COP 3503 10/27/26

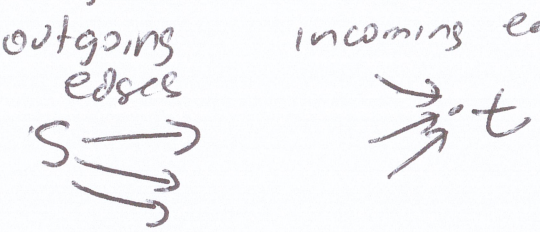
Network Flow

Input Graph Directed s t

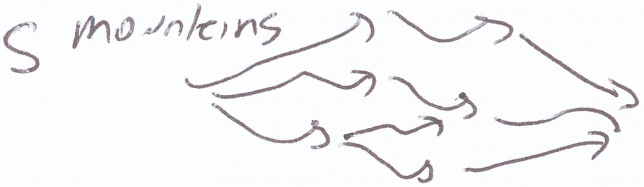
2 special vertices: SOURCE, SINK

only has outgoing edges

only has incoming edge



River system



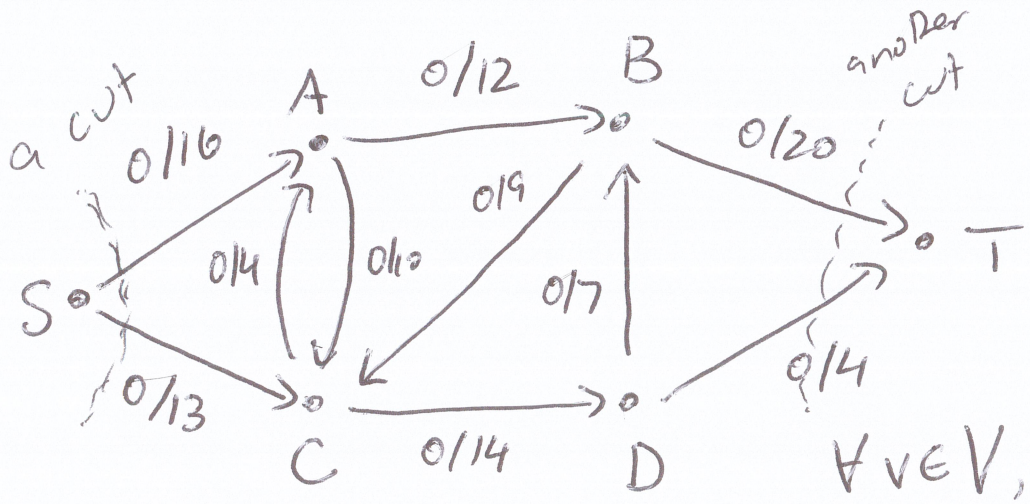
flow through edge is sort of a steady state amount of fluid/material goes through in a unit of time.

Goal: Determine Maximum Flow we can send through the system.

0/10 → directed edge

associated value = CAPACITY

Initially we would set all FLOWS = 0



for any vertex, we can calculate sum of incoming flow + sum of outgoing flow

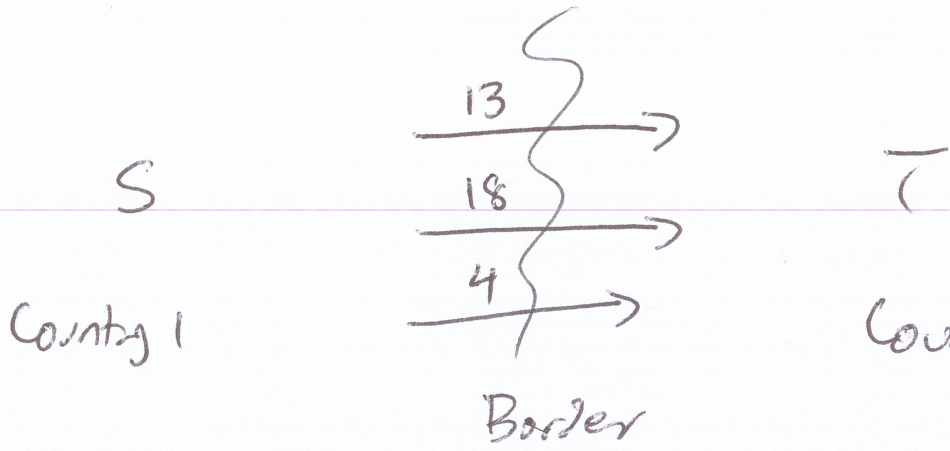
$\forall v \in V, v \neq S, v \neq T$

Sum incoming = Sum outgoing

Sum out of S = Sum into T

Observation about maximum flow:

≤ 29 since $S \xrightarrow{16} A + S \xrightarrow{13} C = 29$



Cut: Partition of vertices into 2 groups one with S, the other with T. (Set A, Set B)

Take sum of the edges (capacities) that connect a vertex from A to a vertex from B, and this is the VALUE of that cut.

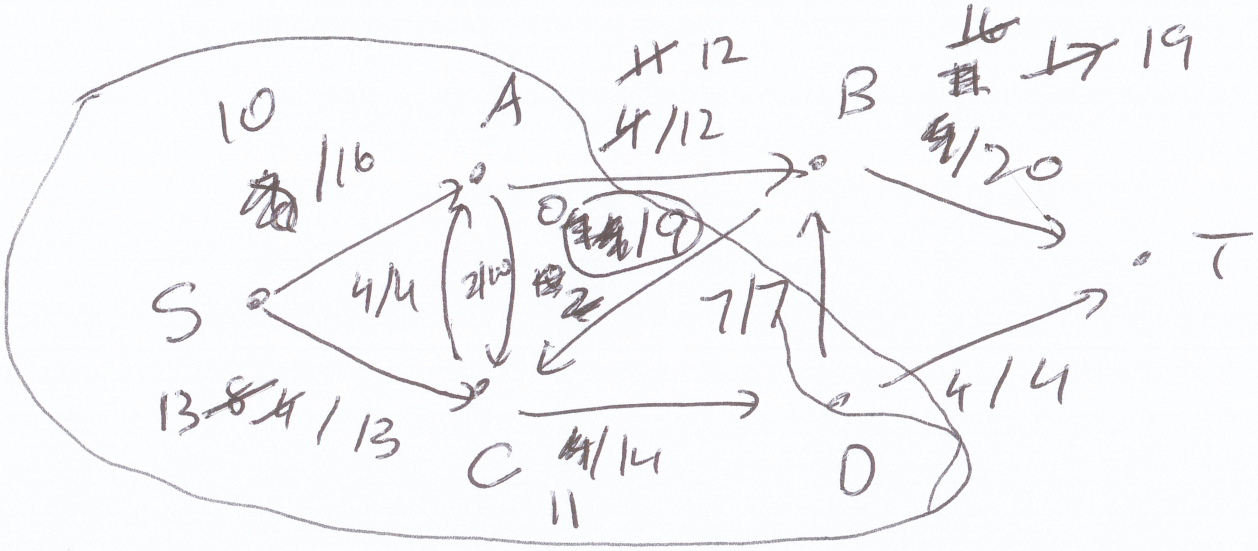
Minimum Cut: Of all valid partitions of the vertices the ~~maximum~~ minimum sum of edges across the cut.

Max-Flow Min-Cut Theorem: The maximum flow through a flow network equals the minimum cut!

General Solution Idea

While we can get more flow through from S to T

Add the flow on an "augmenting path" to our current configuration.



augmenting path: starts at S, ends at T, and every edge on the path has "residual capacity"

↳ find min. residual capacity + send it through

A 5/12 B → from A→B residual cap = 7

← residual capacity from B to A is 5

"back edge"

NO PATH CUT SACD

A→B 12
D→B 7
D→T+4
23

S→C→D→T 4

S→C→A→B→~~D→B~~→T 4

S→A→B→C→D→B→T 7

S→C→B→T 5

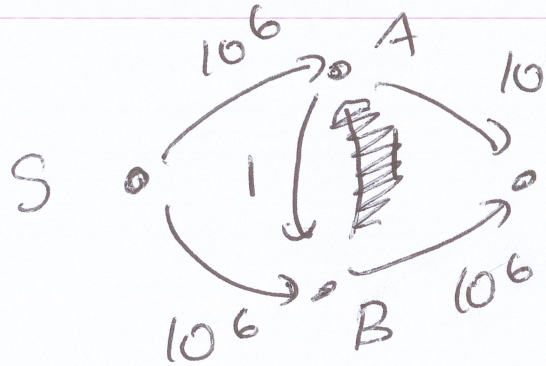
S→A→B→T 1

S→A→C→B→T 2

Ford-Fulkerson Algs

Use DFS to find augmenting paths

not polynomial in size of graph



$\begin{cases} S \to A \to B \to T & 1 \\ S \to B \to A \to T & 1 \\ \& \text{ repeat } 2 \times 10^6 \text{ times} \end{cases}$

Edmunds-Karp Algs

Use BFS ... this is proven to run in polynomial time of the input size.

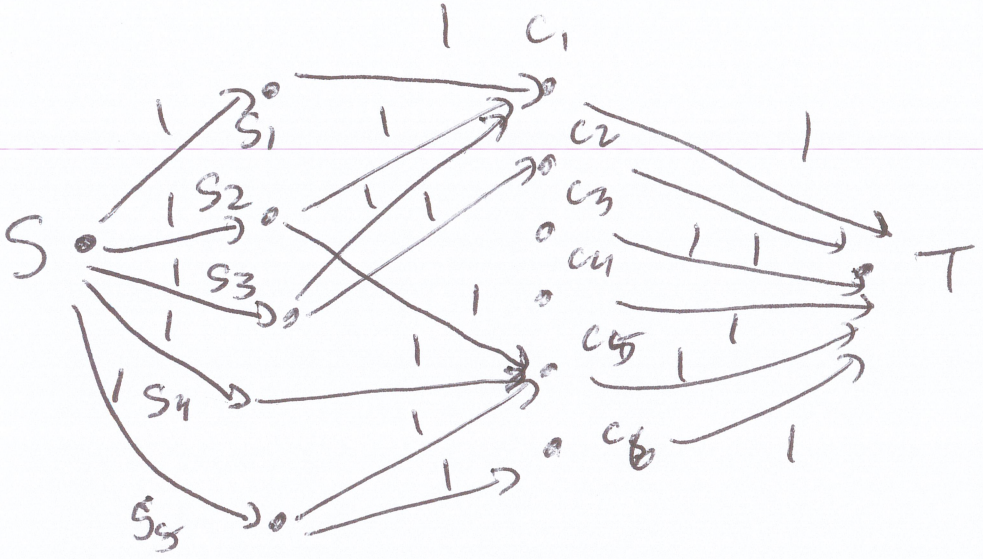
Dinitz / Dinic ✓ (real spelling ✓) a lot people use

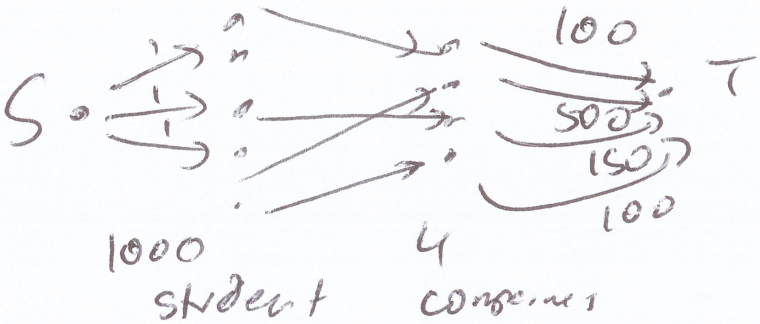
While aug paths:

1. Run ONE BFS store its length
2. Keep running DFS capped at the length above.

Bipartite Matchings

Set A
Set B
Goal is to create maximum # pairs (a_i, b_j) where those items are allowed to "go together"

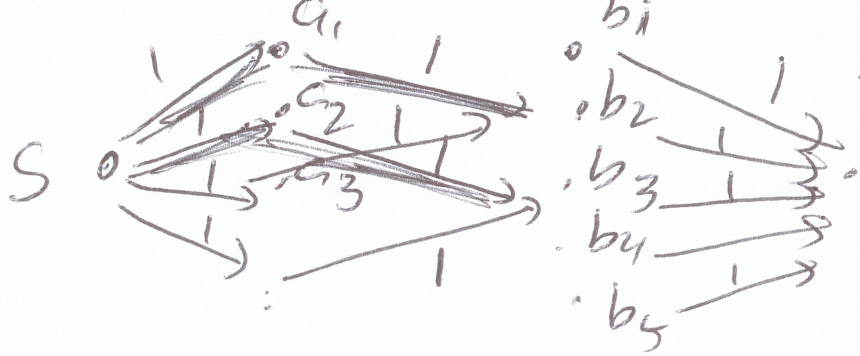




Answer is based subtracting out MAXFLOW

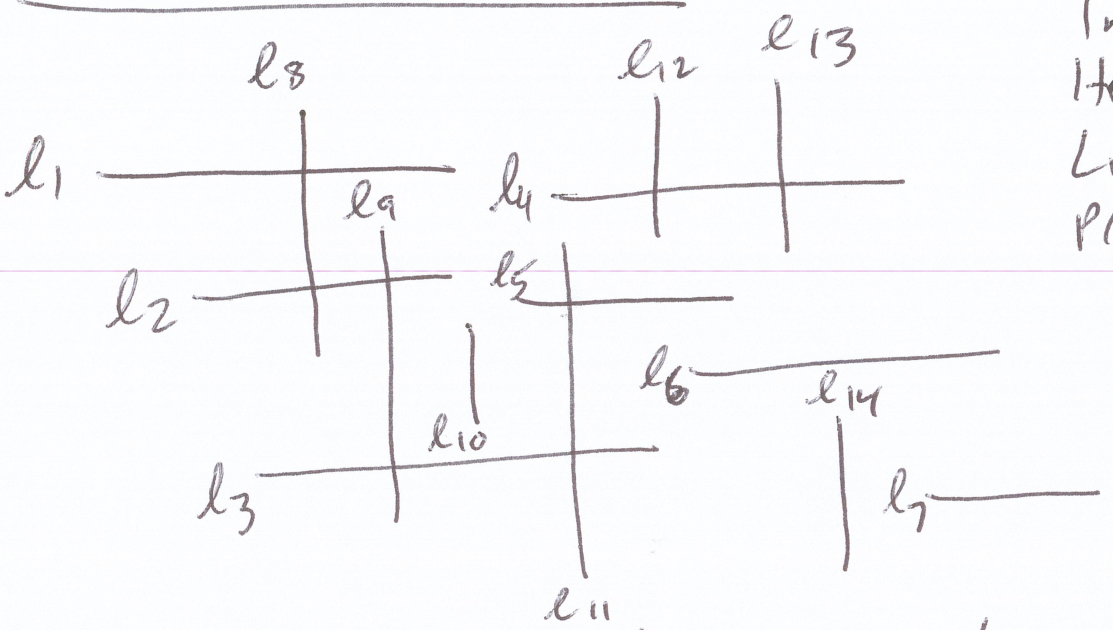
---

2 classes CLASS A CLASS B trip
some kids in class A hate kids in class B.
Max # of kids we can take without taking any pair that hate each other.



$|A| + |B| - \text{maxflow}(G)$

Cow Steeplechase



Input
Horiz + Vert
Lines → Cartesian
Plane → Segments

Output: Max # of lines segments we can keep so that no 2 in our subset intersect

