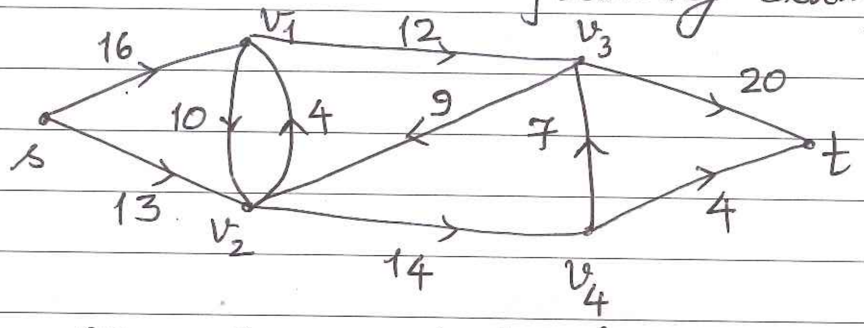
Lecture 7

Max-flow algorithms : we saw Ford-Fulkerson algorithm in the last lecture.

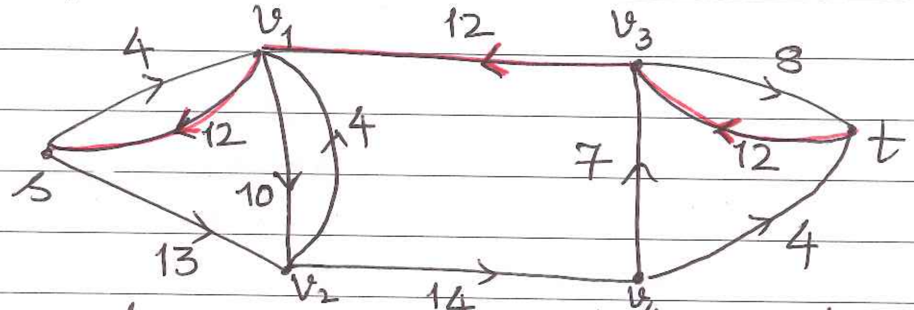
Let us run it on the following example.



The flow f is initialized to $f(e)=0$ $\forall\, e \in E$.
So $G_f = G$ at the beginning.

- Now find an s-t path in G.
  Say we found $s - v_1 - v_3 - t$.
  - we can send 12 units along this path.
  - update f.

G_f now :



- the reverse edges e'' in G_f are highlighted in red.

- Now we find another s-t path in G_f.
  Say we found $s - v_2 - v_4 - t$
  - we send 4 units along this path.
  - update f and G_f.

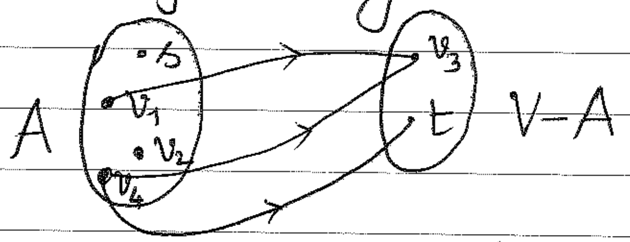
- In the updated G_f, we find another s-t path $s - v_2 - v_4 - v_3 - t$ and send 7 units along this path.
  - update f and G_f

It is easy to check that there is no s-t path in this residual graph G_f.

- The claim is f is a max-flow.

The proof that f is a max-flow will use the fact that there is an s-t cut that is saturated by f.

Consider the following s-t cut in the above example:



Here we have drawn only the forward edges in this cut: these are the edges e with start$(e) \in A$ and end$(e) \in V-A$.

- note that our example has only 3 such edges (v_1, v_3), (v_4, v_3), (v_4, t).
- the sum of their capacities is $12 + 7 + 4 = 23$.
- so this is the bottleneck between s and t in this graph and we cannot send more than 23 units of flow from s to t.

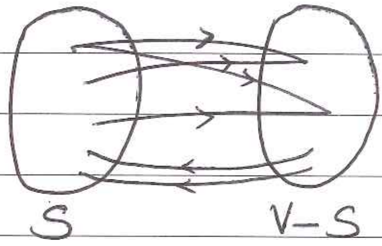
- observe that value$(f) = 23$ for the flow f that our example constructed. Thus f is a max-flow.

We will now formally prove all these points in any given graph G and prove the correctness of Ford-Fulkerson algorithm.

The following claim is important here.

__Claim.__ For any flow f and s-t cut $C = (S, V-S)$, we have value$(f) \leq$ capacity(C)

__Proof.__



This is the set of edges in G with one endpoint in S and another endpoint in $V-S$.

Capacity of this cut is

$$\sum_{\substack{e:\ \text{start}(e) \in S \\ \text{and end}(e) \in V-S}} c(e)$$

⤳ this is the sum of capacities of forward arcs, the edges directed from S to $V-S$.

Recall that value(f) = $-$excess(s)

$$= -\sum_{u \in S} \text{excess}(u)$$

[we are adding some terms here, each of these is 0]

$$= \sum_{e \in \underbrace{S \times (V-S)}_{\text{edges directed from } S \text{ to } V-S}} f(e) \;-\; \sum_{e \in \underbrace{(V-S) \times S}_{\text{edges directed from } V-S \text{ to } S}} f(e)$$

$$\leq \sum_{e \in S \times (V-S)} c(e) \;-\; 0$$

[we are using the condition that $0 \leq f(e) \leq c(e)$ for all e]

$$= \text{capacity}(C).$$

$\square$

Our next theorem which is Max flow - Min cut theorem shows that if f is a max flow then there exists an s-t cut $(S, V-S)$ such that value(f) = capacity$(S, V-S)$.

Max flow - Min cut theorem.

The following statements are equivalent.

(1) f is a max flow.
(2) there is no s-t path in G_f.
(3) there is an s-t cut $C = (S, V-S)$ such that capacity (C) = value (f).

<u>Proof</u>. We will show that $(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (1)$.

We have already shown that "if t is reachable from s then f is not a max-flow" (see the claim in previous lecture notes).
Thus if f is a max-flow then there is no s-t path in G_f. Hence $(1) \Rightarrow (2)$.

We will now show $(2) \Rightarrow (3)$. Let S be the set of vertices reachable from s in G_f. Since there is no s-t path in G_f, $(S, V-S)$ is an s-t cut.

$$\text{value}(f) = -\text{excess}(s) = -\sum_{u \in S} \text{excess}(u)$$

$$= \sum_{e \in S \times (V-S)} f(e) - \sum_{e \in (V-S) \times S} f(e)$$

$$= \sum_{e \in S \times (V-S)} c(e) - 0$$

$$= \text{capacity of the cut } (S, V-S).$$

Thus we have proved $(2) \Rightarrow (3)$.

Exercise. Why is $f(e) = c(e)$ for all e in $S \times (V-S)$ and $f(e) = 0$ for all e in $(V-S) \times S$?

Finally $(3) \Rightarrow (1)$ by the claim we proved in the previous page. □
(Please check this.)

This theorem immediately proves the correctness of Ford-Fulkerson algorithm.

This is because when this algorithm terminates there is no s-t path in G_f. That is, (2) holds. We know (2) $\Rightarrow$ (1), i.e. f is a max-flow.

Question: Does Ford-Fulkerson algorithm always terminate?

- Let us assume all edge capacities are integers. Then in every iteration the value of f increases by at least 1.

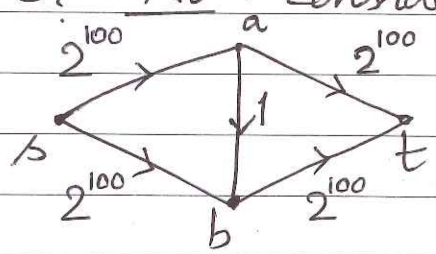
  So the number of iterations of the repeat-loop $\leq$ value of max-flow.

  Let C_{max} be the largest edge capacity.

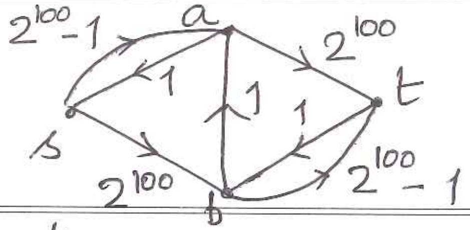
Exercise. Show that the running time of Ford-Fulkerson algorithm is $O(m \cdot n \cdot C_{max})$ where m is the number of edges and n is the number of vertices in G.

Note that this is under the assumption that edge capacities are integers. Even in this case, this is not a polynomial time algorithm since our running time involves "C_{max}". To represent C_{max}, we need $\log(C_{max})$ bits — so the input size is $poly(m, n, \log(C_{max}))$.

Is this a pessimistic estimate of the running time? No : consider the following example.

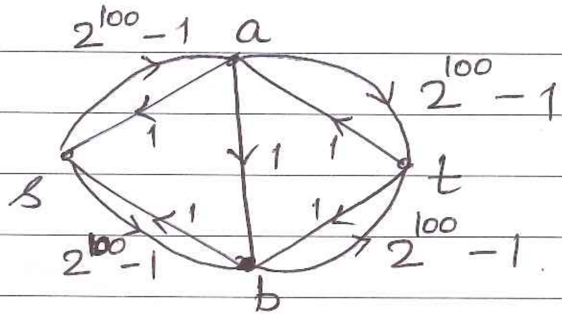


Suppose the algo. finds the s-t path $s-a-b-t$ in the first iteration. Only 1 unit of flow can be sent along this path.

Then G_f becomes 

Let us choose the path $s - b - a - t$ now & we can again send only 1 unit of flow along this path.

Now G_f becomes

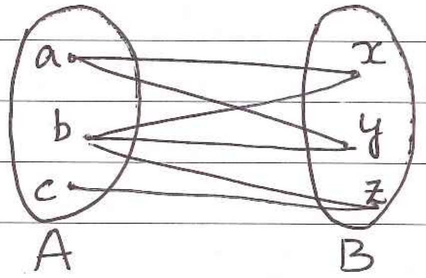


We again find the path $s - a - b - t$ and send 1 unit of flow along this path and so on.

The max flow value here is $2 \cdot 2^{100}$ and the above choice of paths leads to 2^{101} iterations of the repeat-loop.

Bipartite Matchings

The input here is a bipartite graph $G = (A \cup B, E)$. This means the vertex set can be partitioned into 2 sets A, B such that every edge has one endpoint in A and the other endpoint in B.



A matching M is a subset of E such that no 2 edges in M share a common endpoint.

We seek a max-size matching in the input graph G. For example, $M = \{(a,x), (b,y), (c,z)\}$ is a max-size matching in the above graph.

- A max-size matching is useful in several applications. Say, A is a set of students and B is a set of projects and an edge (a, x) means student a is interested in project x.