Chapter 13

1 Section 13.1: Basic Notation and Terminology

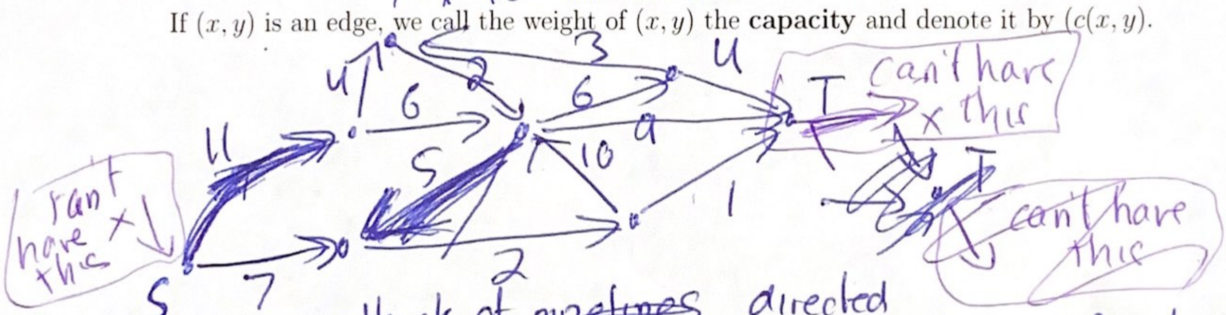
Definition 1.1. In this section we will be working with a **network** N. This is a directed graph with the following additional two properties:

can't have this

- Between any two vertices there is at most one directed edge.
- There are two special vertices, a **source** S, that only has directed edges coming out of it, and a **sink** T, that only has edges going into it.

T fa terminus

If (x, y) is an edge, we call the weight of (x, y) the **capacity** and denote it by $c(x, y)$.



- think of ~~pipelines~~ directed edges as pipelines, transferring fluid
- capacity is max volume of fluid pipeline can handle

Definition 1.2. A **network flow** ϕ, or just **flow** for short, is a function that assigns to each edge in a network a positive number such that the following is upheld:

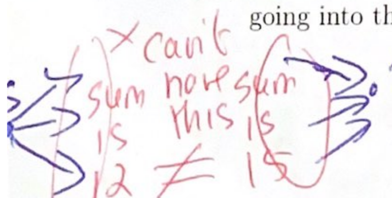
can't have this

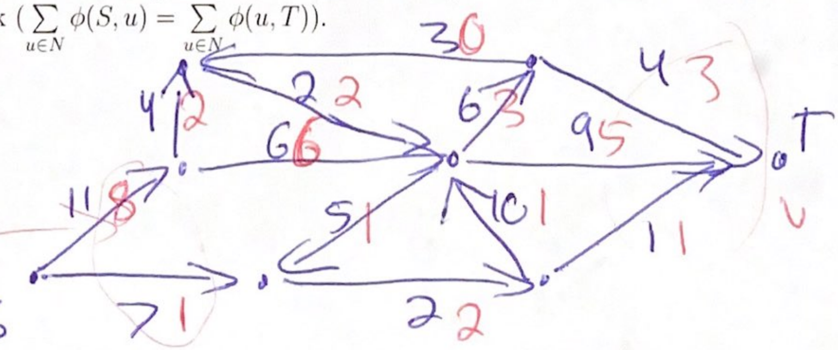
(i) The flow assigned to an edge can't exceed the capacity of that edge ($\phi(x, y) \leq c(x, y)$).

can't have this

(ii) At any vertex v that is not the source or the sink, the total amount of flow coming in to v must equal the total amount of flow leaving v ($\sum_{u \in N} \phi(u, v) = \sum_{u \in N} \phi(v, u)$).

(iii) The total amount of flow coming out of the source must equal the total amount of flow going into the sink ($\sum_{u \in N} \phi(S, u) = \sum_{u \in N} \phi(u, T)$).





value of this flow is 9

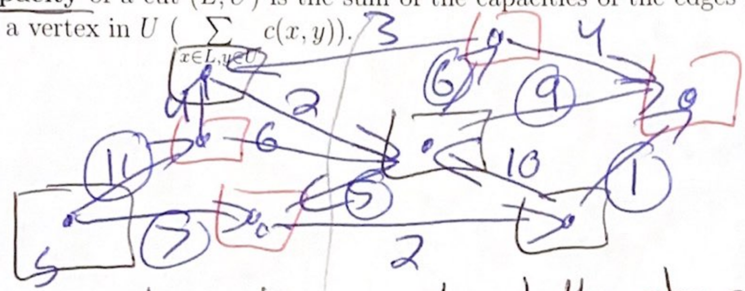
2 Section 13.2: Flows and Cuts

Definition 2.1. If N is a network, a **cut** (L, U) is a partition of the vertices of N into two sets L and U such that $S \in L$ and $T \in U$.

The **capacity** of a cut (L, U) is the sum of the capacities of the edges going from a vertex in L to a vertex in U ($\sum_{x \in L, y \in U} c(x, y)$).

L

U



to get capacity of the above cut, sum capacities of all edges from □ to □

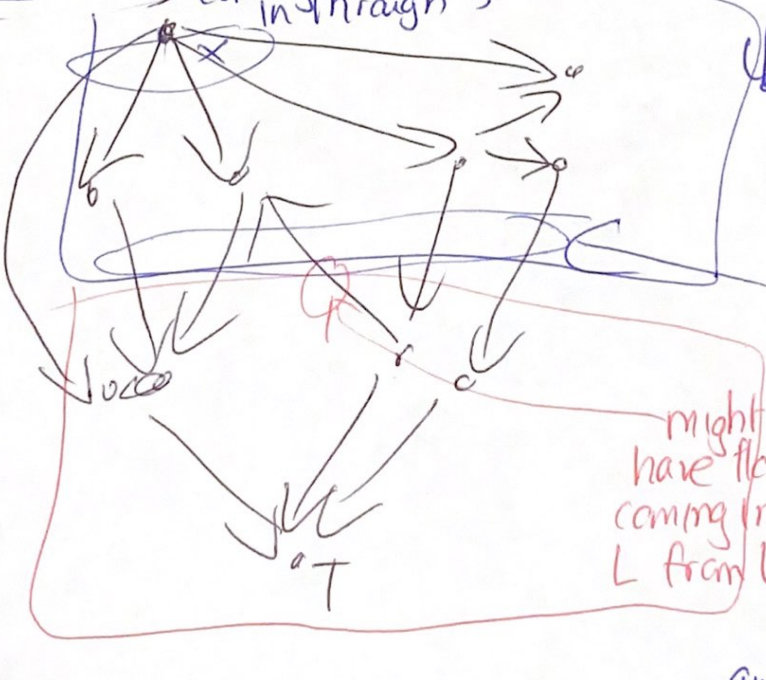
11+7+5+~~[illegible]~~9+6+1= ~~49~~ 39 ≤ capacity of

Theorem 2.2 (Max Flow ≤ Min Cut). *If N is a network, the largest value of a flow on N is less than or equal to the smallest capacity of a cut of N.*

in fact, max flow = min cut

~~we will see~~ this later

the value of flow is X



L input into L must equal output from L to maintain equilibrium

U amount of flow leaving L must equal flow entering L

amount of flow leaving L is bounded by capacity of cut

$X \le$ amount of flow going from L to U $\le$ capacity of cut (L, U)

3 Section 13.3: Augmenting Paths

Given a network N, how do we find a maximum flow on N?

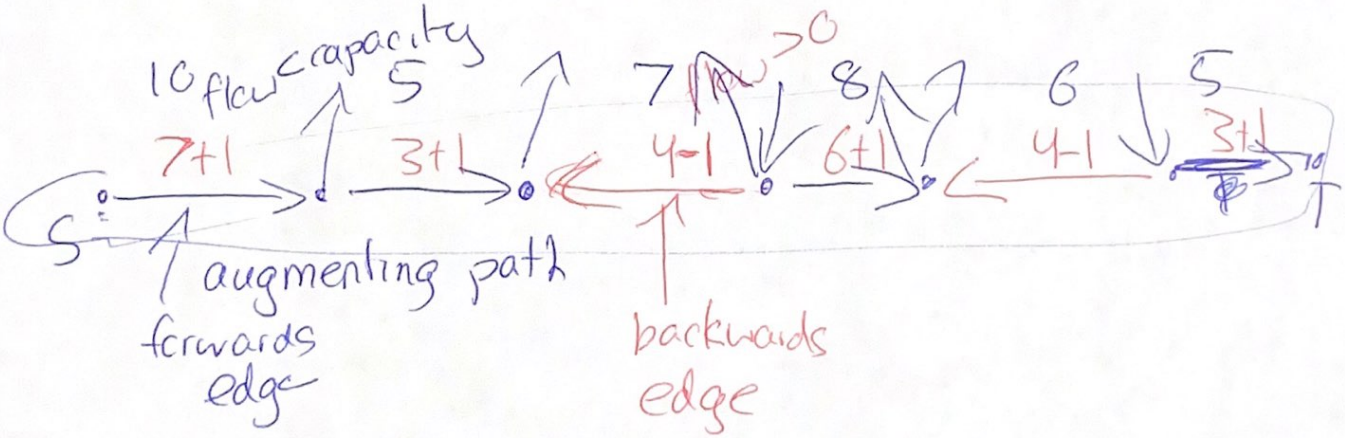
Definition 3.1. If we have a flow ϕ, an **augmenting path** is a sequence of vertices $S = v_1, v_2, \ldots, v_{n-1}, v_n = T$ such that for each v_i and $vi + 1$, there is either

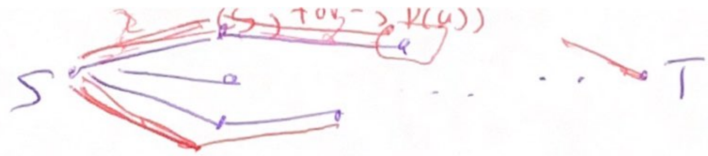
- There is a directed edge from v_i to v_{i+1} for which the flow is less than the capacity ($\phi(v_i, v_{i+1}) < c(v_i, v_{i+1})$), or
- There is a directed edge from v_{i+1} to v_i on which the flow is nonzero ($\phi(v_{i+1}, v_i) > 0$).

Say that if the directed edge goes from v_i to v_{i+1} then it is a **forwards edge**, and if it is from v_{i+1} to v_i then it is a **backwards edge**.

$$\text{Set } \delta = \min(\{c(v_i, v_{i+1}) - \phi(v_i, v_{i+1}) \mid (v_i, v_{i+1}) \text{ a forwards edge}\} \cup \{\phi(v_{i+1}, v_i) \mid (v_{i+1}, v_i) \text{ a backwards edge}\}).$$

Then we can create a new flow by increasing the flow in each forwards edge by δ, and decreasing the flow in each backwards edge by δ.





4 Section 13.4: The Ford-Fulkerson Algorithm

Overall Idea: Starting at S, we build augmenting paths across the network N in a systematic, vertex-by-vertex way.

- When we hit T we have our augmenting path, so we update and start all over.
- If we get stuck, and can't get an augmenting path, then we have reached a maximum flow and are finished.

• Starting from S, build a "scanning queue" of vertices. We **scan** from a vertex u and **label** any potential vertex v. The idea is that u represents the end of an augmenting path "under construction" and v is the next vertex on that path.

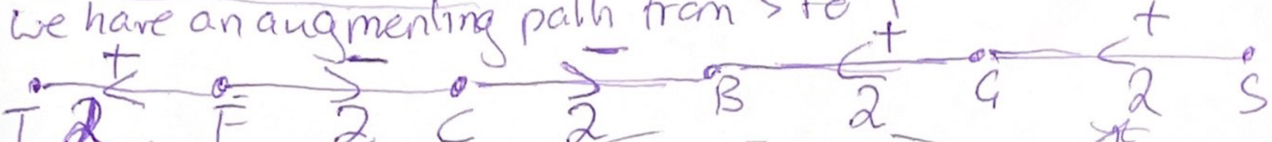
How to scan from u?

- Look for a vertex v adjacent to u that has not already been labeled and added to the queue. (Skip over v if it has.)
- Check to make sure the edge between u and v is not "full" (if $u \to v$) or "empty" (if $u \leftarrow v$). Ignore v otherwise.
- Label v with $(u, + \text{ or } -, p(v))$, where $p(v)$ is the smaller of $p(u)$ and the available capacity on the edge between u and v.
- Add v to the queue.

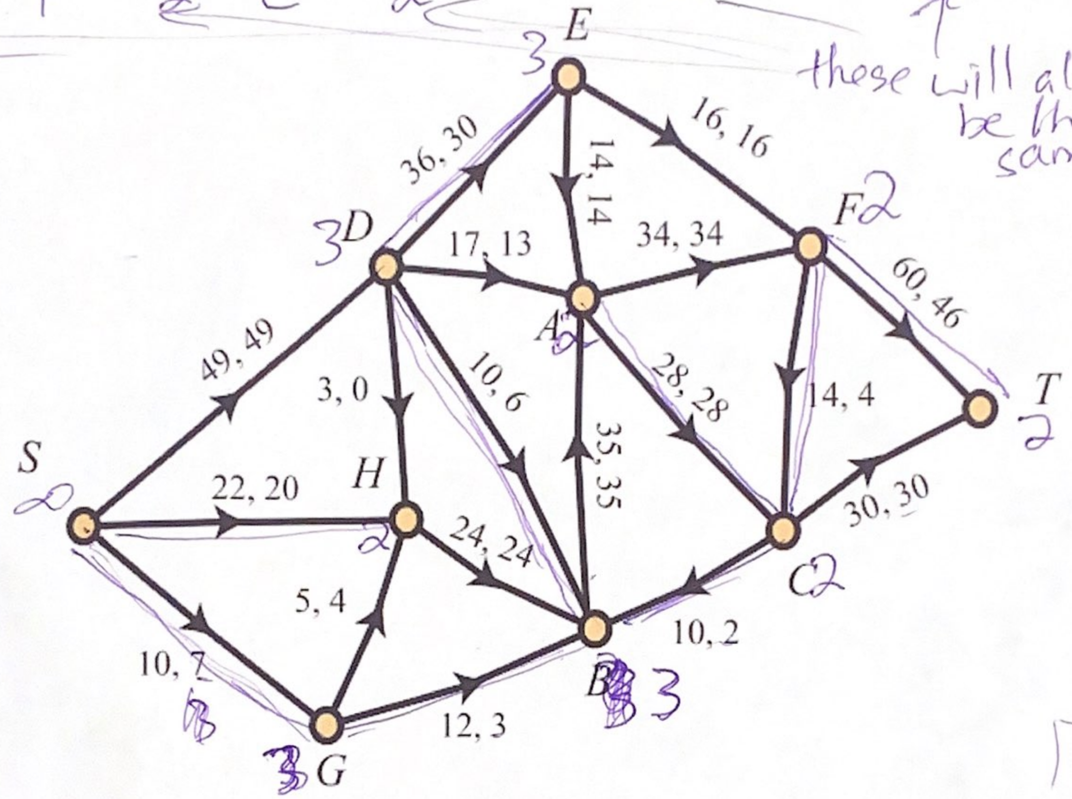
What if when we scan from u, find multiple vertices to label and add? Is there an order to add them to the queue? Yes, based on lexicographic order.

From our work below:

we have an augmenting path from S to T



these will always be the same



this figure

Exercise 4.1 (Exercise 13.9 from the textbook). Run an update step of the Ford-Fulkerson Algorithm for the network flow given in Figure ??.

scanning from S

skip over D because edge to D is full

(G , ↑ , ↑)

prior vertex in path, forwards or backwards, potential

scanning from G

skip over H and S because they were already labeled and added to our queue

scanning from D

skip over D because edge from D is empty

Scanning Queue

✓S (*, +, ∞)

✓G (S, +, 3)

✓H (S, +, 2)

✓B (G, +, 3)

✓C (B, −, 2)

✓D (B, −, 3)

✓A (C, −, 2)

✓F (C, −, 2)

E (D, +, 3)

T (F, +, 2)

stop, T is in the queue