27 Linear Programming Algorithms

In this lecture, we’ll see a few algorithms for actually solving linear programming problems. The most famous of these, the simplex method, was proposed by George Dantzig in 1947. Although most variants of the simplex algorithm performs well in practice, no deterministic simplex variant is known to run in sub-exponential time in the worst case. However, if the dimension of the problem is considered a constant, there are several linear programming algorithms that run in linear time. I’ll describe a particularly simple randomized algorithm due to Raimund Seidel.

My approach to describing these algorithms will rely much more heavily on geometric intuition than the usual linear-algebraic formalism. This works better for me, but your mileage may vary. For a more traditional description of the simplex algorithm, see Robert Vanderbei’s excellent textbook Linear Programming: Foundations and Extensions [Springer, 2001], which can be freely downloaded (but not legally printed) from the author’s website.

27.1 Bases, Feasibility, and Local Optimality

Consider the canonical linear program $\max\{c \cdot x \mid Ax \leq b, x \geq 0\}$, where $A$ is an $n \times d$ constraint matrix, $b$ is an $n$-dimensional coefficient vector, and $c$ is a $d$-dimensional objective vector. We will interpret this linear program geometrically as looking for the lowest point in a convex polyhedron in $\mathbb{R}^d$, described as the intersection of $n + d$ halfspaces. As in the last lecture, we will consider only non-degenerate linear programs: Every subset of $d$ constraint hyperplanes intersects in a single point; at most $d$ constraint hyperplanes pass through any point; and objective vector is linearly independent from any $d - 1$ constraint vectors.

A basis is a subset of $d$ constraints, which by our non-degeneracy assumption must be linearly independent. The location of a basis is the unique point $x$ that satisfies all $d$ constraints with equality; geometrically, $x$ is the unique intersection point of the $d$ hyperplanes. The value of a basis is $c \cdot x$, where $x$ is the location of the basis. There are precisely $\binom{n + d}{d}$ bases. Geometrically, the set of constraint hyperplanes defines a decomposition of $\mathbb{R}^d$ into convex polyhedra; this cell decomposition is called the arrangement of the hyperplanes. Every subset of $d$ hyperplanes (that is, every basis) defines a vertex of this arrangement (the location of the basis). I will use the words ‘vertex’ and ‘basis’ interchangeably.

However, there are randomized variants of the simplex algorithm that run in subexponential expected time, most notably the RANDOMFACET algorithm analyzed by Gil Kalai in 1992, and independently by Jiří Matoušek, Micha Sharir, and Emo Welzl in 1996. No randomized variant is known to run in polynomial time. In particular, in 2010, Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick proved that the worst-case expected running time of RANDOMFACET is superpolynomial.
A basis is **feasible** if its location \( x \) satisfies all the linear constraints, or geometrically, if the point \( x \) is a vertex of the polyhedron. If there are no feasible bases, the linear program is **infeasible**.

A basis is **locally optimal** if its location \( x \) is the optimal solution to the linear program with the same objective function and only the constraints in the basis. Geometrically, a basis is locally optimal if its location \( x \) is the lowest point in the intersection of those \( d \) halfspaces. A careful reading of the proof of the Strong Duality Theorem reveals that local optimality is the dual equivalent of feasibility; a basis is locally feasible for a linear program \( \Pi \) if and only if the same basis is feasible for the dual linear program \( \Pi^\ast \). For this reason, locally optimal bases are sometimes also called **dual feasible**. If there are no locally optimal bases, the linear program is **unbounded**.²

Two bases are **neighbors** if they have \( d - 1 \) constraints in common. Equivalently, in geometric terms, two vertices are neighbors if they lie on a line determined by some \( d - 1 \) constraint hyperplanes. Every basis is a neighbor of exactly \( dn \) other bases; to change a basis into one of its neighbors, there are \( d \) choices for which constraint to remove and \( n \) choices for which constraint to add. The graph of vertices and edges on the boundary of the feasible polyhedron is a subgraph of the basis graph.

The Weak Duality Theorem implies that the value of every feasible basis is less than or equal to the value of every locally optimal basis; equivalently, every feasible vertex is higher than every locally optimal vertex. The Strong Duality Theorem implies that (under our non-degeneracy assumption), if a linear program has an optimal solution, it is the unique vertex that is both feasible and locally optimal. Moreover, the optimal solution is both the lowest feasible vertex and the highest locally optimal vertex.

### 27.2 The Primal Simplex Algorithm: Falling Marbles

From a geometric standpoint, Dantzig's simplex algorithm is very simple. The input is a set \( H \) of halfspaces; we want the lowest vertex in the intersection of these halfspaces.

```plaintext
PrimalSimplex(H):
  if \( \cap H = \emptyset \)
    return INFEASIBLE
  x ← any feasible vertex
  while x is not locally optimal
    \( \langle \text{pivot downward, maintaining feasibility} \rangle \)
    if every feasible neighbor of \( x \) is higher than \( x \)
      return UNBOUNDED
    else
      x ← any feasible neighbor of \( x \) that is lower than \( x \)
  return x
```

Let's ignore the first three lines for the moment. The algorithm maintains a feasible vertex \( x \). At each so-called **pivot** operation, the algorithm moves to a lower vertex, so the algorithm never visits the same vertex more than once. Thus, the algorithm must halt after at most \( \binom{n+d}{d} \) pivots. When the algorithm halts, either the feasible vertex \( x \) is locally optimal, and therefore the optimum vertex, or the feasible vertex \( x \) is not locally optimal but has no lower feasible neighbor, in which case the feasible region must be unbounded.

²For non-degenerate linear programs, the feasible region is unbounded in the objective direction if and only if no basis is locally optimal. However, there are degenerate linear programs with no locally optimal basis that are infeasible.
Notice that we have not specified which neighbor to choose at each pivot. Many different pivoting rules have been proposed, but for almost every known pivot rule, there is an input polyhedron that requires an exponential number of pivots under that rule. No pivoting rule is known that guarantees a polynomial number of pivots in the worst case, or even in expectation.³

27.3 The Dual Simplex Algorithm: Rising Bubbles

We can also geometrically interpret the execution of the simplex algorithm on the dual linear program II. Again, the input is a set $H$ of halfspaces, and we want the lowest vertex in the intersection of these halfspaces. By the Strong Duality Theorem, this is the same as the highest locally-optimal vertex in the hyperplane arrangement.

Let’s ignore the first three lines for the moment. The algorithm maintains a locally optimal vertex $x$. At each pivot operation, it moves to a higher vertex, so the algorithm never visits the same vertex more than once. Thus, the algorithm must halt after at most $\binom{n+d}{d}$ pivots. When the algorithm halts, either the locally optimal vertex $x$ is feasible, and therefore the optimum vertex, or the locally optimal vertex $x$ is not feasible but has no higher locally optimal neighbor, in which case the problem must be infeasible.

³In 1957, Hirsch conjectured that for any linear programming instance with $d$ variables and $n + d$ constraints, starting at any feasible basis, there is a sequence of at most $n$ pivots that leads to the optimal basis. This long-standing conjecture was finally disproved in 2010 by Francisco Santos, who described an counterexample with 43 variables and 86 constraints, where the worst-case number of required pivots is 44.
From the standpoint of linear algebra, there is absolutely no difference between running PRIMALSIMPLEX on any linear program Π and running DUALSIMPLEX on the dual linear program Π. The actual code is identical. The only difference between the two algorithms is how we interpret the linear algebra geometrically.

27.4 Computing the Initial Basis

To complete our description of the simplex algorithm, we need to describe how to find the initial vertex \( x \), in the first three lines of PRIMALSIMPLEX or DUALSIMPLEX. There are several methods to find feasible or locally optimal bases, but perhaps the most natural method uses the simplex algorithm itself. Our algorithm relies on the following simple observations.

First, the feasibility of a vertex does not depend on the choice of objective vector; a vertex is either feasible for every objective function or for none. Equivalently (by duality), the local optimality of a vertex does not depend on the exact location of the \( d \) hyperplanes, but only on their normal directions and the objective function; a vertex is either locally optimal for every translation of the hyperplanes or for none. In terms of the original matrix formulation, feasibility depends on \( A \) and \( b \) but not \( c \), and local optimality depends on \( A \) and \( c \) but not \( b \).

The second important observation is that every basis is locally optimal for some objective function. Specifically, it suffices to choose any vector that has a positive inner product with each of the normal vectors of the \( d \) chosen hyperplanes. Equivalently, we can make any basis feasible by translating the hyperplanes appropriately. Specifically, it suffices to translate the chosen \( d \) hyperplanes so that they pass through the origin, and then translate all the other halfspaces so that they strictly contain the origin.

Our strategy for finding our initial feasible vertex is to choose any vertex, choose a new objective function that makes that vertex locally optimal, and then find the optimal vertex for that objective function by running the (dual) simplex algorithm. This vertex must be feasible, even after we restore the original objective function!

(a) (b) (c)

(a) Choose any basis. (b) Rotate objective to make it locally optimal, and pivot ‘upward’ to find a feasible basis. (c) Pivot downward to the optimum basis for the original objective.

Equivalently, to find an initial locally optimal vertex, we choose any vertex, translate the hyperplanes so that that vertex becomes feasible, and then find the optimal vertex for those translated constraints using the (primal) simplex algorithm. This vertex must be locally optimal, even after we restore the hyperplanes to their original locations!
(a) Choose any basis. (b) Translate constraints to make it feasible, and pivot downward to find a locally optimal basis. (c) Pivot upward to the optimum basis for the original constraints.

Here are more complete pseudocode descriptions of the primal and dual simplex algorithms with these initialization rules. As usual, the input to both algorithms is a set $H$ of halfspaces, and the algorithms either return the lowest vertex in the intersection of these halfspaces or report that no such vertex exists.

**DualPrimalSimplex($H$):**

```
x ← any vertex
\tilde{H} ← any rotation of $H$ that makes $x$ locally optimal
while $x$ is not feasible
    if every locally optimal neighbor of $x$ is lower (wrt $\tilde{H}$) than $x$
        return INFEASIBLE
    else
        $x ←$ any locally optimal neighbor of $x$ that is higher (wrt $\tilde{H}$) than $x$
while $x$ is not locally optimal
    if every feasible neighbor of $x$ is higher than $x$
        return UNBOUNDED
    else
        $x ←$ any feasible neighbor of $x$ that is lower than $x$
return $x$
```

**PrimalDualSimplex($H$):**

```
x ← any vertex
\tilde{H} ← any translation of $H$ that makes $x$ feasible
while $x$ is not locally optimal
    if every feasible neighbor of $x$ is higher (wrt $\tilde{H}$) than $x$
        return UNBOUNDED
    else
        $x ←$ any feasible neighbor of $x$ that is lower (wrt $\tilde{H}$) than $x$
while $x$ is not feasible
    if every locally optimal neighbor of $x$ is lower than $x$
        return INFEASIBLE
    else
        $x ←$ any locally-optimal neighbor of $x$ that is higher than $x$
return $x$
```

Sketch (or at least) mention polynomial-time ellipsoid and interior-point methods.
27.5 Network Simplex

Our first natural examples of linear programming problems were shortest paths, maximum flows, and minimum cuts in edge-weighted graphs. It is instructive to reinterpret the behavior of the abstract simplex algorithm in terms of the original input graphs; this reinterpretation allows for a much more efficient implementation of the simplex algorithm, which is normally called network simplex.

As a concrete working example, let’s consider a special case of the minimum-cost flow problem called the transshipment problem. The input consists of a directed graph $G = (V, E)$, a balance function $b : V \to \mathbb{R}$, and a cost function $\$: $E \to \mathbb{R}$. Our goal is to compute a flow function $f : E \to \mathbb{R}$ that is non-negative everywhere, satisfies the balance constraint

$$\sum_{u \to v} f(u \to v) - \sum_{v \to w} f(v \to w) = b(v)$$

at every vertex $v$, and minimizes the total cost $\sum_{e} f(e) \cdot \$. We can easily express this problem as a linear program with a variable for each edge and constraints for each vertex and edge.

$$\begin{align*}
\text{maximize} & \quad \sum_{u \to v} \cdot f(u \to v) \\
\text{subject to} & \quad \sum_{u \to v} f(u \to v) - \sum_{v \to w} f(v \to w) = b(v) \quad \text{for every vertex } v \\
& \quad f(u \to v) \geq 0 \quad \text{for every edge } u \to v
\end{align*}$$

By interpreting the balance, cost, and flow functions as vectors, we can write this linear program more succinctly as follows:

$$\begin{array}{l}
\text{max} & \quad \cdot f \\
\text{s.t.} & \quad Af = b \\
& \quad f \geq 0
\end{array}$$

Here $A$ is the vertex-edge incidence matrix of $G$, which has one row for each vertex and one column for each edge, and whose entries are defined as follows:

$$A(v, x \to y) = \begin{cases} 
1 & \text{if } v = y \\
-1 & \text{if } v = x \\
0 & \text{otherwise}
\end{cases}$$

Lemma 1. The rank of $A$ is $V - 1$.

Proof: If we add the columns of $A$ corresponding to any directed cycle in $G$, th…

★★★★
A has rank $V - 1$: balance at all but one vertex determines balance at the last vertex.

Basis consists $E - V + 1$ edges with flow value 0.

The $V - 1$ edges not in a basis always define a spanning tree of $G$. If non-basis edges contained a cycle, we could push flow around it, so the constraints are not linearly independent. (If non-basis edges were disconnected, we would have a cut with balance zero, which is degenerate.)

For any spanning tree $T$, there is a unique flow $\text{flow}_T$ that meets all the balance constraints; easily computable in $O(V)$ time.

For any non-tree edge $e$, define $\text{slack}_T(e)$ as residual cost of fundamental cycle in $T + e$.

A spanning tree $T$ (or more formally, the basis that defines it) is feasible iff $\text{flow}_T \geq 0$ and locally optimal iff $\text{slack}_T \geq 0$.

Pivot = remove one edge $e_{\text{out}}$ and add one edge $e_{\text{in}}$ to get a new tree. Edge $e_{\text{out}}$ must be in fundamental cycle defined by $e_{\text{in}}$; conversely, $e_{\text{in}}$ must be in fundamental cut defined by $e_{\text{out}}$. Flows in the cycle all decrease by $f(e_{\text{out}})$; slacks in the cut decrease by $\text{slack}(e_{\text{in}})$.

Primal simplex: Initial feasible basis: Find feasible (=non-negative) flow by adding source (connected to surplus vertices) and target (connected to deficit vertices) and computing a maximum flow. Primal pivot = cancel a negative-slack fundamental cycle.

Dual simplex: Any shortest-path tree is a locally optimal basis. Dual pivot = cancel an negative-flow fundamental cut.

### 27.6 Linear Expected Time for Fixed Dimensions

In most geometric applications of linear programming, the number of variables is a small constant, but the number of constraints may still be very large.

The input to the following algorithm is a set $H$ of $n$ halfspaces and a set $B$ of $b$ hyperplanes. ($B$ stands for "basis." The algorithm returns the lowest point in the intersection of the halfspaces in $H$ and the hyperplanes $B$. At the top level of recursion, $B$ is empty. I will implicitly assume that the linear program is both feasible and bounded. (If necessary, we can guarantee boundedness by adding a single halfspace to $H$, and we can guarantee feasibility by adding a dimension.) A point $x$ violates a constraint $h$ if it is not contained in the corresponding halfspace.

```
SEIDELLP(H, B):
  if |B| = d
    return $\bigcap B$
  if |H $\cup$ B| = d
    return $\bigcap (H \cup B)$
  $h \leftarrow$ random element of $H$
  $x \leftarrow$ SEIDELLP($H \setminus h, B$)  (#)
  if $x$ violates $h$
    return SEIDELLP($H \setminus h, B \cup \partial h$)
  else
    return $x$
```

The point $x$ recursively computed in line (#) is the optimal solution if and only if the random halfspace $h$ is not one of the $d$ halfspaces that define the optimal solution. In other words, the probability of calling $\text{SEIDELLP}(H, B \cup h)$ is exactly $(d - b)/n$. Thus, we have the following recurrence for the expected number of recursive calls for this algorithm:

$$T(n, b) = \begin{cases} 
1 & \text{if } b = d \text{ or } n + b = d \\
T(n - 1, b) + \frac{d - b}{n} \cdot T(n - 1, b + 1) & \text{otherwise}
\end{cases}$$
The recurrence is somewhat simpler if we write $\delta = d - b$:

$$T(n, \delta) = \begin{cases} 1 & \text{if } \delta = 0 \text{ or } n = \delta \\ T(n-1, \delta) + \frac{\delta}{n} \cdot T(n-1, \delta - 1) & \text{otherwise} \end{cases}$$

It's easy to prove by induction that $T(n, \delta) = O(\delta!n)$:

$$T(n, \delta) = T(n-1, \delta) + \frac{\delta}{n} \cdot T(n-1, \delta - 1)$$

$$\leq \delta!(n-1) + \frac{\delta}{n} \cdot (\delta - 1)! \cdot (n-1) \quad \text{[induction hypothesis]}$$

$$= \delta!(n-1) + \delta! \cdot \frac{n-1}{n}$$

$$\leq \delta! \cdot n$$

At the top level of recursion, we perform one violation test in $O(d)$ time. In each of the base cases, we spend $O(d^3)$ time computing the intersection point of $d$ hyperplanes, and in the first base case, we spend $O(dn)$ additional time testing for violations. More careful analysis implies that the algorithm runs in $O(d! \cdot n)$ expected time.
Exercises

1. Fix a non-degenerate linear program in canonical form with \(d\) variables and \(n + d\) constraints.
   (a) Prove that every feasible basis has exactly \(d\) feasible neighbors.
   (b) Prove that every locally optimal basis has exactly \(n\) locally optimal neighbors.

2. (a) Give an example of a non-empty polyhedron \(Ax \leq b\) that is unbounded for every objective vector \(c\).
   (b) Give an example of an infeasible linear program whose dual is also infeasible.
   In both cases, your linear program will be degenerate.

3. Describe and analyze an algorithm that solves the following problem in \(O(n)\) time: Given \(n\) red points and \(n\) blue points in the plane, either find a line that separates every red point from every blue point, or prove that no such line exists.

4. In this exercise, we develop another standard method for computing an initial feasible basis for the primal simplex algorithm. In both cases, we are given a canonical linear program \(\Pi\) with \(d\) variables and \(n + d\) constraints as input:

\[
\begin{align*}
\text{max} & \quad c \cdot x \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
\]

To compute an initial feasible basis for \(\Pi\), we solve a modified linear program \(\Pi'\) defined by introducing a new variable \(\lambda\) and two new constraints \(0 \leq \lambda \leq 1\), and modifying the objective function:

\[
\begin{align*}
\text{max} & \quad \lambda \\
\text{s.t.} & \quad Ax - b\lambda \leq 0 \\
& \quad \lambda \leq 1 \\
& \quad x, \lambda \geq 0
\end{align*}
\]

(a) Prove that \(x_1 = x_2 = \cdots = x_d = \lambda = 0\) is a feasible basis for \(\Pi'\).
(b) Prove that \(\Pi\) is feasible if and only if the optimal value for \(\Pi'\) is 1.
(c) What is the dual of \(\Pi'\)?

5. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves arbitrary linear programs in polynomial time. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. [Hint: Add one variable and one constraint.]
6. A minor of a matrix $A$ is the submatrix defined by any subset of the rows and any subset of the columns. A matrix $A$ is totally unimodular if, for every square minor $M$, the determinant of $M$ is $-1$, $0$, or $1$.

(a) Let $A$ be an arbitrary totally unimodular matrix.

i. Prove that the transposed matrix $A^\top$ is also totally unimodular.

ii. Prove that negating any row or column of $A$ leaves the matrix totally unimodular.

iii. Prove that the block matrix $[A | I]$ is totally unimodular.

(b) Prove that for any totally unimodular matrix $A$ and any integer vector $b$, the canonical linear program $\max \{c \cdot x | Ax \leq b, x \geq 0\}$ has an integer optimal solution. [Hint: Cramer’s rule.]

(c) The unsigned incidence matrix of an undirected graph $G = (V,E)$ is an $|V| \times |E|$ matrix $A$, with rows indexed by vertices and columns indexed by edges, where for each row $v$ and column $uw$, we have

$$A[v, uw] = \begin{cases} 1 & \text{if } v = u \text{ or } v = w \\ 0 & \text{otherwise} \end{cases}$$

Prove that the unsigned incidence matrix of every bipartite graph $G$ is totally unimodular. [Hint: Each square minor corresponds to a subgraph $H$ of $G$ with $k$ vertices and $k$ edges, for some integer $k$. Argue that at least one of the following statements must be true: (1) $H$ is disconnected; (2) $H$ has a vertex with degree 1; (3) $H$ is an even cycle.]

(d) Prove that for every non-bipartite graph $G$ that the unsigned incidence matrix of $G$ is not totally unimodular. [Hint: Consider any odd cycle.]

(e) The signed incidence matrix of a directed graph $G = (V,E)$ is also an $|V| \times |E|$ matrix $A$, with rows indexed by vertices and columns indexed by edges, where for each row $v$ and column $u \rightarrow w$, we have

$$A[u, v \rightarrow w] = \begin{cases} 1 & \text{if } v = w \\ -1 & \text{if } v = u \\ 0 & \text{otherwise} \end{cases}$$

Prove that the signed incidence matrix of every directed graph $G$ is totally unimodular.

7. Recall that the single-source shortest path problem can be formulated as a linear programming problem, with one variable $d_v$ for each vertex $v \neq s$ in the input graph, as follows:

$$\begin{align*}
\text{maximize} & \quad \sum_v d_v \\
\text{subject to} & \quad d_v \leq \ell_{s \rightarrow v} \quad \text{for every edge } s \rightarrow v \\
& \quad d_v - d_u \leq \ell_{u \rightarrow v} \quad \text{for every edge } u \rightarrow v \text{ with } u \neq s \\
& \quad d_v \geq 0 \quad \text{for every vertex } v \neq s
\end{align*}$$

This problem asks you to describe the behavior of the simplex algorithm on this linear program in terms of distances. Assume that the edge weights $\ell_{u \rightarrow v}$ are all non-negative and that there is a unique shortest path between any two vertices in the graph.
(a) What is a basis for this linear program? What is a feasible basis? What is a locally optimal basis?

(b) Show that in the optimal basis, every variable \( d_v \) is equal to the shortest-path distance from \( s \) to \( v \).

(c) Describe the primal simplex algorithm for the shortest-path linear program directly in terms of vertex distances. In particular, what does it mean to pivot from a feasible basis to a neighboring feasible basis, and how can we execute such a pivot quickly?

(d) Describe the dual simplex algorithm for the shortest-path linear program directly in terms of vertex distances. In particular, what does it mean to pivot from a locally optimal basis to a neighboring locally optimal basis, and how can we execute such a pivot quickly?


(f) Using the results in problem 6, prove that if the edge lengths \( \ell_{u \rightarrow v} \) are all integral, then the optimal distances \( d_v \) are also integral.

8. The maximum \((s, t)\)-flow problem can be formulated as a linear programming problem, with one variable \( f_{u \rightarrow v} \) for each edge \( u \rightarrow v \) in the input graph:

\[
\begin{align*}
\text{maximize} & \quad \sum_w f_{s \rightarrow w} - \sum_u f_{u \rightarrow s} \\
\text{subject to} & \quad \sum_w f_{v \rightarrow w} - \sum_u f_{u \rightarrow v} = 0 \quad \text{for every vertex } v \neq s, t \\
& \quad f_{u \rightarrow v} \leq c_{u \rightarrow v} \quad \text{for every edge } u \rightarrow v \\
& \quad f_{u \rightarrow v} \geq 0 \quad \text{for every edge } u \rightarrow v
\end{align*}
\]

This problem asks you to describe the behavior of the simplex algorithm on this linear program in terms of flows.

(a) What is a basis for this linear program? What is a feasible basis? What is a locally optimal basis?

(b) Show that the optimal basis represents a maximum flow.

(c) Describe the primal simplex algorithm for the flow linear program directly in terms of flows. In particular, what does it mean to pivot from a feasible basis to a neighboring feasible basis, and how can we execute such a pivot quickly?

(d) Describe the dual simplex algorithm for the flow linear program directly in terms of flows. In particular, what does it mean to pivot from a locally optimal basis to a neighboring locally optimal basis, and how can we execute such a pivot quickly?

(e) Is the Ford-Fulkerson augmenting-path algorithm an instance of network simplex? Justify your answer. [Hint: There is a one-line argument.]

(f) Using the results in problem 6, prove that if the capacities \( c_{u \rightarrow v} \) are all integral, then the maximum flow values \( f_{u \rightarrow v} \) are also integral.