15-451/651 Algorithm Design & Analysis
Fall 2022, Recitation #7

Objectives

- To review the max flow algorithms we have learned and understand their differences.
- To practice analyzing properties of max flow algorithms.
- To see how we can use minimum-cost flows to model real-world problems.

Recitation Problems

1. **(Warm up)** We've learned a long list of algorithms for max flow at this point. Let's quickly review them and make sure we understand the differences. First, briefly describe the strategy that each algorithm uses:

   - **Ford-Fulkerson:**

   - **Edmonds-Karp (SAP):**

   - **Dinic's:**

   Now lets fill in the following table that describes the runtime of each of them.

Algorithm	Time per augmenting path	# augmenting paths	Total Time
Ford-Fulkerson			
Edmonds-Karp			
Dinic's			

2. **(Super fast matching)** In lecture, we saw that Dinic's algorithm runs in time $O(n^2m)$ on any graph, but on some graphs it runs even faster! For instance, in a *unit-capacity* network, one where every edge has capacity one, we proved that Dinic's algorithm runs in time $O(m\sqrt{m})$. In this problem we will take this one step further.

   Suppose our graph has one additional restriction (we still keep the unit-capacity restriction): The net flow across every vertex (except s and t) can be at most one. This is equivalent to saying that every vertex other than s and t has either indegree one or outdegree one (but not necessarily both). Such a network is called a "unit network".

   (a) Prove that in a unit network, the number of blocking flows required to find a max flow is at most $O(\sqrt{n})$. (Hint: use a similar argument to the one in lecture for unit-capacity graphs)

   (b) Prove that we can solve the Bipartite Matching problem in $O(m\sqrt{n})$ time.

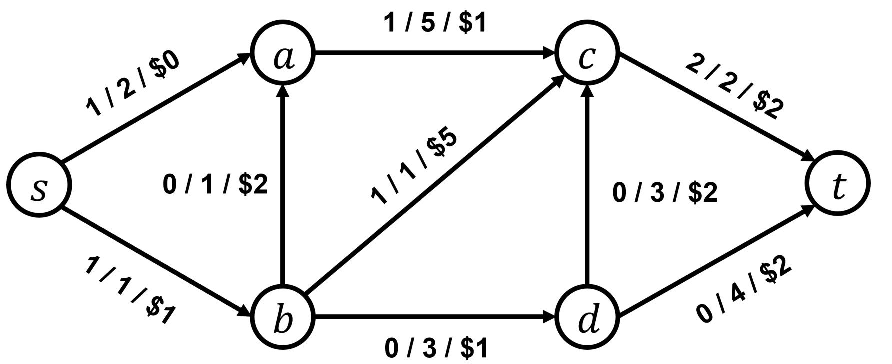
3. **(Oral scheduling)** You've been hired to help the 451 TAs schedule their oral sessions. There are n TAs, and TA i has s_i slots that they need to book a room for. There are m available room bookings, and each TA i has a list L_i of which room bookings $\{1, 2, \ldots, m\}$ would be suitable for them. Since there is a shortage of room bookings, however, the department has started to sell the bookings for money! The j^{th} booking costs c_j dollars. Your job is to find a way to schedule all of the oral sessions for the minimum amount of money, or report that it is not possible.

Further Review

1. **(Short answer / multiple choice)**

   (a) Draw a flow network for which the Edmonds-Karp algorithm is guaranteed to find the maximum flow in just one iteration, but the Ford-Fulkerson algorithm could take an arbitrarily high number of iterations depending on the capacities.

   (b) Consider the following flow network. Edges are labeled with $f/c/\$$, representing the current flow, the capacity, and the cost, respectively.



   i. Draw the residual graph of the flow network with respect to the current flow.

   ii. Is the flow a maximum flow? Justify your answer using the residual graph.

   iii. Is the flow a cost-optimal flow? Justify your answer using the residual graph.

2. **(More bounds for unit-capacity networks)** Recall that a *unit-capacity* network is one where every edge has capacity one. In lecture we proved that we can find a max flow in such a network in $O(\sqrt{m})$ blocking flow computations. Suppose we are given a *simple* unit-capacity network (where simple means there are no duplicate edges between the same pair of vertices). We will show that such a network requires at most $n^{\frac{2}{3}}$ blocking flow computations.

   (a) Consider the network after we have found k blocking flows. Prove that in the layered graph, at most half of the layers can have at least $\frac{2n}{k}$ vertices.

   (b) Using Part (a), prove that there exists a cut in the residual network of capacity at most $O\left(\left(\frac{n}{k}\right)^2\right)$

   (c) Using Part (b), prove that the number of blocking flows required is at most

$$O\left(k + \left(\frac{n}{k}\right)^2\right),$$

   for any k, then pick k to show that at most $n^{\frac{2}{3}}$ blocking flows are required.

   (d) Deduce that on a simple unit-capacity network, Dinic's algorithm runs in

$$O\left(m \min\left(\sqrt{m}, n^{\frac{2}{3}}\right)\right)$$

3. **(Kuhn's algorithm)** A popular algorithm in programming competitions for solving the Bipartite Matching problem is "*Kuhn's algorithm*". For each vertex $u \in L$, the algorithm tries to find a matching vertex $v \in R$ by either:

   (a) Finding an unmatched adjacent vertex v and matching it, or

   (b) Finding an already matched vertex v on the right, then recursively finding a *new match* for the vertex currently matched to v, then matching u to v.

   Here is some psuedocode. n is the number of vertices in L, and m is the number of vertices in R.

```
used := array⟨bool⟩(n, false)
match := array⟨int⟩(m, -1)     // match[v] = id of the vertex matched to v
matching_size = 0              // size of the matching found

dfs : (u : int) -> bool = {
  if (used[u]) return false
  used[u] = true
  for each vertex v adjacent to u do {
    if (match[v] == -1 || dfs(match[v])) {
      match[v] = u
      return true
    }
  }
  return false
}

for v = 0 to n - 1 do {
  used = array⟨bool⟩(n, false) // reset used to all false
  if (dfs(v)) matching_size++
}
```

   Explain how this algorithm is actually the same as the Ford-Fulkerson-based maximum flow algorithm for the Bipartite Matching problem that we learned in class. (Hint: what is the *dfs* function doing?)

4. **(Another polynomial-time algorithm for max flow)** In lecture, we saw the Edmonds-Karp Shortest Augmenting Paths algorithm, which we proved had a polynomial running time. Here's another natural strategy for picking augmenting paths: pick the augmenting path with the *largest capacity* (remember that the capacity of an augmenting path is the lowest capacity edge on the path, so we are trying to maximize the minimum capacity edge).

   (a) There are several ways to find such a path. Describe an algorithm that runs in $O(m \log n)$ time.

   (b) Prove the following lemma: In a graph with maximum s-t flow F, there exists a s-t path with capacity at least F/m. Hint: What happens if you delete all edges with capacity less than F/m?

   (c) Prove that this strategy requires at most $O(m \log F)$ augmenting paths to find a maximum flow, assuming the capacities are integers. Hint: Part (b) shows that we remove a fraction of $1/m$ of the remaining flow at each iteration. You might then want to use the inequality $(1 - 1/m)^m < 1/e$.

(d) Deduce that this algorithm finds a maximum flow in $O(m^2 \log n \log F)$ time. Explain why this is a polynomial-time algorithm for the usual definition.

5. **(When you forget Dijkstra)** You have a directed graph G with non-negative edge weights and you want to compute the length of the shortest path from some node u to some node v. Unfortunately, you've forgotten Dijkstra's algorithm and every other shortest path algorithm you learned in 15-210. However, you have some code written up that solves the minimum-cost max flow problem. Describe how to solve your shortest path problem using minimum-cost max flow.

6. **(Transshipment)** In the transshipment problem, you are given a directed graph, where each vertex has a *balance* b, which may be positive or negative. You can think of a vertex with a positive balance (called a *supply* vertex) as having an excess of some commodity, and a vertex with a negative balance (called a *demand* vertex) needing more of that commodity delivered to it. Edges in the graph have unlimited capacity, and a cost $\$(e)$ per unit of commodity send along it. The goal is to ship commodity around the network such that each vertex ends up with a zero balance, at the minimum possible cost. Describe how to solve this problem using minimum-cost maximum flow.

7. **(Finding a spanning tree)** You are given n points $p_1, p_2, \ldots, p_n$ in the 2-D plane. Assume that there is a unique point p_r that has maximum y coordinate. For each point p other than p_r, we're going to select another point q as its "parent", with the condition that the parent of p must have strictly larger y coordinate than p. These parent pointers naturally form a spanning tree, and the cost of the spanning tree is the sum of the Euclidean distances of all the edges in it.

   Suppose that each point can have at most d children, for some d given in the input. Give an algorithm that computes the minimum cost of a valid tree, or determines that no valid tree exists.

8. **(Make the matrix better)** You are given an $n \times n$ matrix M of integers. You want to re-order the columns of the matrix in such a way that the sum of the elements of the diagonal is as large as possible. Describe an algorithm that determines the largest possible diagonal sum.

9. **(Generalized Generalized Matching)** Generate the cheapest $m \times n$ matrix that meets the following specifications

   - Each entry is either 0 or 1
   - Row i sums to a given constant r_i
   - Column j sums to a given constant c_j
   - The cost of a matrix A is $\sum_{(i,j)} d_{ij} \cdot A[i][j]$ for given constants d_{ij}