CS 381 Introduction to the Analysis of Algorithms
Fall 2021 Simina Branzei and Alexandros Psomas HW 5

Due Fri November 12 at 11:59PM

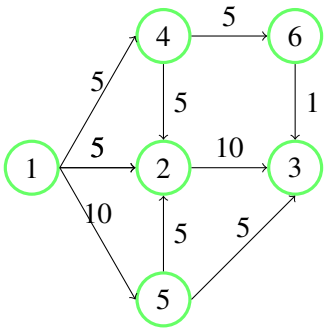
1. 10 points Prove using induction that the recurrence $T(n) = T(n-1) - T(n-2) + n^2 - 2n + 3$ with $T(1) = 1$ and $T(2) = 4$ exactly equals n^2 for $n \geq 3$.

   Answer: Written by Michael Beshear and checked by Nicholas Recker.
   Base case1: $n = 3$
   $T(3) = T(2) - T(1) + 3^2 - 2(3) + 3$
   $T(3) = 4 - 1 + 6 = 9 = 3^2$
   Base case2: $n = 4$
   $T(4) = T(3) - T(2) + 4^2 - 2(4) + 3$
   $T(4) = 9 - 4 + 11 = 16 = 4^2$
   Inductive Hypothesis: assume that $T(k) = k^2$ and $T(k-1) = (k-1)^2$ (stronger induction will also work; i.e. $T(n) = n^2$ for $3 \leq n \leq k$
   Inductive Step: Prove $T(k+1) = (k+1)^2$
   $T(k+1) = T(k) - T(k-1) + (k+1)^2 - 2(k+1) + 3$
   $T(k+1) = k^2 - (k^2 - 2k + 2) + (k^2 + 2k + 2) - 2k - 2 + 3$
   $T(k+1) = (1 - 1 + 1)k^2 + (2 + 2 - 2)k + (-2 + 2 - 2 + 3)$
   $T(k+1) = k^2 + 2k + 1 = (k+1)^2$

2. 15 + 4 + 3 + 3 points Answer the following questions for the graph given below:

   - Run the Ford-Fulkerson algorithm on the graph. You need to present the residual graph after each step i.e., after you send the flow along a specific path. To keep solutions consistent, assume Ford-Fulkerson uses DFS prioritizing lower numbers to choose its augmenting paths. (e.g. if the only valid paths are 1463 and 153, it will choose 1463)
   - State the max flow for the given graph.
   - If the edge from vertex 4 to vertex 6 has its weight changed from 5 to 10, what would the maximum flow be?
   - If the edge from vertex 6 to vertex 3 has its weight changed from 1 to 2, what would the maximum flow be?

   The vertex 1 is the source, while the vertex 3 is the sink.



Answer: Written by Nitish Kumar, checked by Michael Beshear

- First remove 1->2->3 with flow 5. Then remove 1->4->2->3 with flow 5. Then remove 1->5->2->4->6->3 using augmenting paths for flow 1. Then remove 1->5->3 with flow 5. The total flow is 5+5+1+5 = 16.
- The maximum flow is 16.
- The maximum flow remains the same i.e., 16.
- The maximum flow right now is 17.
- The maximum flow changes only for sub question 4, and this is because is the min-cut is changed only in sub question 4 (another way to understand this is, the edges which form the real constraint on the flow have their weight increased in sub question 4).

3. 25 points

In a certain hospital, there are $2n$ doctors and in the next month there are $3n$ surgeries planned. Each doctor i has reported an array S_i of the surgeries she is qualified to perform. For example, if doctor 1 can perform surgeries 1, 3, and 4, but not any others, then $S_1 = [1,3,4]$. Each surgery requires 2 qualified doctors. Each doctor only has time to participate in at most 3 surgeries.

Find and describe a max-flow-based algorithm to find an assignment of doctors to surgeries. If not every surgery can be fully staffed with 2 qualified doctors, fill as many slots as possible (i.e. two surgeries with one qualified doctor each is exactly as good as one fully staffed surgery and one completely unstaffed surgery). Then analyze and prove your algorithm's runtime and correctness.

You may use max flow as a subroutine without explaining how it works (though you do need explain what the input graph is, how to interpret the output, etc.). If every edge capacity is an integer, you may assume max flow uses an integer amount of each edge. Assume that max flow requires $O(|E|C)$ time, where $|E|$ is the number of edges in the input graph and C is the value of the max flow.

Here's an example:
$n = 2$
$S_1 = [1,2,3]$
$S_2 = [2,5]$
$S_3 = [1,2,4]$
$S_4 = [1,2,3,4,5,6]$

One correct answer would be as follows:

Surgery	Doctors assigned to it
1	1,3
2	1,2
3	1,4
4	3,4
5	2,4
6	

with total 10 slots filled, since 11 is not possible. Another correct answer would be

Surgery	Doctors assigned to it
1	1,3
2	1,2
3	1
4	3,4
5	2,4
6	4

since it also has 10 slots filled.

Answer: Written by Nicholas Recker, checked by Nithish Kumar.

We set up a graph G as follows. For every doctor i, we create a vertex x_i. For every surgery j, we create a vertex y_j. We also create a source vertex s and a sink vertex t. From s to every doctor vertex x_i we have an edge of capacity 3. From every surgery vertex y_j we have an edge of capacity 2. For every entry j in each qualification list S_i we have an edge from x_i to y_j of capacity 1.

To get our answer A, we run max flow on G, then check which $x-y$ edges were used. If (x_i, y_j) is used, we assign doctor i to surgery j.

By assumption, each edge is used an integer amount, so no doctor is "half assigned" to a surgery. No doctor is assigned to more than 3 surgeries lest the $s-x$ edge overflow its capacity. No surgery is assigned more than 2 doctors lest the $y-t$ edge overflow its capacity. No doctor is assigned to the same surgery multiple times lest the $x-y$ edge overflow its capacity. No doctor is assigned to a surgery they're unqualified for since there wasn't any edge there. Therefore A is a valid assignment.

To show the assignment A is optimal, suppose for sake of contradiction that there is a better assignment A'. Then if we route flow along the $x-y$ edges corresponding to the assignment A' we will get a better flow, which is a contradiction with our flow being maximal. (s-x and y-t edges will be able to satisfy that flow since A' is a valid assignment).

This graph has at most $6n^2+4n$ edges and the max flow has value at most $6n$, so the cost of running max flow is $O(n^3)$. All other steps take much less than this; for example, setting up the graph takes $O(n^2)$ times and reading off the solution takes $O(n^2)$ time. Thus the total runtime is $O(n^3)$. While not necessary, you can define a new variable $m = \sum_i |S_i|$; then the runtime becomes nm.

4. 25 points

   There is a large convention which wants to coordinate assignments of its $100n$ attendees to n affiliated hotels. Each attendee wants to stay at one hotel. Each hotel can accommodate 100 attendees. Each attendee i reports her preference list a_i; i.e. she prefers $a_i[1]$ over $a_i[2]$ over $a_i[3]$ ... over $a_i[n]$. Each hotel j reports its preference list h_j in a similar way.

   Describe a modification of the Gale-Shapely algorithm to find a stable assignment of attendees to hotels; i.e.

   - Every attendee is assigned to one hotel
   - Every hotel has 100 attendees assigned to it
   - There does not exist an attendee i and hotel j such that
     - i is not assigned to j
     - i prefers j over the hotel she was assigned to
     - j prefers i over its least preferred assigned attendee

   Then analyze and prove your algorithm's runtime and correctness.

   Answer: Written by Nicholas Recker, checked by Nithish Kumar.

   The hotels will be "proposing" to the attendees in this solution; there is an equivalently good solution where the attendees propose to the hotels. First we preprocess the attendee preference lists into an array b with a new format; $b_i[j]$ is the "rank" of hotel j; i.e. if attendee i's favorite hotel is hotel 5, then $b_i[5] = 1$. We do this preprocessing by iterating over the preference lists, at each step filling in one entry of a b_i.

   Each attendee maintains her "best so far" hotel, initialized to be n+1, worse than all hotels. Each hotel proposes to its top 100 attendees. Whenever an attendee with a best-so-far hotel is proposed to, she rejects whichever she less prefers, and the hotel proposes to its next most preferred attendee that it hasn't proposed to. When all attendees have a best-so-far hotel, the algorithm terminated with each attendee assigned to her best-so-far hotel.

   The preprocessing step can be done in $O(100n^2)$ time since we iterate over $100n$ lists of size n each doing $O(1)$ work for each entry. Each of n hotels proposes to at most $100n$ attendees, so there are at most $100n^2$ proposals. Each proposal takes $O(1)$ time to process. Therefore the whole algorithm takes $O(100n^2) = O(n^2)$ time.

   The algorithm must terminate, since each hotel can only propose to at most all $100n$ attendees. When it returns, every attendee has a hotel. Since every hotel reproposes whenever someone rejects them, on return every hotel has 100 attendees. Suppose for sake of contradiction that some attendee i and hotel j are an unstable pair. Then, since they aren't matched, either j never proposed to i or i rejected j. In the former case, j proposed to the 100 attendees that it did get assigned before it would have proposed to i, which is a contradiction with j preferring i to its least preferred assigned attendee. In the latter case, i rejected j, and so immediately afterward had a better hotel than j. However, by the end of the algorithm i had a worse hotel. i only ever trades up, so this is a contradiction. Either case is a contradiction, so no such (i, j) pair exists.

5. 5 + 5 + 5 points

   Consider the Reverse-Delete algorithm for finding an MST:

   (a) Sort all edges by weight

   (b) For each edge e in descending weight order:

   (c) If removing e would not disconnect the graph, delete e

   (d) After considering all edges, return the remaining graph

Prove the following statements relating to the Reverse-Delete algorithm for finding the minimum spanning tree. You may assume the graph is undirected, has distinct edge weights, and is connected. You may use the property that (in graphs with distinct edge weights) no MST contains the heaviest edge in a cycle. You may use each statement as part of your proofs for the statements after it.

- If an edge belongs to a cycle, then deleting it will not disconnect the graph
- The Reverse-Delete algorithm will delete the heaviest edge from every cycle
- Reverse-Delete always returns an MST

Answer: Written by Michael Beshear and checked by Nicholas Recker.

- For the sake of contradiction, assume that removing the edge $e = (s,t)$ disconnects vertices u and v, where e is in a cycle C. u and v were previously connected, so there was previously a path P from u to v. If $e \notin P$, then P still connects u to v. If alternatively $e \in P$, then we can replace $C \cap P$ with $C \setminus P$; the result is another path from u to v that doesn't include e. So either way u and v are still connected, which is a contradiction with our assumption.
- For every cycle, there is a heaviest edge. Since it is the heaviest in the cycle, the whole cycle is still in the graph when it is considered. So when it is considered, it is still part of a cycle, and thus by statement one, its deletion will not disconnect the graph. Therefore Reverse-Delete will delete it. Therefore Reverse-Delete will delete the heaviest edge from every cycle.
- Eventually, by statement two, all cycles will be removed. Also, by definition no deletion disconnects the graph. So returned graph will be a tree. Since every removed edge is the heaviest edge in a cycle, Reverse Delete never removes an MST edge. Because Reverse-Delete always results in a tree and never removes an edge in the MST, the tree that remains after Reverse-Delete must be the MST.