Problem 1. True or False
Answer True or False to the following questions:

(a) If an iteration of the Ford-Fulkerson algorithm on a network places flow 1 through an edge \((u, v)\), then in every later iteration, the flow through \((u, v)\) is at least 1.

(b) If \textsc{VertexCover} reduces to some yes-no problem \(A\), \(A\) is NP-complete.

(c) If all of the edge capacities in a graph are an integer multiple of 7, then the value of the maximum flow will be a multiple of 7.

(d) Skip lists are more useful than balanced binary search trees in solving the predecessor problem because they can query in \(O(\log n)\) time.

Solution

(a) **False.** A later augmenting path may pass through \((v, u)\), causing the flow on \((u, v)\) to be decreased.

(b) **False.** \(A\) is NP-hard, but is not necessarily in NP.

(c) **True.** Each iteration of Ford-Fulkerson will increase the value of the flow by an integer multiple of 7.

(d) **False.** BBSTs also solve the predecessor in \(O(\log n)\) time, and in fact skip lists only achieve \(O(\log n)\) expected time. Skip lists provide a simpler randomized solution.

Problem 2.
Run Ford-Fulkerson on the following graph from \(s\) to \(t\), using the method of augmenting paths discussed in class. What is the final flow? Show the residual network at the final step of building the flow. It also may be helpful to show intermediate residual networks.
Solution
The dotted lines represent “backwards” or “used” flow. First we send 3 units of flow along \((s, a)\) and \((a, t)\). Then we send 1 unit of flow along \((s, b), (b, c),\) and \((c, t)\). This gives us a total flow of 4.

Problem 3. Balls into Bins (CLRS 5.4-6)
Suppose that \(n\) balls are tossed into \(n\) bins, where each toss is independent and the ball is equally likely to end up in any bin.

(a) What is the expected number of empty bins?

(b) What is the expected number of bins with exactly one ball?

Hint: Use indicator random variables.

Solution

(a) Let \(X\) be the expected number of empty bins. Then we can write \(X = \sum_{j=1}^{n} I_j\) where \(I_j\) is the indicator of the \(j\)th bin being empty. Then by linearity of expectation, and the expectation of an indicator being equivalent to the probability of the event:

\[
E[X] = E \left[ \sum_{j=1}^{n} I_j \right] = \sum_{j=1}^{n} E[I_j]
\]

\[
= \sum_{j=1}^{n} P(\text{bin } j \text{ is empty})
\]

\[
= n P(\text{bin 1 is empty}) \quad \text{(by symmetry)}
\]

\[
= n \times \left( \frac{n-1}{n} \right)^n,
\]

because for bin 1 to be empty, each ball must not land in it. The probability of a specific ball not landing in bin 1 is \((n-1)/n\), and all the ball tosses are independent.

(b) Similarly let \(Y\) be the expected number of bins with exactly one ball and \(I_j\) be the indicator
of bin $j$ having exactly one ball.

$$E[Y] = E\left[\sum_{j=1}^{n} I_j\right] = \sum_{j=1}^{n} E[I_j]$$

$$= \sum_{j=1}^{n} P(\text{bin } j \text{ has } 1 \text{ ball})$$

$$= n P(\text{bin } 1 \text{ has } 1 \text{ ball}) \quad \text{(by symmetry)}$$

$$= n \times n \times \frac{1}{n} \times \left(\frac{n-1}{n}\right)^{n-1} = n \times \left(\frac{n-1}{n}\right)^{n-1}$$

Note that the number of balls in a bin is distributed Binomial$(n, 1/n)$, so the probability that bin 1 has 1 ball follows.

**Problem 4. Shortest Path LP**

Suppose you are given a weighted, directed graph $G = (V, E)$, with weight function $w$ mapping edges to positive real-valued weights, a source vertex $s$, and a destination vertex $t$. Show how to compute the value $d[t]$, which is the weight of a shortest path from $s$ to $t$, by linear programming.

*Hint:* Have variables $x_{(u,v)}$ for each edge and think about constraints similar to that of flow.

**Solution**

Use a variable $x_{(u,v)}$ for every edge. $x_{(u,v)}$ is 1 when $(u, v)$ is in the shortest path from $s$ to $t$ and 0 otherwise:

$$\text{minimize } \sum_{(u,v)} w_{(u,v)} x_{(u,v)}$$

subject to

$$\sum_{u} x_{(s,u)} = 1$$

$$\sum_{u} x_{(u,t)} = 1$$

$$\sum_{u} x_{(u,v)} - \sum_{w} x_{(v,w)} = 0 \text{ for every vertex } v \notin \{s, t\}$$

$$x_{(u,v)} \geq 0 \text{ for every edge } (u, v)$$

Notice that this might give you a fractional solution. However, I claim that given any fractional solution, I can find an integral solution with the same value. Given a fractional solution $\bar{x}_{(u,v)}$, construct a graph where edge $(u, v)$ has weight $\bar{x}_{(u,v)}$. Notice that the max flow of this graph is 1 due to our LP constraints. Run a Ford-Fulkerson-like algorithm to decompose this flow into augmenting paths, where path $p_i$ is the set of edges in the path, and path $p_i$ has flow $f_i$. Note that $\sum_i f_i = 1$ because our max flow is 1.

Define $\tilde{x}_{i,(u,v)}$ to be $f_i$ if $(u, v) \in p_i$ and 0 otherwise. Now let $t' = \arg \min_i \frac{1}{f_i} \sum_{(u,v)} w_{(u,v)} \tilde{x}_{i,(u,v)} = \arg \min_i \sum_{(u,v) \in p_i} w_{(u,v)}$. This gives us the shortest path $p_{t'}$ of all the augmenting paths.
We can decompose the value of the objective function at $\bar{x}_{(u,v)}$ as:

$$\sum_{(u,v)} w_{(u,v)} \bar{x}_{(u,v)} = \sum_i \sum_{(u,v) \in p_i} \bar{x}_{i,(u,v)} w_{(u,v)}$$

$$= \sum_i f_i \sum_{(u,v) \in p_i} w_{(u,v)}$$

$$\geq \sum_i f_i \sum_{(u,v) \in p_i} w_{(u,v)} = \sum_{(u,v) \in p_i^\prime} w_{(u,v)}$$

Now we have an integer solution $x_{(u,v)} = 1$ for $(u,v) \in p_i^\prime$ with objective function less or equal to the fractional solution. Thus the objective function for the fractional and integral solutions are equal, so the value of the LP solution corresponds to the weight of the true shortest path.

**Alternative solution:** The hint didn’t suggest this formulation, but this is another way to solve the problem without worrying about integrality (but still won’t return the shortest path itself). Consider variables of the form $d_u$ representing the shortest distance from $s$ to the vertex $u$. Then we can formulate the following LP:

maximize $d_t$

subject to $d_s = 0$

$$d_v \leq d_u + w_{(u,v)}$$

for every edge $(u,v)$

The maximization as the objective function may seem weird, but intuitively the constraints are doing the minimizing rather than the objective function. If we minimized rather than maximized, we would get $d_v = 0$ for all $v$. This formulation is more analogous to our shortest-path algorithms than the flow-like formulation above.

**Problem 5. Building Hospitals**

Suppose we are given a set of cities represented by points in the plane, $P = \{p_1, \ldots, p_n\}$. We will choose $k$ of these cities in which to build a hospital. We want to minimize the maximum distance that you have to drive to get to a hospital from any of the cities $p_i$. That is, for any subset $S \subset P$, we define the cost of $S$ to be

$$\max_{1 \leq i \leq n} \text{dist}(p_i, S) \quad \text{where} \quad \text{dist}(p_i, S) = \min_{s \in S} \text{dist}(p_i, s).$$

This problem is known to be NP-hard, but we will find a 2-approximation. The basic idea: Start with one hospital in some arbitrary location. Calculate distances from all other cities — find the "bottleneck city," and add a hospital there. Now update distances and repeat.

(a) Give a precise description of this algorithm, and prove that it runs in time $O(nk)$.

(b) Prove that this is a 2-approximation.

*Hint:* Any two points within a circle of radius $r$ are at most $2r$ away from each other.

**Solution**

(a) For each city $p_i$ we let $d_i$ denote its distance from the set of hospitals so far. Suppose we have just added the $j$-th hospital $s_j$. To decide where to add the next hospital, we look at $O(n)$
cities, updating the distances \( d_i \) by taking \( d_i = \min(d_i, \text{dist}(p_i, s_j)) \). The bottleneck city is the one with maximal \( d_i \) after this update, so we set \( s_{j+1} \) to be this city. We continue until \( k \) hospitals have been added, and we do a last update of the distances to find the final cost of our choice of \( S \).

(b) Consider the optimal choice of \( S^* = \{s_1, \ldots, s_k\} \), and let \( D_j \) denote the disc with center \( s_j \) and radius \( r^* \), where \( r^* \) denotes the cost of \( S^* \). Now consider the \( S \) that we obtained from our approximation algorithm, which has cost \( r \). If every \( D_j \) contains some point of \( S \) then we are done, because then every \( p_i \) will be within distance \( 2r^* \) of \( S \). Suppose not: By the pigeonhole principle, two points \( s, t \in S \) must be contained in a single disc \( D = D_j \). But by the way we constructed \( S \), \( \text{dist}(s, t) \geq r \) because \( s \) and \( t \) both have hospitals. We also have \( \text{dist}(s, t) \leq 2r^* \), so again \( r \leq 2r^* \) which concludes the proof.

Problem 6. NP-Completeness
Show that the following 2 problems are NP-complete:

(a) **Subgraph isomorphism:** Consider graphs \( G = (V, E) \), \( H = (V_2, E_2) \). Does \( G \) contain a subgraph \((V_1, E_1)\) which is isomorphic to \( H \)?

(An isomorphism of graphs \((V_1, E_1)\) and \((V_2, E_2)\) is a 1-1 function \( f : V_1 \to V_2 \) such that the map \((u, v) \mapsto (f(u), f(v))\) is a 1-1 function \( E_1 \to E_2 \).)

(b) **4-Exact-cover:** Given a finite set \( X \) of size \( 4q \) and a collection \( C \) of 4-element subsets of \( X \), does \( C \) contain an exact cover (i.e. a partition) of \( X \)?

You may use without proof that 3-exact-cover is NP-complete.

Solution

(a) This is clearly in NP because our certificate is the subgraph and isomorphism \( f \): we can apply the isomorphism to the subgraph and check for equality of the subgraph and \( H \). To show this is NP-complete, we reduce from clique: Asking whether there is a CLIQUE of size \( \geq K \) is equivalent to asking whether there is a subgraph isomorphic to \( C_K \), the complete graph on \( K \) vertices.

Another possible reduction is from the HAMILTONIANPATH. Asking whether there is a Hamiltonian path in graph \( G = (V, E) \) is equivalent to asking the subgraph isomorphism question on the graphs \( G \) and the Hamiltonian cycle (simple cycle) on \( |V| \) vertices.

(b) 4-Exact-cover is in NP because we can use the partition as a certificate and check if it is an exact cover. To prove NP-hardness, we reduce 3-Exact-cover to 4-Exact-cover. Expand \( X \) to \( X' \) by adding the elements \( e_1, \ldots, e_q \). For each 3-element set \( S \in C \), define 4-element sets \( S(i) = S \cup \{e_i\} \) for \( 1 \leq i \leq q \). Solving 4-Exact-cover for \( X' \) with

\[
C' = \{S(i) : S \in C, 1 \leq i \leq q\}
\]

will give the result for 3-Exact-cover.

Problem 7. Two Player Game
Consider the following zero-sum game, where the entries denote the payoffs of the row player:
\[
\begin{pmatrix}
2 & -4 & 3 \\
1 & 3 & -3
\end{pmatrix}
\]

Write the column player’s minimization as an LP. Then, write the dual LP, which is the row player’s maximization problem.

**Solution**
Let \( p \) be the payoff of the row player. The column player wishes to minimize \( p \) with to a strategy of \((y_1, y_2, y_3)\) (probabilities of picking column 1, 2, 3 respectively). Recall that for any given strategy of one player, the best response of the other player will occur when the other player plays one of their pure strategies. That’s why in the constraints, we only consider the opponent playing pure strategies.

Thus, the column player’s LP is
\[
\begin{align*}
\min \ p \\
p &\geq 2y_1 - 4y_2 + 3y_3 \\
p &\geq y_1 + 3y_2 - 3y_2 \\
y_1 + y_2 + y_3 &= 1, \\
y_1, y_2, y_3 &\geq 0.
\end{align*}
\]

On the other hand, let the row player’s strategy be \((x_1, x_2)\). Then, the dual LP is
\[
\begin{align*}
\max \ p \\
p &\leq 2x_1 + x_2 \\
p &\leq -4x_1 + 3x_2 \\
p &\leq 3x_1 - 3x_2 \\
x_1 + x_2 &= 1 \\
x_1, x_2 &\geq 0.
\end{align*}
\]

**Problem 8. Edge Connectivity**
The edge connectivity of an undirected graph is the minimum number \(k\) of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Let \( V = \{v_0, v_1, \ldots, v_n\} \). For each \( i \), consider the flow network \( F_i \) where \( v_0 \) is the source, \( v_i \) is the sink, and both \((u, v)\) and \((v, u)\) have capacity 1 for each \((u, v) \in E\). Let \( f_i \) be the total flow of \( F_i \). Prove that the edge connectivity is \( \min_i f_i \).

**Solution**
We will show the inequality in both directions:

(a) \( \min_i f_i \geq \) edge connectivity: consider flow network \( F_j \) that has the minimal flow. Now consider the min-cut derived from that maximum flow. When the edges in that cut had capacity zero, the sink is not reachable from the source. Hence, erasing those edges would render a disconnected graph.
(b) edge connectivity $\geq \min_i f_i$: Suppose $k$ is the minimum amount of edges to render $G$ disconnected and consider the graph $G'$ which is $G$ without the $k$ edges that makes it disconnected. Let $v_j$ be a vertex that is not reachable from $v_0$ in $G'$. Then the flow network $f'_j$ defined analogous to $f_j$ for the graph $G'$ must have flow 0, otherwise there would be a path connecting source $v_0$ and sink $v_j$. However, notice that the flow of $f_j$ is at most $k +$ the flow of $f'_j$, so the flow of $f_j \leq k$. So: $\min_i f_i \leq f_j \leq k$.

Problem 9. MST Bound

Suppose that we are given a graph with the following guarantee: for any two disjoint sets $S, T$ of vertices, there exists some edge between a vertex in $S$ and one in $T$ with length no more than $\frac{1}{\max(|S|, |T|)}$. Find a bound on the size of the minimum spanning tree for this graph, in terms of the number of vertices. Hint: Think Prim’s algorithm. You may find the following approximation useful:

$$\sum_{i=1}^{n} \frac{1}{i} \approx \ln(n)$$

Solution

We create a tree similarly to that of Prim’s algorithm. In particular, start with sets of vertices of size 1 and $n - 1$; we know that there exists an edge of size no more than $1/(n - 1)$ from the single vertex to one of the other vertices. Add that edge. Now, we have sets of size 2 (connected) and $n - 2$ (disconnected), and can find an edge of size no more than $1/(n - 2)$ from one of the two vertices into a disconnected one. We can continue this process until we have one connected set of size $\lceil n/2 \rceil$ (the tree so far) and one entirely disconnected set of $\lfloor n/2 \rfloor$ vertices.

This is the point at which we switch from using the bound $1/(\#\text{disconnected vertices})$ for a bound on the edge length to $1/(\#\text{vertices in tree so far})$.

Now there is some disconnected vertex of distance at most $1/\lceil n/2 \rceil$ from our tree so far, then one of $1/\lfloor n/2 + 1 \rfloor$, etc. Finishing out this process gives a total edge length of

$$2 \sum_{i=\lceil n/2 \rceil}^{n-1} \frac{1}{i} \leq 2 \sum_{i=1}^{n-1} \frac{1}{i} - 2 \sum_{i=1}^{\lfloor (n-1)/2 \rfloor} \frac{1}{i} \approx 2\ln(n - 1) - 2\ln\left(\lceil n/2 \rceil - 1\right) \approx 2\ln 2$$

This does not depend on $n$!

Problem 10. $s - t$ Min Cut

Suppose we try to use Karger’s contraction algorithm not for global min-cut, but for $s - t$ min cut in an undirected, unweighted graph. We do the same algorithm, but we modify it so that if it ever tries to contract an edge from the $s$-superverTEX to the $t$-superverTEX, we don’t do it (i.e. we always choose a random edge to contract amongst the edges which don’t connect $s$ to $t$).

(a) Prove that there are graphs such that the probability that this modified Karger’s algorithm finds an $s - t$ min cut is exponentially small.

(b) Show that unlike with global min-cut, the number of $s - t$ min cuts is not $O(n^2)$ but can be much bigger.
Solution

(a) Consider a graph where $s$ and $t$ are each connected to the other “middle” $n - 2$ vertices, and these other vertices are also arranged in a cycle. Then there are only 2 $s - t$ min cuts: $\{s\}, V - \{s\}$ and $\{t\}, V - \{t\}$ of size $n - 2$. Every other cut has weight at least $n$, because for every middle vertex has one edge cut (to either $s$ or $t$, each cut must be at least weight $n - 2$), and if the entire cycle isn’t on the same side of the cut, then at least 2 edges will cross the cut.

Then to return a min $s - t$ cut, the algorithm must either (1) never cut an edge touching $s$ or (2) never cut an edge touching $t$. This happens with exponentially small probability. Let $C$ be the $\{s\}, V - \{s\}$ cut. Then analogously to in lecture:

$$P(C \text{ is output}) = P(C \text{ survives every recursive level})$$

$$= \prod_{i=0}^{n-2} P(C \text{ survives level } i|C \text{ survived levels } 0, 1, \ldots, i - 1)$$

$$= \prod_{i=0}^{n-2} \frac{2(n - 2) - i}{3(n - 2) - i}$$

$$< \left(\frac{2}{3}\right)^{n-2} = O \left(\left(\frac{2}{3}\right)^{n-2}\right)$$

Note there are originally $3(n - 2)$ edges, and $n - 2$ of them come out of $s$.

(b) Note the maximum number of $s - t$ min cuts is $2^{n-2}$, because every vertex which isn’t $s$ or $t$ needs to choose to either be on the $s$ side or the $t$ side. This upper bound is achievable: consider the graph where $s$ has an edge to every non-$t$ vertex and $t$ also has an edge to every non-$t$ vertex. Then the size of the min cut is $n - 2$, and every valid cut achieves this.

Problem 11. Scheduling Jobs

There are $n$ jobs that we would like to schedule on $m$ machines. Job $i$ requires processing time $p_i$ and each job must be exactly assigned to one machine. The completion time of a machine is then the sum of processing time of each job assigned to it and the load of the assignment is the maximum completion time of any machines in this assignment. We want to minimize load. Prove that there is a greedy algorithm that achieves completion time at most $(2 - \frac{1}{m})$ times the optimal completion time.

Solution
Let $p_{\text{max}}$ be the maximum processing time of any job. Then $OPT \geq \max\{p_{\text{max}}, \frac{1}{m} \sum_{i=1}^{n} p_i\}$ because $OPT$ at least needs to cost $p_{\text{max}}$ time and if each machine takes less than average time then, you can’t finish all the jobs. Now consider a greedy algorithm that loops over 1 to $n$ and assign job $i$ to the machine that has the least load so far. In the worst case scenario, $p_{\text{max}} = p_n$, and $p_1, \ldots, p_{n-1}$ are perfectly distributed over the $m$ machines after $n - 1$ jobs are assigned. The load after $n - 1$ jobs is then $\frac{1}{m} (-p_{\text{max}} + \sum_{i=1}^{n} p_i)$, and then the load after adding the $n$th job the load will be:

$$\frac{1}{m} \left(-p_{\text{max}} + \sum_{i=1}^{n} p_i\right) + p_{\text{max}} = \frac{1}{m} \sum_{i=1}^{n} p_i + \frac{m - 1}{m} p_{\text{max}} \leq \left(1 + \frac{m - 1}{m}\right) OPT = \left(2 - \frac{1}{m}\right) OPT$$
Problem 12. Reduce the Flow (Berkeley CS 170)
Given a directed graph \( G = (V, E) \), where each edge has capacity 1. You are also given vertices \( s, t \in V \), and a number \( k \in \mathbb{N} \).

(a) Is it possible to compute the maximum \( s - t \) flow in \( O(|V||E|) \)? How or why not?

(b) Give an algorithm that returns \( k \) edges that, when deleted, reduce the maximum \( s - t \) flow by as much as possible. \textit{Hint}: the max flow is the same as what?

Solution

(a) Yes! We can just run Ford-Fulkerson. The runtime is \( O(|E|f^*) \) where \( f^* \) is the value of the max flow, but since the edge weights are all 1, we have \( f^* \leq |V| - 1 \), giving the desired runtime.

(b) First we run Ford-Fulkerson to compute the max flow \( f \) and construct the residual graph \( G^f \). We define \( S = \{ v \in V : v \) is reachable from \( s \) in \( G^f \} \) and then \( T = \{ (v, w) \in E : v \in S, w \notin S \} \). In other words, we find the set \( T \) of edges that cross the cut \( (S, V - S) \). Then we return any \( k \) edges in \( T \). Because of Ford-Fulkerson, \( (S, V - S) \) is a min cut equal in value to the max flow. By removing any \( k \) edges from \( T \), we reduce the capacity of the cut by \( k \) because each edge is unit weight, and the cut remains minimal in the modified graph because we’ve only made it smaller. Thus, the max flow also decreases by \( k \), and because we can only delete \( k \) edges, this is the most by which we can decrease the max flow so the algorithm is optimal.