Exercises

1. **Matrix Multiplication is Useful.** Given an undirected simple graph $G = (V, E)$, a triangle is just a clique of size 3; i.e., 3 vertices such that all 3 edges are present. Give algorithms for the following problems:
   - Find a triangle in $G$ in time $n^{\omega}$.
   - Find a $3k$-clique in a graph in time $n^{k\omega}$.
   - Find a triangle in $G$ in time $m^{1.5}$. (This one is slightly harder—not an exercise—and does not use matrix multiplication.)

2. **Low-Diameter Decompositions for Simple Graphs.** Recall a $\beta$-low-diameter decomposition, given graph $G$ and distance $D$, randomly breaks it into pieces of max-distance $D$, such that each pair $x, y$ is separated with probability at most $d_G(x, y) / D \cdot \beta$.
   
   (a) Show that if each edge $(x, y) \in E(G)$ is cut with probability $d_G(x, y) / D \cdot \beta$, then so is any pair $x, y \in V$. Hence, if the graph only has unit-weight edges, each edge can be cut with probability at most $\beta / D$.
   
   (b) Show that (i) any path graph has an LDD with $\beta = 1$, (ii) any tree with $\beta = 2$, and (iii) the standard $k$-dimensional $(n^{1/k} \times n^{1/k} \times \cdots \times n^{1/k})$-grid with $\beta = k$.

3. **Approximation via Randomized Simplification.** In Lecture #5 we saw low-stretch spanning trees, and used them to approximate TSP on general graphs. We explore this connection further, via the $k$-median problem we saw in HW#0: Given a graph $G$ and $k$, the $k$-median problem asks you to find a set $C \subseteq V$ with $|C| = k$ to minimize $\Phi_G(C) := \sum_{v \in V} d_G(v, C)$.
   
   (a) Given an algorithm to solve $k$-median optimally on trees, show that the algorithm that samples a tree $T$ from (random) low-stretch spanning tree distribution with stretch $\alpha$, solves $k$-median on $T$ to get $C_T$, and outputs this set $C_T$, ensures that the expected cost $E_T[\Phi_G(C_T)] \leq \alphaOPT$.\(^1\)
   
   (b) Show that if you perform $L := O(\log n)$ independent runs of the above algorithm to get sets $C_1, C_2, \ldots, C_L$, and return the set with the least $\Phi_G(C_i)$ value from among these (call it $C^*$), then $\Pr[\Phi_G(C_T) > (1 + \varepsilon)\alphaOPT] \leq 1 / \text{poly}(n)$.
   
   (c) Show that the expected weight of a low-stretch spanning tree is at most $O(\alpha)$ times the MST.
   
   (d) What kinds of problems can you solve using the ideas in the above parts. E.g., does it work for the TSP? How about the $K$-center problem? Or the $K$-means problem which wants to minimize $\Psi_d(C) := \sum_{v \in X} (d(v, C))^2$. Why or why not?

---

\(^1\)You saw this proved for a deterministic LSST, just make sure you see the proof for the randomized case.
(e) (Slightly non-trivial) Extend your dynamic-programming algorithm from HW#0 to solve $k$-median on an edge-weighted tree. Hint: first solve it on a binary tree. Then show how to reduce the problem to binary trees.

4. Flows, Kings, and Halls. Recall that König’s theorem says for a bipartite graph $G$, the size of the maximum matching in $G$ is equal to the size of the minimum vertex cover.

(a) Use the max-flow/min-cut theorem to prove König’s theorem. (Recall that the max-flow/min-cut theorem says that in any flow network, the maximum $s$-$t$ flow equals the minimum $s$-$t$ cut. Moreover if the arc capacities are integers, then the max-flow is guaranteed to be integral.)

(b) Use König’s theorem to prove Hall’s theorem:

In a bipartite graph $G = (L, R, E)$, for any set $S \subseteq L$, let $N(S) = \{r \in R \mid \exists \ell \in S, (\ell, r) \in E\}$ be the neighbors of $S$. Then $G$ has a matching of size $|L|$ if and only if $|N(S)| \geq |S|$ for all $S \subseteq L$.

(c) We never did get to the perfect matching polytope on general graphs in class. Here it is, given by the exponentially many inequalities:

$$K_{pm\text{-nonbip}} := \{x \in \mathbb{R}^m \mid \sum_{e \in \partial v} x_e = 1, \sum_{e \in \partial S} x_e \geq 1 \text{ for all odd sets } S, x \geq 0\}.$$  

As before $\partial S$ is the set of edges with one end the set $S$ and the other end outside. Show that the “odd set inequalities” $\sum_{e \in \partial S} x_e \geq 1$ for odd cardinality sets $S$ can be replaced by $\sum_{e \in E_S} x_e \leq |S|/2$, where $E_S$ is the set of edges both of whose endpoints lie in $S$.

5. Matching Reductions.

(a) Suppose you have an algorithm that solves max-weight perfect matchings (MaxWPM) for all non-negative weight functions. Give reductions that allow you to solve (a) min-weight perfect matchings (MinWPM) and (b) min-weight max-cardinality matchings (MinWMAXM)—i.e., among all matchings of size equal to $MM(G)$ find the one with least weight.

If the MinWPM and MinWMAXM instances are bipartite, ensure that the reductions give you MaxWPM instances that are bipartite too.

(b) Show how to solve the maximum cardinality matching problem for bipartite graphs using the algebraic approach from Lecture #9. Observe that you are not solving the weighted version here. (Hint: one way might use red-blue matchings.)

6. (An LP Equivalence) Consider this huge LP for maximum $s$-$t$-flow with edge capacities $c_e$: let $\mathcal{P}$ be the (potentially huge) set of paths from $s$ to $t$. This LP has variables $y_P$ for each path $P \in \mathcal{P}$, and goes thus:

$$\begin{align*}
\max \sum_{P \in \mathcal{P}} y_P & \\
\sum_{P \in \mathcal{P}, e \in P} y_P & \leq c_e \quad \forall e \in E \\
y_P & \geq 0.
\end{align*}$$

Show this LP is “equivalent” to the standard max-$s$-$t$-flow LP here. In particular, show that for any solution $f$ to the standard LP, there is a solution $y$ to this LP, and vice versa. (Hint: decompose any $s$-$t$-flow into flow-paths.)
7. (A Compact LP for Arborescences) In the notes for Lecture #2 we give an LP of exponential size for min-cost arborescences. Here’s a (sketch) of a polynomial-sized LP:

\[
\min w^\top x
\]

variables \( \{f_u\}_{e \in E} \) represent a flow of value 1 from node \( u \) to root \( r \). \( \forall u \neq r \quad (\star) \)

\[ f_e, x_e \geq 0. \]

Flesh (\( \star \)) out into \( O(n^2) \) LP constraints. (Hint: recall the LP for \( s-t \) flow, with flow-conservation at nodes.) Hence there are \( O(mn) \) variables and constraints in all.

Show that this LP is “equivalent” to the min-cost arborescence LP from lecture. I.e., as above, show that for any solution \( x \) to that LP, there is a solution \( (x, f) \) to this LP, and vice versa. You may assume non-negative weights \( w_e \). (Hint: max-flow min-cut.)

8. Self-Reduction

Suppose you have an algorithm \( A \) that takes a graph \( G \) and a number \( K \), and outputs \text{yes} if the graph has a vertex cover of size at most \( K \), and \text{no} otherwise. Give an algorithm for the search problem, i.e., one that takes \( HG, K \) and outputs a vertex cover of size at most \( K \) (if one exists), using at most \( n \) calls to \( A \).

Problems

Please solve any three of the following five problems.

1. Negative Triangles and Min-Sum Product. Recall from Lecture #4 that the All-Pairs-Shortest-Path problem can be solved using \( O(\log n) \) applications of Min-Sum Product. As a reminder, the Min-Sum Product is defined to take two \( n \times n \) matrices \( A, B \) and return a matrix \( T \) such that

\[
T_{ij} = \min_k (A_{ik} + B_{kj}).
\]

Now, let us consider another graph problem: Negative Triangles. The input consists of a weighted, undirected, tripartite graph with vertices \( A \cup B \cup C \) (each of size \( n \)), edge set \( E \) and integer edge weights \( w : E \to [-n, n] \). The output consists of all pairs of vertices \( a \in A, b \in B \) such that there exists \( c \in C \) such that \( (a, b), (b, c), (a, c) \in E \) and \( w(a, b) + w(a, c) + w(b, c) < 0 \), i.e., such that \( (a, b, c) \) forms a negative triangle.

We will use the Min-Sum Product to compute Negative Triangles, and vice-versa.

(a) Using a single application of Min-Sum Product, solve the Negative Triangles problem.

(b) Given an instance of the Min-Sum Product problem where all entries in the input matrices are integers in the interval \( [-n, n] \), solve the Min-Sum Product problem using \( O(\log n) \) applications of Negative Triangles.

\text{Hint: Using a single application of All Pairs Negative Triangles, how can you tell whether the entries of a candidate solution to Min-Sum Product are too high or too low?}

2. Spans, Ranks, and Duals. In HW1 you saw matroids; we’ll explore them a little more in this problem. Let \( \mathcal{M} = (U, \mathcal{I}) \) be a matroid. A few more definitions:

- Rank of a set. For \( S \subseteq U \), let \( \text{rank}(S) \) be the size of a largest independent set inside \( S \); i.e., \( \text{rank}(S) = \max \{|T| \mid T \subseteq S, T \in \mathcal{I} \} \).
• Span of a set. For $S \subseteq U$, let $\text{span}(S)$ be the largest $T \subseteq U$ such that $S \subseteq T$ and $\text{rank}(S) = \text{rank}(T)$. (We will see in part (d) that this is well-defined.)

We use $S + e$ to denote $S \cup \{e\}$, and $S - e$ to denote $S \setminus \{e\}$.

(a) (Do not submit) Show that $S \in I$ if and only if $\text{rank}(S) = |S|$.

(b) Show the following inequality for any $S \subseteq T \subseteq U$, and any $e \in U$:

$$\text{rank}(S + e) - \text{rank}(S) \geq \text{rank}(T + e) - \text{rank}(T).$$

(c) Using part (b) repeatedly, prove that for any $A, B \in U$,

$$\text{rank}(A) + \text{rank}(B) \geq \text{rank}(A \cup B) + \text{rank}(A \cap B).$$

This shows the rank function is "submodular".

(d) (Do not submit) Suppose $T_1$ and $T_2$ each satisfy $S \subseteq T_i \subseteq U$ and $\text{rank}(S) = \text{rank}(T_i)$. Show that $\text{rank}(S) = \text{rank}(T_1 \cup T_2)$. Deduce that $\text{span}(S)$ is well-defined.

(e) (Do not submit) For $S \subseteq U$, let $X(S) = \{ e \in U \mid \text{rank}(S + e) = \text{rank}(S) \}$. Prove that $X(S)$ is identical to $\text{span}(S)$.

(f) Suppose each element in $U$ has a distinct non-negative weight $w_e \in \mathbb{R}_{\geq 0}$. Consider the following LP for the problem of finding the max weight independent set:

$$\max \sum_e w_e x_e$$

s.t. $\sum_{e \in A} x_e \leq \text{rank}(A)$ for all $A \subseteq U$, $A \neq \emptyset$

$$x_e \geq 0$$

For any set $S \subseteq U$, let $\chi_S \in \{0,1\}^{|U|}$ be its characteristic vector. Show that $S \in I$ if and only if $\chi_S$ is feasible for this LP.

(g) Write down the dual of this LP.

(h) The greedy algorithm starts with the empty set, and each time picks the max-weight element $e$ that does not lie in the span of the current set of picked elements. Let $G$ be the set eventually output by the greedy algorithm, and let $w(G) := \sum_{e \in G} w_e$ be the weight of this set. Give a feasible dual solution $y^*$ such that the dual objective function equals $w(G)$ the primal objective function.

**Hint:** If greedy picks elements $g_1, g_2, \ldots, g_r$ in this order, just focus on the sets $R_i := \text{span}(\{g_1, g_2, \ldots, g_i\})$ for $i \in [r]$. Also, if you need something concrete to think about, think about the "graphical" matroid where $U$ is the set of edges of a graph and $I$ are the acyclic subsets.

3. Sparse Spanners. Given a graph $G$ with edge lengths $\ell_e$, a subgraph $H$ is a spanner with stretch $\gamma \geq 1$ if for every edge $(x, y) \in E(G)$,

$$d_H(x, y) \leq \gamma \cdot d_G(x, y).$$

(a) (Do not submit) Use the triangle inequality to show that for all $x, y \in V$, even if $(x, y)$ is not an edge, $d_H(x, y) \leq \gamma \cdot d_G(x, y)$. 

4
Clearly if $H = G$, we can set $γ = 1$. The goal is to find $H$ with few edges, such that $γ$ is also small. Let’s give two different constructions of good spanners.

(b) Approach #1. Sample $t = 4 \log n$ trees $T_1, T_2, \ldots, T_t$ from an $α$-stretch (randomized) low-stretch spanning tree. Let $H$ be the union of all these edges.

i. Show that for any fixed edge $(x, y) ∈ E(G)$,

$$\Pr[d_H(x, y) ≥ 2α d_G(x, y)] ≤ 2^{-t}.\)

(Hint: for any single value of $i$, bound $\Pr[d_{T_i}(x, y) ≥ 2α d_G(x, y)].)

ii. Use the Abraham-Neiman result quoted in Lecture to show that with probability $1 - \frac{1}{n^2}$, the graph $H$ is an $O(\log n \log \log n)$-stretch spanner with $O(n \log n)$ edges.

(c) Large-girth graphs are Sparse. On a seemingly different note, define the *girth* of graph $G$ to be smallest number of edges on any cycle in $G$. We will show that any graph $G$ with $m$ edges and $n$ nodes, and girth strictly more than $g$ must have $m \leq O(n + n^{1+1/\lceil g/2 \rceil})$.

i. The average degree of $G$ is $d := \frac{2m}{n}$. Show that there exists a subset $S ⊆ V$ such that the induced subgraph $H := G[S]$ has minimum degree at least $d/2$. [Hint: drop some low-degree vertices.]

ii. For this graph $H$ and any vertex $v ∈ H$, show that the number of distinct vertices reachable within $\lceil g/2 \rceil$ hops from $v$ is at least $(d/2 - 1)\lceil g/2 \rceil$.

iii. Prove: the number of edges $m$ in the original graph $G$ satisfies $m \leq O(n + n^{1+1/\lceil g/2 \rceil})$.

(d) Approach #2. It is a variant of Kruskal’s algorithm for $α ≥ 1$. Consider the edges of $G$ in increasing order of lengths $e_1, e_2, \ldots, e_m$. Initialize $H_0 = ∅$. When considering edge $e_i = (x, y) ∈ E(G)$, if the current distance $d_{H_{i-1}}(x, y) ≤ α d_G(x, y)$, then discard $e$ (i.e., set $H_i ← H_{i-1}$), else take it (i.e., set $H_i ← H_{i-1} \cup \{e_i\}$).

i. (Do not submit.) Show that if we set $α = n - 1$, then you will get Kruskal’s algorithm. Also, observe that by construction, the graph $H$ at the end of the process is an $(n - 1)$-stretch spanner. (In fact, an $(n - 1)$-stretch spanning tree.)

ii. If we set $α = O(\log n)$, use (c) with $g = O(\log n)$ to show the final graph $H$ is an $O(\log n)$-stretch spanner with $O(n)$ edges.

4. Min-weight Perfect Matchings. Suppose $G = (L, R, E)$ is a undirected bipartite graph with (possibly negative) integer edge weights $w_e$, suppose $M$ is some perfect matching. Let $H_M$ be the digraph obtained by directing edges in $E \setminus M$ from left-to-right and putting weights $w_e$ on them, and directing all edges in $M$ from right-to-left and putting weights $-w_e$ on them.

(a) Prove: $H_M$ has a negative-weight cycle iff $M$ is not a min-weight perfect matching.

(b) Consider the algorithm: Start with any perfect matching $M$. While there is a negative-weight cycle $C$ in $H_M$, set $M ← M \triangle C$. Show that you will eventually get a MwPM.

If $T$ is the time to find a negative-weight cycle in an $n$-node graph, and $W := \max_{e ∈ E} |w_e|$, a naive bound on the runtime of the above algorithm would be $O(nW) \cdot mn$, plus the time to find the first perfect matching. (Do you see why? Recall that Bellman-Ford can find the negative-weight cycle in time $T = O(mn)$.)

(c) For matching $M$, define $w(M) := \sum_{e ∈ M} w_e$. Show that if $w(M) > w(M^*)$ then there exists a cycle in $H_M$ with weight-ratio (defined in HW#1) no more than $\frac{w(M^*) - w(M)}{n}$, where $M^*$ is a min-weight perfect matching.
(d) Change the algorithm in (b) to say “While there exists a negative-weight cycle in \( H_M \), let \( C \) be the minimum weight-ratio, and set \( M \leftarrow M \triangle C \).” Bound the runtime of this algorithm by \( O(n \log(nW)) \cdot T' \), plus the time to find the first perfect matching. Here \( T' \) is the time to find a min weight-ratio cycle.

Since in HW1(#4) we gave an algorithm to find the min weight-ratio cycle in time \( T' = O(mn(\log(nW))) \), we get an algorithm that finds a MwPM in time \( O(mn^2(\log(nW))^2) \).

5. A Market-Based Bipartite Max-Matching Algorithm. Given \( G = (I, B, E) \), the left vertices are items, the right are buyers. Let \( n = \max(|I|, |B|) \). For each edge \( ib \in E \), let \( v_{ib} = 1 \); if \( ib \not\in E \), then \( v_{ib} = 0 \). The initial prices are \( p_i = 0 \); define the utility of buyer \( b \) for item \( i \) under prices \( \bar{p} = (p_1, p_2, \ldots, p_{|I|}) \) to be

\[
   u_{ib}(\bar{p}) = \max\{v_{ib} - p_i, 0\}.
\]

For some parameter \( \delta \in (0, 1) \), consider the following algorithm:

(A1) Start with the empty matching \( M = \emptyset \).

(A2) Pick any unmatched buyer \( b \) such that its highest-utility item \( i \) has \( u_{ib}(\bar{p}) \geq \delta \). Match \((i, b)\), which may require dropping \((i, b')\) for some other \( b' \) from the current matching. (So now \( i \) is assigned to \( b \) instead of \( b' \).) Raise the price \( p_i \leftarrow p_i + \delta \).

(A3) If for each unmatched buyer, \( \max_i u_{ib}(\bar{p}) < \delta \), let the current matching be denoted \( M_1 \). Use the augmenting paths algorithm to augment \( M_1 \) to a maximum matching \( M^* \).

We now show that \( M_1 \) is a “large” matching, and so step (A3) does not take “much” time.

(a) Show that \(|M_1| \geq |M^*| - n\delta\).

(Hint: you could try to use the following LP:

\[
\begin{align*}
   \max \sum_{ib} v_{ib} x_{ib} \\
   \sum_i x_{ib} &\leq 1 \\
   \sum_b x_{ib} &\leq 1 \\
   x_{ib} &\geq 0
\end{align*}
\]

And show that \(|M_1| \geq \text{OPT}_{LP} - n\delta\) using the duals.)

(b) Show how to use the appropriate data structures to implement the algorithm so that all executions of step (A2) can be done in a total of \( O(\frac{1}{\delta} \cdot (m + n \log n)) \) time.

(c) Choose the value of \( \delta \) so that the running time of the entire algorithm is as close to \( O(m\sqrt{n}) \) as possible. (You may assume that each augmenting path can be found in \( O(m) \) time, and hence (A3) takes a total of \( O(m) \times (|M^*| - |M_1|) \) time.)

Observe that this algorithm is different from the one in Lecture #8: there the prices went up in lock-step, here we just pick a single item and unilaterally raise its price by a tiny constant. We also don’t get a perfect matching here at the end, we have to do some correction (Step (A3)).

BTW, what can you say about the performance of such an algorithm when you have general weights? (You don’t have to submit any answers for these.)