CME 305: Discrete Mathematics and Algorithms
Instructor: Professor Aaron Sidford (sidford@stanford.edu)
January 18, 2018

Lecture 4 - Matroids and Maximum Flow[1]

There are two main topics we cover in this lecture. First, we introduce and study *matroids* a generalization of spanning trees and the notion of independence that embody the greedy algorithm we saw last class. Second, we start a new problem, the maximum flow problem. In some sense, todays lecture is about one of the easiest and hardest problems in combinatorial optimization for which we have reasonably fast algorithm. We will see how to optimize over matroids easily (i.e. in nearly linear time), whereas solving the maximum flow problem in nearly linear time is one of the biggest open problems in combinatorial optimization.

1 Matroids

Matroids are a mathematical construct that generalize spanning trees and more broadly the notion of *independence*. While matroids are fairly abstract, they have enough structure to allow us to reason about concepts like cuts, cycles, and greedy optimization algorithms. Reasoning about such general combinatorial objects is a common technique in discrete optimization and powerful lens for obtaining perspective on the structure of particular problems and the reasons for certain algorithms to work. Obviously, the downside in working in such generality is that specific motivations for studying such objects can occasionally be lost. The appeal of such mathematical abstractions is a matter of taste, but it is important to have a feel for how this is done if one wishes to dig deeper into discrete mathematics or combinatorial optimization literature. As we will see in this lecture, the study of matroids will ultimately give a broad characterization of when the greedy algorithm works for solving linear optimization problems.

We begin by formally defining a matroid as follows.

Definition 1 (Matroid)**.** *A matroid (S, I) consists of a finite set S, known as the* ground set*, and a non-empty set of subsets of the ground set $I \subseteq 2^S$ that obey two properties*

1. **hereditary property**: if $A \in I$ and $B \subseteq A$ then $B \in I$
2. **exchange property**: if $A, B \in I$ and $|A| < |B|$ then $\exists b \in B \setminus A$ s.t. $A \cup \{b\} \in I$

We typically call the sets in I the *independent sets* and we call $A \in I$ *independent*. Similarly, we call the sets not in I the *dependent sets* and we call $A \notin I$ dependent.

Note that a matroid is simply a set of subsets I of a finite set S that obey two natural properties, the *hereditary property* and the *exchange property*.

The hereditary property simply says that all subsets of independent sets need to be independent too. Note this also immediately implies that if a set is dependent, then all sets that contain it are dependent.

The exchange property says that if we have two independent sets and one has larger cardinality then the other, then we can always find an element from the larger set that is not in the smaller, such that adding it to the smaller preserves independence. Note that neither property is redundant, it is possible to have the hereditary property but not the exchange property and vice versa.

Note that the exchange property immediately implies that all maximal independent sets (that is $A \in I$ where $A \cup \{b\} \notin I$ for all $b \in S \setminus A$) have the same size.

[1]These lecture notes are a work in progress and there may be typos, awkward language, omitted proof details, etc. Moreover, content from lectures might be missing. These notes are intended to converge to a polished superset of the material covered in class so if you would like anything clarified, please do not hesitate to post to Piazza and ask.

Lemma 2. *If (S, I) is a matroid and $A, B \in I$ are maximal independent sets, then $|A| = |B|$.*

Proof. Proceed by contradiction and suppose $|A| < |B|$. By the exchange property there is some $b \in B \setminus A$ with $A \cup \{b\} \in I$ and therefore A is not maximal. $\square$

This makes the following notion of rank well defined.

Definition 3 (Matroid Rank)**.** The size of a maximum independent set in a matroid is called the *rank* of that matroid.

1.1 Matroid Examples

Let's see some examples of matroids. The first, known as linear matroids, is what is naturally suggested by our use of the terms independence and rank.

Definition 4 (Linear Matroid)**.** If $S = \{v_1, ..., v_n\}$ where each v_i is a vector from the same vector space (e.g. $v_i \in \mathbb{R}^d$ for all $i \in [n]$) and I is the set of linearly independent subsets of S, i.e.

$$S = \left\{ v_{i_1}, v_{i_2}, ..., v_{i_k} | \sum_{j \in [k]} \alpha_j v_{i_j} = 0 \Leftrightarrow \forall j, \alpha_j = 0 \right\} ,$$

then (S, I) is a matroid known as a linear matroid.

The fact that linear matroids are matroids follows from standard analysis in linear algebra which we will not repeat here. Note that the term rank of the matroid coincide with the dimension of the space spanned by the vectors v_i, i.e. the rank of the matrix $\mathbf{A}$ with columns v_i. Furthermore, the set of maximal linearly independent sets of a matroid constitute the set of basis of the vector space spanned by the v_i.

The next example is our motivation for studying matroids in the first place, graphic matroids.

Definition 5 (Graphic Matroids)**.** If $G = (V, E)$ is an undirected graph, $S = E$, and I is the set of edges from E that constitute a forest in G (i.e. an acyclic subgraph) then (S, I) is a matroid known as a *graphic matroid.*

Interestingly (though we will not show it in this lecture), all graphic matroids are linear matroids. Note that the set of maximal independent sets in a graphic matroid of a connected graph are the spanning trees of the graph and therefor the rank of a graphic matroid is $n - 1$. For completeness we both prove the rank of graphic matroids in the general case and the number of independent sets they contain.

Lemma 6. *The rank of a graphic matroid, i.e. maximum number of edges in an acyclic subgraph of an undirected graph, is $n - c$ where c is the number of connected components in the associated graph.*

Proof. Suppose we are given an undirected graph $G = (V, E)$ and we add an edge e to it. If e does not create a cycle its endpoints must be between two different connected components and therefore adding the edge decreases the number of connected components by 1. Consequently if m edges are added to an empty graph without creating a cycle the number of resulting connected components $c = n - m$. $\square$

Lemma 7. *A graphic matroid is a matroid.*

Proof. Let $G = (V, E)$ be an undirected graph and let (S, I) be its associated graphic matroid. Clearly (S, I) has the hereditary property as deleting edges from a subgraph cannot induce cycles. To see that (S, I) has the exchange property let F_1, F_2 be two independent sets or forests in G and suppose that $|F_1| < |F_2|$. By the previous lemma F_2 has less connected components then F_1. Furthermore, $F_1 \cup F_2$ has at most the number of connected components of F_2 and consequently at least one edge $e \in F_2$ has endpoints in two different connected components in F_1 and therefore $F_1 \cup \{e\}$ is independent. $\square$

There are many more example of matroids. For example if S is any finite set and $I = \{A \subseteq S \,|\, |A| \leq k\}$ then (S, I) is a matroid (known as the uniform matroid). There is also a theory for constructing certain types of matroids. For example, if $G = (V, E)$ is an undirected graph and $S = E$ and $I = \{F \subseteq E | (V, E \setminus F) \text{ is connected}\}$ is a matroid. Though we will not prove it, this is because this matroid is the dual of a matroid which is always a matroid.

Definition 8 (Dual Matroid)**.** For any matroid (S, I) its *dual* matroid (S, I^*) is given by $I^* = \{A \subseteq S | A \text{ is disjoint from some maximal } B \in I\}$.

Though we will not prove it, a dual matroid is always a matroid and the dual matroid of the dual matroid is the original matroid. This is mentioned to give a taste of some of the broader behind constructing matroids and reasoning about them.

1.2 Optimizing over Matroids

One of the nice properties about matroids, and the reason we are discussing them here, is that optimizing over them is fairly easy and the algorithm that does it is a generalization of the algorithm for computing minimum weight spanning trees. In the remainder of this section we consider the following (linear) matroid optimization problem.

Definition 9 ((Linear) Matroid Optimization)**.** The (linear) matroid optimization problem is as follows: given a matroid (S, I) and weights $w \in \mathbb{R}^S$ compute a maximal independent set A that minimizes $\sum_{a \in A} w_a$.

To properly address this question we need to discuss how (S, I) is represented. If I was represented directly set by set, then this optimization problem could be easily solved in linear time (by just trying every set in I). However, as we saw with things like graphic matroids, the matroid can be represented more compactly (e.g. a graph can have an exponential number of spanning trees). To provide general tools for solving this problem that may be useful in different scenarios we therefore abstract away the representation of the matroid by instead of assuming we have the matroid directly instead assuming that we can efficiently query properties of the matroid. There are many such *oracles* that might make sense, but the one we consider here is the following.

Definition 10 (Independence Oracle)**.** For a matroid (S, I) an *independence oracle* is a function that given $A \subseteq S$ can output whether or not $A \in I$ with a single query that we assume takes $O(\mathcal{T})$ time.

Consequently, we now ask the question how efficiently can we solve the linear matroid optimization problem given an independence oracle? As was suggested a natural algorithm to try is the greedy algorithm.

Below we analyze this algorithm. We actually show that it doesn't just output the maximal independent set, we actually show it does something stronger, it computes that minimum weight independent set of every cardinality.

Lemma 11. *For all k the set A_k is a minimum weight independent set of cardinality k.*

Algorithm 1 Greedy Matroid Optimization Algorithm

```
Input: set $S$, weights $w \in \mathbb{R}^S$, and a independence oracle for matroid $(S, I)$
Let $S = \{1, ..., n\}$ and sort the weights so $w_1 \leq w_2 \leq ... \leq w_n$
Let $k = 0$ and $A_0 = \emptyset$
for $i = 1$ to $n$ do
  if $A_k \cup \{i\} \in I$ then
    $A_{k+1} = A_k \cup \{i\}$
    $k := k + 1$
  end if
end for
Return $A_k$
```

Proof. Let $a_1, ..., a_k \in [n]$ be the elements added to A_k in order of increasing weight, i.e. $A_j = \{a_1, ...a_j\}$ for all $j \in [k]$. Now proceed by contradiction and suppose that for some s the set $B_s = \{b_1, ..., b_s\} \in I$ and has $\sum_{i\in[s]} w(b_i) < \sum_{i\in[s]} w(a_i)$. Let $j \in \mathbb{Z}_{>0}$ be the minimum value where $w(b_j) < w(a_j)$; note there must be such a j as $\sum_{i\in[k]} w(b_i) < \sum_{i\in[k]} w(a_i)$. Then since $j = |\{b_1, ..., b_j\}| > |A_{j-1}| = j - 1$ by the exchange property there is some $l \in [j]$ with $A_{j-1} \cup \{b_l\} \in I$. Furthermore, by assumption $w(b_l) \leq w(b_j) < w(a_j)$. However, since $A_{j-1} \cup \{b_l\} \in I$ and $w(b_l) < w(a_j)$ the greedy algorithm would have added b_l before it added a_j and thus we have a contradiction. □

Theorem 12. *In $O(n \log n + n\mathcal{T})$ time the greedy algorithm outputs a minimum weight independent set.*

Proof. Correctness is immediate from the previous lemma and the running time follows from sorting taking $O(n \log n)$ time and computing independence n times taking $O(n\mathcal{T})$ times. □

Thus we see that the greedy algorithm very efficiently, i.e. nearly linear time, solves the (linear) matroid optimization problem.

Interestingly, it can be shown (though we will not prove it here) that any set system that has the hereditary property, for which greedy always works, must be a matroid. Consequently, the greedy algorithm in some sense categorizes a broad class of combinatorial optimization problems.

Theorem 13. *Suppose S is a finite set $I \subseteq 2^S$ is a set of subsets satisfying the hereditary property. Furthermore, suppose that for all $w \in \mathbb{R}^S$ the greedy algorithm output a maximal element of I of minimal weight, then (S, I) satisfies the exchange property (i.e. is a matroid).*

1.3 Cuts and Circuits

When we analyzed the greedy MST algorithm, we actually showed something more general. We showed that if any cut has a unique minimum weight edge, then that edge is in any MST. Here we can show that matroids have similar structure. Here we provide this structure and thereby give an alternative proof of the greedy algorithm. This shows that much of our MST analysis actually holds in much greater generality.

We begin by defining generalizations of *cycles* (*circuits*) and *cuts* (*co-circuits*) in a matroid.

Definition 14 (Circuits)**.** In a matroid (S, I) a set $A \subseteq S$ is called a *circuit* if it is a minimal dependent set and it is called a *co-circuit* if it is a minimal set with a non-trivial intersection with every maximal independent set.

The naming follows from the fact that it can be shown that a circuit in a matroid is a co-circuit in its dual and vice versa. Note that a set is a circuit in a graphic matroid if and only if it is a cycle in the graph. However, the definition of co-circuits are a little more restrictive; there are cuts which are not co-circuits as the co-circuit definition is a little more restrictive as it needs to be minimal. For example if a cut makes more than two connected component, then there is a smaller cut which disconnects the graph (by adding back the edges to one of the removed components).

Now, we wish to prove the following generalization of our result about the structure of minimum spanning trees.

Lemma 15. *If (S, I) is a matroid, $w \in \mathbb{R}^S$, $A \subseteq S$ is a co-circuit, and $a \in A$ has the property that $w_a < w_b$ for all $b \in A \setminus \{a\}$ then a is in every minimum weight maximum independent set.*

Before we prove this lemma, we prove a few helper lemmas which are generalizations of what we have seen so far about minimum spanning trees. First we show how elements of a cycle are interchangeable in independent sets.

Lemma 16. *Let (S, I) be a matroid $C \subseteq S$ be a circuit and $A \subseteq S$ be disjoint from C. If for some $x \in C$ the set $A \cup [C \setminus \{x\}] \in I$ then for all $y \in C$ the set $A \cup [C \setminus \{y\}] \in I$.*

Proof. Proceed by contradiction and suppose for some $y \in C$ with $y \neq x$ we have $A \cup [C \setminus \{y\}] \notin I$. Let $B \subseteq A$ be maximal subset of A such that $B \cup [C \setminus \{y\}] \in I$. We know such a set exists (thought it is possibly the empty-set) as $C \setminus \{y\} \in I$ by the definition of a circuit. Furthermore, we know that $|B| < |A|$ by assumption and therefore $|A \cup [C \setminus \{x\}]| > |B \cup [C \setminus \{y\}]|$. Consequently by the exchange property there is $a \in |A \cup [C \setminus \{x\}]$ with $a \notin B \cup [C \setminus \{y\}]$ such that $a \cup B \cup [C \setminus \{y\}] \in I$. However, $a \neq \{y\}$ as $B \cup C \notin I$ as $C \notin I$. Therefore, $\{a\} \cup B$ is a contradiction to the maximality of B. □

Next we provide an interesting fact that a circuit and co-circuit cannot intersect only once.

Lemma 17. *If (S, I) is a matroid and $A \subseteq S$ is a circuit and $B \subseteq S$ is a co-circuit then $|A \cap B| \neq 1$.*

Proof. Suppose $A \cap B \neq \emptyset$ and let $x \in A \cap B$ be arbitrary. By definition of co-circuit there is some maximal independent set C that is disjoint from $B \setminus \{x\}$ that contains x. Furthermore, $A \setminus \{x\} \in I$ by definition of a circuit and therefore by repeated application of the exchange lemma there is $D \subseteq C$ so that $D \cup [A \setminus \{x\}]$ is a maximal independent set. Consequently, by definition of co-circuit $D \cup [A \setminus \{x\}] \cap B \neq \emptyset$. However, $D \subseteq C$ which is disjoint from B and therefore $[A \setminus \{x\}] \cap B \neq \emptyset$. Consequently, either $A \cap B = \emptyset$ or $|A \cap B| \geq 2$. □

Putting these lemmas let us prove the Lemma 15.

Proof of Lemma 15. Proceed by contradiction and let $T \subseteq S$ be a minimum weight independent set not containing a. Let C be a minimal dependent set in $T \cup \{a\}$; such a cycle is unique and called the *fundamental cycle* but we will not need this fact. Clearly $a \in C$ since $T \in I$. Now by Lemma 17 since $C \cap A \neq \emptyset$ there is some $b \in C \cap A$ with $b \neq a$. Now since C is a minimal dependent set it must be the case that $C \setminus \{b\} \in I$. Now let $E = T \setminus C$. Since clearly E is disjoint from C and since $T = E \cup (C \setminus \{a\}) \in I$ by Lemma 16 we have that $E \cup (C \setminus \{b\}) \in I$. Consequently, $E \cup (C \setminus \{b\}) \in I$ is also a maximal independent set but it has larger weight then T, since $w_b > w_a$ yielding a contradiction to the minimum weight of T. □

These lemmas hopefully convey how general some of our analysis of spanning trees is.

2 Maximum Flow

In the remainder of this lecture we will discuss a new combinatorial optimization problem known as the *maximum flow problem*. This is perhaps one of the most famous and well studied problems in combinatorial optimization and has been an active area of research for decades. Whereas the running time complexity of computing MST is nearly settled it is still fairly open what the best running time for solving the maximum flow problem is. As we will see by the end of the next lecture, the maximum flow problem is polynomial time solvable, but the best running times are still somewhat far from nearly linear. Obtaining nearly linear time maximum flow problems is still one of the biggest open problems in the design of efficient graph algorithms.[2]

The input to the maximum flow problem is as follows:

- Directed graph $G = (V, E)$
- Vertex $s \in V$ called the *source*
- Vertex $t \in V$ with $t \neq s$ called the sink
- Non-negative edge capacities $u \in \mathbb{R}^E_{\geq 0}$

Informally, the goal of the maximum flow problem asks one to compute the maximum amount of stuff one can send from s to t without exceeding the capacities. Here we think of every edge $e = (a, b) \in E$ as a pipe or information channel capable of carrying u_e units of stuff or *flow* from a to b and we wish to find the maximum amount of flow that can be send from s to t. In a system of pipes this would be how much stuff can simultaneously be flowing from s to t, in an information network this might be how many bits can be downloaded at a time from s at t, in a round network this might be how many cars can be simultaneously driving from s towards t.

Formally, we define a s-t flow as follows.

Definition 18 (s-t flow)**.** We call an assignment of non-negative real values $f \in \mathbb{R}^E_{\geq 0}$ to the edges of the graph a s-t flow it obeys flow conservation, meaning that the *imbalance* of f and vertex u, defined as

$$\mathrm{im}(f, u) = \sum_{e=(u,v)\in E} f(e) - \sum_{e=(v,u)\in E} f(e)$$

satisfies that $\mathrm{im}(f, u) = 0$ for all $u \notin \{s, t\}$.

Here the interpretation is that if $f \in \mathbb{R}^E_{\geq 0}$ and $e = (u, v) \in E$ then the flow f on edge e results in $f(e)$ units of flow leaving u and $f(e)$ units of flow entering v. $\mathrm{im}(f, u)$ then corresponds to the net amount of flow leaving u. We are calling f a s-t flow if $\mathrm{im}(f, u) = 0$ for all $u \notin \{s, t\}$ meaning that for every vertex that is not s or t all the flow that comes in goes out, meaning the net flow in or out of $u \notin \{s, t\}$ is 0.

Occasionally we might be a little informal with our notation for f when the meaning of flow in or out is clear. If there is an edge $e = (u, v) \in E$ we typically write $f(e)$ or $f(u, v)$ for the amount of flow on edge e and say $f(e)$ units leave u and enter v. However, we also might occasionally write $f(v, u)$ with the convention that $f(v, u) = -f(e)$ meaning that $-f(e)$ units leave v and $-f(e)$ units enter u as this is consistent with interpretation and doesn't change the definition of imbalance if used correctly. Consequently, we think of $f(u, v)$ as a skew-symmetric function on edge pairs, i.e. $f(u, v) = -f(v, u)$ in the case when the graph has no multi-edges.

Now, as we said we wish to only consider s-t flows that do not exceed the capacity constraints. We define this formally as follows.

[2] I should confess that I am a bit biased here as this is a research problem I work on fairly regularly.

Definition 19 (Capacity Constraints)**.** We call a s-t flow $f \in \mathbb{R}^E_{\geq 0}$ *feasible* or say it meets the capacity constraints if and only if for all edge $e \in E$ we have $0 \leq f(e) \leq u_e$.

Lastly, we need to quantify what we mean when we talk about how much flow we send from s to t. We formalize this through the notion of value of a flow.

Definition 20 (Flow Value)**.** For feasible s-t flow $f \in \mathbb{R}^E_{\geq 0}$ we denote the value of the flow $v(f)$ as $v(f) = \mathrm{im}(f, s)$, i.e. the net amount of flow leaving s. We say that f send $v(f)$ units of flow from s to t.

With these definitions in hand we can formally define the maximum flow problem.

Definition 21 (Maximum Flow Problem)**.** The maximum flow problem asks to compute a feasible s-t flow f with maximum value $v(f)$. We let v^* denote this optimal value.

The maximum flow problem has numerous applications as we will see in later lectures and there are many problems which while on the surface may seem unrelated to the maximum flow problem, can easily be reduced to it. In the remainder of this lecture we take a closer look at the structure of the maximum flow problem and begin to discuss solving it. We will provide actual algorithms for it in the next lecture.

First, we take a closer look at $v(f)$. Note that we could also intuitively have define $v(f)$ as the amount of flow entering t. We prove this by first proving an easy fact about imbalance.

Lemma 22. *For any flow $f \in \mathbb{R}^E_{\geq 0}$ (not just feasible ones) we have $\sum_{v \in V} \mathrm{im}(f, v) = 0$.*

Proof. Note that in $\sum_{v \in V} \mathrm{im}(f, v)$ every edge (u, v) contributes both $f_{(u,v)}$ and $-f_{(u,v)}$ to the sum. $\square$

This lemma simply says that for any flow the net flow leaving the vertices is 0, i.e. all flow that comes in, must go out. Using this we can obtain several characterizations of flow value.

Lemma 23. $v(f) = -\mathrm{im}(f, t)$ *for all feasible* $f \in \mathbb{R}^E_{\geq 0}$.

Proof. Since $\sum_{v \in V} \mathrm{im}(f, v) = 0$ and $\mathrm{im}(f, v) = 0$ for all $v \notin \{s, t\}$ we have $\mathrm{im}(f, s) + \mathrm{im}(f, t) = 0$ and therefore $v(f) = \mathrm{im}(f, s) = -\mathrm{im}(f, t)$. $\square$

Note that $v(f)$ is simply the net flow across the cut induced by $\{s\}$ and $-\mathrm{im}(f, t)$ is simply the negative of the net flow across the cut induced by $\{t\}$. We can actually show more broadly that the flow across any s-t cut is given by the value of the flow.

Definition 24 (s-t cut)**.** A set $S \subseteq V$ is called a s-t cut if $s \in S$ and $t \notin S$.

Recall that for any $S \subseteq V$ we let $\partial(S) = \{(a, b) \in E | a \in S, b \notin S\}$ i.e. the set of edges pointed out of the cut.

Lemma 25. *For any s-t cut S and s-t flow f we have $\sum_{e \in \partial(S)} f(e) - \sum_{e \in \partial(V \setminus S)} f(e) = v(f)$.*

Proof. Consider $\sum_{u \in S} \mathrm{im}(f, u)$. We have $\sum_{u \in S} \mathrm{im}(f, u) = \mathrm{im}(f, s)$. Furthermore, we have that any edge with both endpoints in S appears once with each sign in $\sum_{u \in S} \mathrm{im}(f, u)$ and thus $\sum_{u \in S} \mathrm{im}(f, u) = \sum_{e \in \partial(S)} f(e) - \sum_{e \in \partial(V \setminus S)} f(e)$. $\square$

This lemma essentially says that the net flow across a cut is equal to the value of the flow if the flow is feasible. This gives us a natural way to upper bound the value of the maximum flow. We can bound it by the total capacity of a cut.

Lemma 26. *For all feasible s-t flows f and s-t cuts S we have $v(f) \leq \sum_{e \in \partial(S)} u_e$.*

Proof. By the previous lemma we have $v(f) = \sum_{e \in \partial(S)} f_e - \sum_{e \in \partial(V \setminus S)} f_e$. However since $f_e \in [0, u_e]$ we have $\sum_{e \in \partial(S)} f_e - \sum_{e \in \partial(V \setminus S)} f_e \leq \sum_{e \in \partial(S)} u_e$ as desired. $\square$

We call $\sum_{e \in \partial(S)} u_e$ the capacity, weight, or value of the cut. What the previous lemma shows is that the value of the maximum flow is at most the value of the minimum capacity cut. We will eventually show that this inequality goes in both directions that the value of the maximum flow is equal to the value of the minimum cut. However, we will show that constructively, by providing an algorithm to find the maximum flow.

To start designing a maximum flow algorithm, let's first consider the simplest type of s-t flow one might imagine. Note that any simple s-t path $e_1, ..., e_k$ easily yields an s-t flow. If we simply let $f(e_i) = \alpha$ for all e_i and 0 otherwise then so long as $\alpha \leq \min_{i \in [k]} u_{e_i}$ then this flow is feasible. Moreover, note that $v(f) = 0$ if and only if there is no s-t path (as there is a s-t cut of value 0 if s cannot reach t). Consequently, we can always find some flow to send by looking for a s-t path. The question now is how to turn that fact into an efficient algorithm. Simply greedily sending flow along s-t paths and deleting those edges will not work, as we will see. However, if we more carefully find a way to recurse when sending flows, then we can make this algorithm work as we will see next lecture.