Lecture 16

First, a quick review of what we learned about max flow last time:

1. Edmonds-Karp Algorithm - Keep pushing flow along augmenting paths in the residual graph. Find paths using breadth-first search. If an adjacency matrix is feasible, the implementation is easier. For an adjacency list, use edge objects and have the edge object for (v, w) point at the edge object for (w, v), and vice-versa. Runtime: $O(VE^2)$.

2. There are faster algorithms: Dinitz $O(V^2E)$, FIFO Push-relabel $O(V^3)$.

3. Integral Flow Theorem: If all the edges have integral capacities, then there is a max flow where every edge has integral flow (each augmenting path has integral capacity).

4. Max-Flow Min-Cut theorem: The maximum flow from s to t is the size of the minimum $s-t$ cut. An $s-t$ cut is a partition of the nodes into 2 sets S, T with $s \in S$ and $t \in T$. The value of the cut is the sum of all capacities on edges from S to T.

Let's elaborate on max-flows and min-cuts. Consider any partition S and T of V where $s \in S$ and $t \in T$. Then the max-flow from s to t must be at most the sum of all capacities from S to T. How do we find the min-cut? The key is to determine all of the nodes in S. These are precisely the nodes reachable from s in the residual graph. Any edge leaving S is part of the cut.

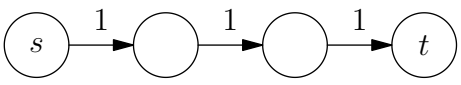
If each edge has a cost and a capacity, you may want to find the min-cost max-flow. To do this simply replace breadth-first search (in Edmonds-Karp) with Bellman Ford using the edge weights. Runtime: $O(V^2E^2)$. Can be improved to $O(V^3E)$ using a mixture of Bellman Ford and Dijkstra.

Review

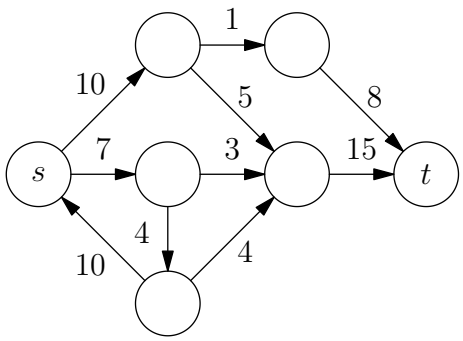
1. What is the runtime of Dijkstra's algorithm if you use an (unsorted) array instead of a heap as your priority queue?

2. Compute a min $s - t$ cut for each of the following graphs:

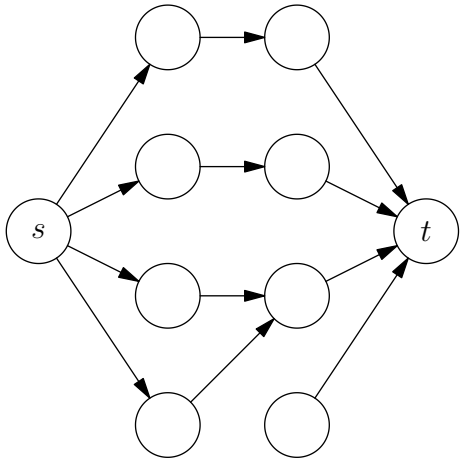
   (a)



   (b)



(c) All capacities 1:



3. Suppose you have a problem where vertices have capacities in addition to edges. Explain how to compute the maximum flow.

4. ($\star$) You are given an $n \times n$ chess board and a list of bad squares. Compute the maximum number of non-attacking rooks you can simultaneously place on the board assuming a rook cannot be placed on a bad square, but they can move across them.

5. Suppose you are given a graph with edge capacities, and a list of sources and sinks. Each source s_i has an amount of flow $f(s_i)$ that must come out of it. Each sink k_j has an amount of flow $f(k_j)$ that must flow into it. Determine whether these constraints can be satisfied by pushing flow through the network. That is, your answer is boolean.

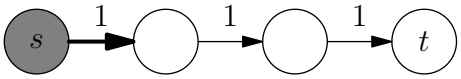
6. ($\star$) In addition to a list of sources and sinks, you are given a graph with edge capacities that have lower and upper bounds. That is, for each directed edge e you are given a lower bound $L(e)$ and an upper bound $U(e)$ so that the flow f along e must satisfy $L(e) \leq f \leq U(e)$. Determine if there exists a flow that satisfies these conditions. That is, your answer is boolean.

Solution

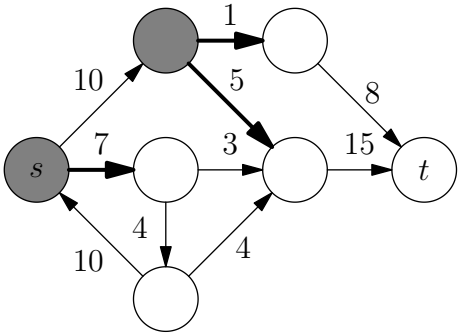
1. $O(V^2)$.

2. In the following we have darkened S and the cut.

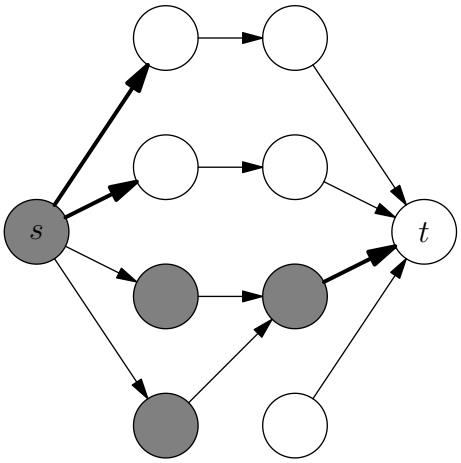
   (a) The min cut has weight 1.



   (b) The min-cut has weight 13.



   (c) The min-cut has weight 3. All capacities 1:



3. Replace each vertex v with 2 (v_1 and v_2) and a directed edge (v_1, v_2) between them with the required capacity. All incoming edges meet v_1 and all outgoing edges leave v_2.

4. Do a bipartite matching of rows to columns omitting the (row,column) pairs that correspond to bad squares.

5. The terminology is that we are looking for a *circulation*. Make a super-source that connects to each source with the required capacity. Make a super-sink that every sink connects to with the required capacity. The constraints can be met if every edge leaving the super-source and every edge entering the super-sink is saturated.

6. For every edge with a non-zero lower bound, assume we push flow through it equal to the lower bound. After we do this, each remaining vertex either has a flow surplus or deficit. Create sources and sinks at these nodes accordingly (surplus of k becomes a source of k flow, deficit of k becomes a sink of k flow).

Max Flow Applications

1. You have a list of customers c_i and a list of products p_i along with a list of which products each customer has bought. Determine if there is a way to get at least 10 product reviews for each product assuming each customer will evaluate at most 3 products he/she has purchased.

2. ($\star$) You are given a DAG of potential jobs for your company to complete, where a job can only be completed once its predecessors in the DAG have been. You also have an integer value v (positive or negative) associated with each job. Determine the subset of jobs that yields the most total value.

3. ($\star$) You are given a list of how many wins w_i each baseball team has, where $1 \leq i \leq n$. You are also given how many games g_{ij} teams i and j have left to play against each other (each game is won or lost). Determine if a given team k has a chance at being first or tied for first (a team is first if it has the most wins).

4. ($\star$) Your statistical model has determined for each pixel the likelihood it should be labeled part of the foreground or background of the image. The pixel p_i has a cost a_i if it is part of the foreground, or a cost b_i if it is part of the background (costs all positive). You have also determined for each pair (i, j) a cost p_{ij} they are placed differently. Give an algorithm to determine an assignment of pixels to the background and foreground that minimizes the cost.

Solutions

1. Let P be the number of products. Make a bipartite graph pairing customers and the products they bought. Make a sink that all of the products connect to that demands $10P$ flow. Each product connects to it with a lower bound of 10 capacity. Make a source that connects to all customers with a capacity of 3. The source must output $10P$ flow.

2. Build the dag where each node points to the nodes it depends on with infinite (or large enough to never be a bottleneck) capacity. Create a source s and a sink t. If a task has positive value, then connect the source to it with that value. If a task has negative value, then connect it to the sink with its absolute value. As each vertex becomes paired with either s or t, the value of the min cut exactly determines the optimal project selection. To get the optimal value, compute C, the sum of all positive project

values, and subtract the value of the cut. Note, we could have used $C + 1$ instead of infinity as it is guaranteed not to be a bottleneck.

3. Assume team k wins the rest of its games getting W wins. If W is less than any w_i for $i \neq k$ we are immediately done (no chance for k). Otherwise, we create a graph where flows denote wins. Create a node for each team t_i and each pair p_{ij}. Have a source s with capacity g_{ij} to each p_{ij}. Each p_{ij} has edges to t_i and t_j, both with capacity g_{ij}. Create a sink t and have an edge from t_i to t with capacity $W - w_i$. Team k has a chance if there is a flow that saturates all edges leaving s.

4. We make a node for every pixel. Between each node i and j, we have a bidirectional edge with capacity p_{ij}. For each node i, there is an edge from the source s to i with capacity b_i, and an edge from i to the sink t with capacity a_i. Pairing with the source means you are in the foreground, and the min cut is exactly what we want.

Dinitz's Algorithm

Based on Blocking Flows. Runtime: $O(EV^2)$ or $O(VE \log V)$ with a complex data structure. For bipartite matching, the $O(EV^2)$ algorithm will actually run in $O(E\sqrt{V})$ giving the Hopcroft-Karp algorithm.

To do Dinitz's algorithm, at each step we are still looking for augmenting paths. We first run BFS to mark each node i with a distance d_i to the source in the residual graph. We then push an $s - t$ flow down all possible paths where each edge increases your distance (i.e., if (i, j) is an edge, then we must have $d_j > d_i$). This is called creating a blocking flow. Using a DFS, we can perform this operation in $O(EV)$. It can be proven that we will repeat this BFS process at most $O(V)$ times. Hence the runtime is $O(EV^2)$.

Floating Point Arithmetic

Doubles are used to represent the extended real numbers on our computer using a floating point representation. The bit layout of a double is as follows.

1. Bit 63: Sign Bit
2. Bits 62-52: Exponent (represented as $e + 1023$)
3. Bits 0-51: Significand (or mantissa)

Each number takes the form

$$(-1)^{b_{63}} 1.b_{51}b_{50}b_{49} \ldots b_0 \times 2^e,$$

where e is as described above. Two of the exponents (the largest and smallest) are special, and are used to represent zero (which can be signed), infinities, and denormalized numbers.

Floating Point Exercises

1. What are 1.1011_2 and $1.1101_2 \times 2^3$?

2. Suppose you have the following code:

```
double d = 39.0, e = 13.0;
if (d/3.0 == e)
  System.out.println("Yes1");
if ((1/49.0)*49.0 == 1)
  System.out.println("Yes2");
```

   Will it print Yes1, Yes2, both, or neither?

3. Suppose you have the following code:

```
import static java.lang.Math.*;
System.out.printf("2^53 = %f\n",pow(2,53));
System.out.printf("2^53 - 1 = %f\n",pow(2,53)-1.0);
System.out.printf("2^53 + 1 = %f\n",pow(2,53)+1.0);
System.out.printf("2^53 + 2 = %f\n",pow(2,53)+2.0);
System.out.printf("Next 2^53 = %f\n",nextUp(Math.pow(2,53)));
```

   Give the output in terms of 2^{53}.

4. Suppose $x \in \mathbb{R}$ and let $\hat{x}$ be its floating point representation as a double.

   (a) If $\hat{x} = 5$ what is the biggest possible absolute error $|\hat{x} - x|$?

   (b) If $\hat{x} = 5$ what is the biggest possible relative error $\frac{|\hat{x}-x|}{x}$ (you can approximate as $\frac{|\hat{x}-x|}{\hat{x}}$)?

   (c) How do the above two types of error change as x gets larger or smaller?

5. What is the largest range of integers containing 0 that doubles can represent exactly?

6. What is the largest double? What is the smallest double?

7. How would you compare two doubles for equality? How about strictly greater-than, or greater-than-or-equal?

Solutions

1.
$$1.1011_2 = 1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = 1 + \frac{11}{16}$$
and
$$1.1101_2 \times 2^3 = 1110.1_2 = 14 + \frac{1}{2} = 14.5.$$

2. Only Yes1.

3. 2^{53}, $2^{53}-1$, 2^{53}, $2^{53}+2$, $2^{53}+2$.

4. (a) $2^{-53} \cdot 2^2 = 2^{-51}$.
   (b) Approximately $\frac{2^{-51}}{5}$.
   (c) The relative error roughly remains the same (stays within an interval around 2^{-53}). The absolute error grows as $|x|$ grows, and decreases as $|x|$ decreases.

5. $[-2^{53}, 2^{53}]$.

6. $\pm\infty$. For finite values, it is roughly $\pm 2^{1024}$.

7. Often eps is something like $1e{-}9 = 10^{-9}$.

```
boolean approxEquals(double a, double b, double eps)
{
  return Math.abs(a-b) < eps;
}
boolean gt(double a, double b, double eps)
{
  return a > b + eps;
}
boolean geq(double a, double b, double eps)
{
  return a > b-eps;
}
```

Cycle Finding and Division Exercises

1. Suppose you are given a singly linked list that may have a loop. Give code using finitely many pointers and counters which determines if it has a loop, where the loop begins, and how long the loop is.

2. Let $f(x) = (x^2 + \lfloor 17\sin(x) \rfloor + 9)\%1000003$. Define a sequence by $a_0 = 1$ and $a_{n+1} = f(a_n)$. Give an algorithm to find a cycle in a_n and determine its length.

3. Write a function isOdd that determines if an int is odd.

4. Give a, b, c so that $(a\%b)\%c \neq a\%c$. What does this imply when $b = 2^{32}$?

5. What is $5\%3$? What is $-7\%2$? What is $7\%-2$?

6. Suppose you have an int a and a mod M. Determine how to get the smallest non-negative value b such that M divides $a-b$. That is, we want the smallest non-negative b with $a \equiv b \pmod{M}$.

7. Let $a = qb + r$ where $a, b, q, r \in \mathbb{Z}$ are non-negative and $0 \leq r \leq b$. Show that if g divides r and b then g divides a. What does this allow?

Solutions

1. Advance two pointers p, q, where p is twice as fast as q. If p ever crosses q, there is a loop. Once p crosses q, both are in the loop. Set $p = q$, freeze p, and advance q until it meets p again. This gives the length L of the loop. Finally, start p, q from the beginning. Advance p until it is L steps ahead of q. Then advance p, q at the same time until they meet. They will meet at the beginning of the loop. It is possible to find the beginning of the loop using no counters.

2. Use the above cycle finding algorithm.

3. **boolean** isOdd(**int** a){ **return** a%2!=0; } Note that a%2==1 is incorrect.

4. $(5\%3)\%2 \neq 5\%2$. That you can't allow overflow when you are returning values mod some M.

5. $5\%3 = 2$. Note that $-7/2 = -3$ so

$$-7 = (-7/2) \cdot 2 + r = -6 + r \implies r = -1.$$

   Furthermore

$$7 = (7/-2) \cdot -2 + r = 6 + r \implies r = 1.$$

   Hence $-7\%2 = -1$ and $7\%-2 = 1$.

6. You could do b = ((a % M)+ M)% M;, but the following avoids doing 2 mods:

```
int b = a % M;
if (b < 0) b += M;
```

7. If $r = gj$ and $b = gk$ then $a = qb + r = qgk + gj = g(qk + j)$. This allows us to use the Euclidean algorithm for finding GCDs:

```
int gcd(int a, int b) { return b == 0 ? a : gcd(b,a%b); }
```