give a graph whose full (dense, as in zeros included) adjacency matrix has a billion entries?

    An acceptable solution:
    An example that is similar to "the person-to-person social network fo the world" is the family tree of the world's population. Each node is a person, and two nodes share an edge if the people are parent and child. There are over a billion nodes because there are 6 billion people in the world. Each node has at least 2 edges, one for each of the person's parents, and at most, perhaps, 50 or 100? Other good answers can be derived from:

    (a) the neurons in the human brain, and synapse connections between them
    (b) search queries on Google.com, and if they are made by the same person
    (c) photographs, and if they include the same object
    (d) web pages, and if they link to each other

    Punchline: As long as we come up with one billion items and some relationship between those items, we can model the result as a graph. Many of the resulting graphs are sparse.

    It is much easier to come up with an example of a graph with 1 billion entries in the adjacency matrix. For that, all we need is about 40000 nodes. So, for example, students at UT and whether they have taken the same class together would work.

---

11. **[Theory Problem]**

    (a) Let A be the adjacency matrix of a graph G. What does the (i, j)'th entry of A^2 store?
    (b) What does the (i, j)'th entry of A^k store?
    (c) Prove your answer to the previous question is correct.
    (d) What is the maximum value a single entry of A^k could have?
    (e) What is the running time of the fastest algorithm you can think of to compute A^k?

    An acceptable solution:

    (a) The (i, j)'th entry of A^2 contains the number of paths consisting of two edges (possibly with repeating nodes, if you take a self-loop) that start at i and end at j.
    (b) The number of paths consisting of k edges that start at i and end at j.
    (c) You can prove it by induction on k. It is true for $A^1 = A$. Then, $A^k = A^{k-1} \cdot A$. Call the entries of A^{k-1} as $a_{i\ell}^{k-1}$, which by the induction hypothesis are the number of paths of length $k-1$ that start at i and end at ℓ. The dot product $A^{k-1} \cdot A$ produces entries of A^k with

$$a_{ij}^k = \sum_{\ell \in \{0 \ldots n\}} a_{i\ell}^{k-1} \cdot a_{\ell j}.$$

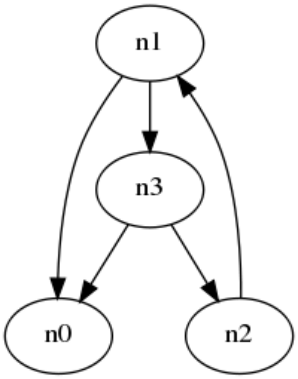


Figure 1: A graph used for a few problems in the homework

Consider the terms in this sum, $a_{i\ell}^{k-1} \cdot a_{\ell j}$. This term multiplies the number of paths that start at i, end at ℓ, and have length of $k-1$, by either 0 if there is no edge from ℓ to j or 1 if there is. Thus, this term counts the number of length k paths that start at i, end at j and have ℓ as the next-to-last node. The sum over all ℓ gives the number of length k paths that start at i and end at j.

(d) The maximum value of a single entry of A^k is n^{k-1}, this comes from counting the number of possible intermediate nodes between i and j in a length k path.

(e) Assuming that additions and multiplications take $O(1)$ time, you can compute this in time $O(n^3 \log k)$ through naive matrix multiplication. You can actually multiply matrices faster than $O(n^3)$ using algorithms like Strassen's algorithm (try looking it up, if you are curious). The currently fastest known algorithm to multiply matrices is about $O(n^{2.372})$.
