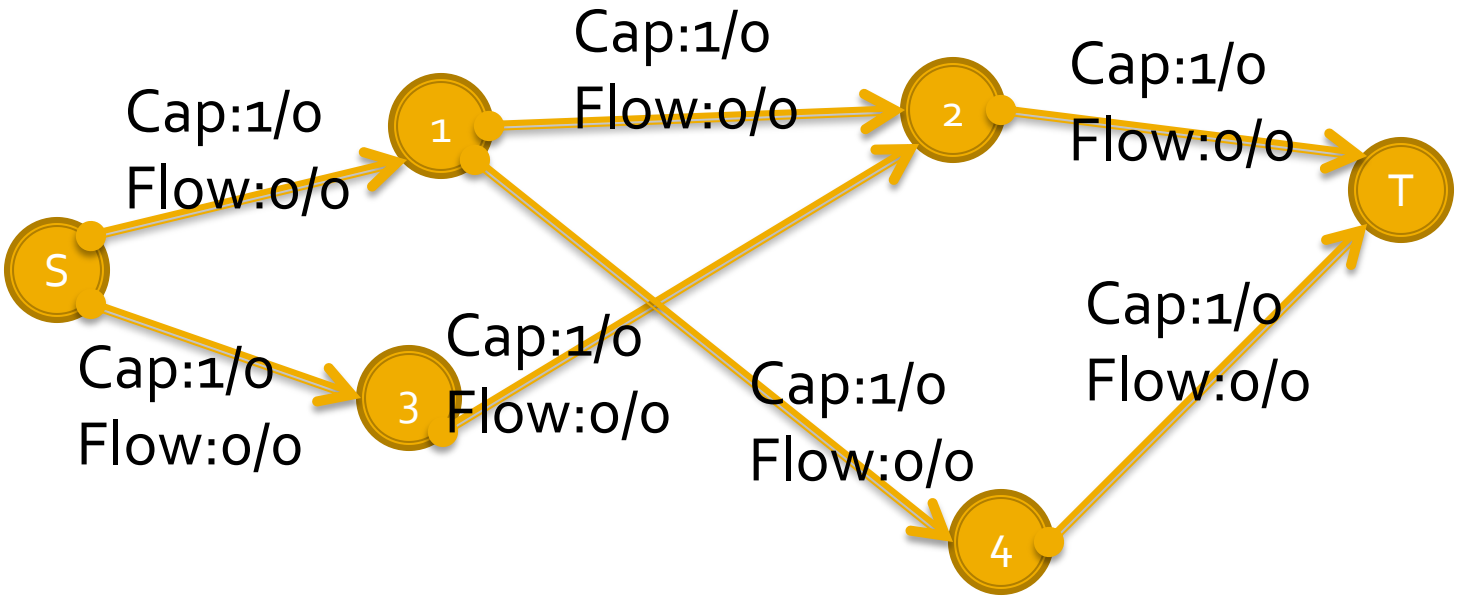
Maximum flow part 2

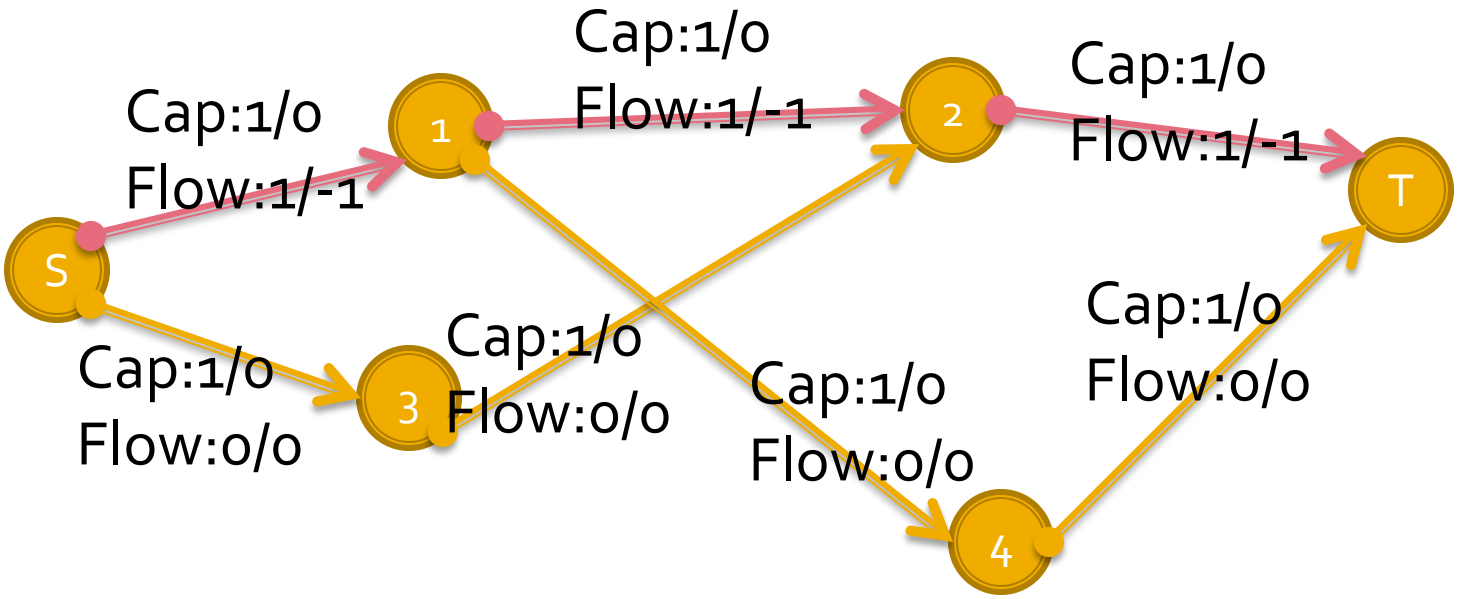
Maxflow, how to cancel?

- Consider this network



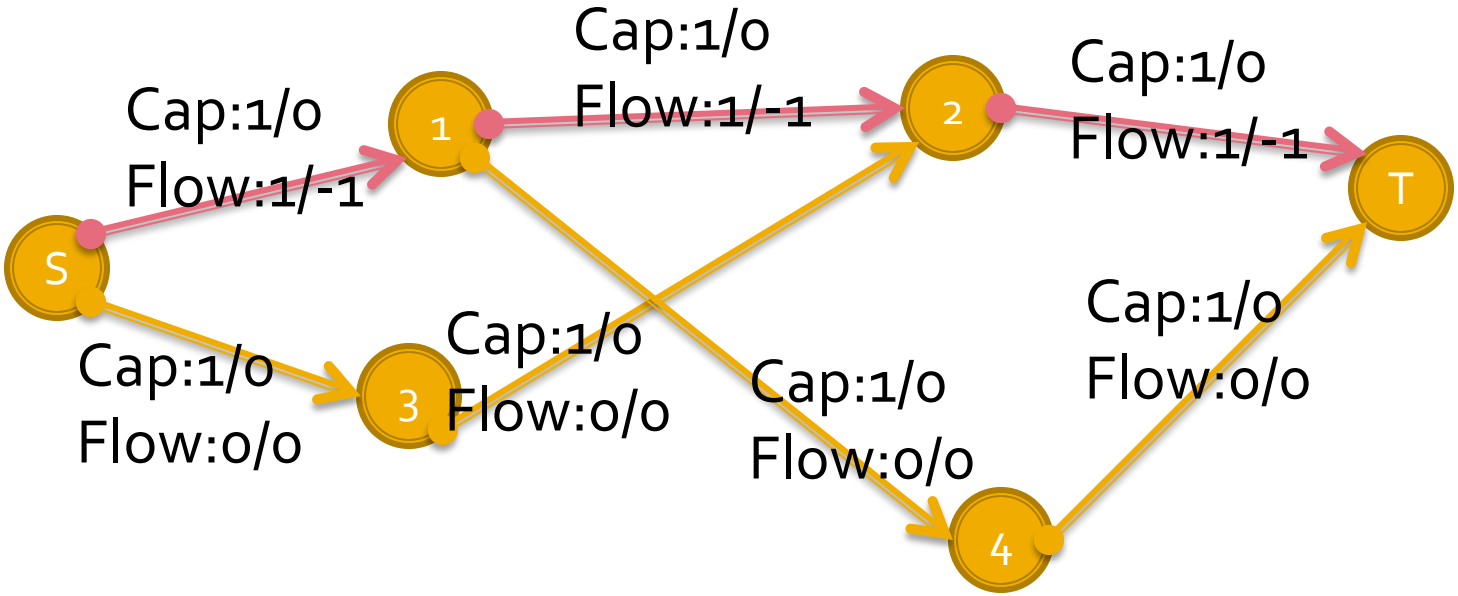
Maxflow, how to cancel?

- residue flow = cap - flow



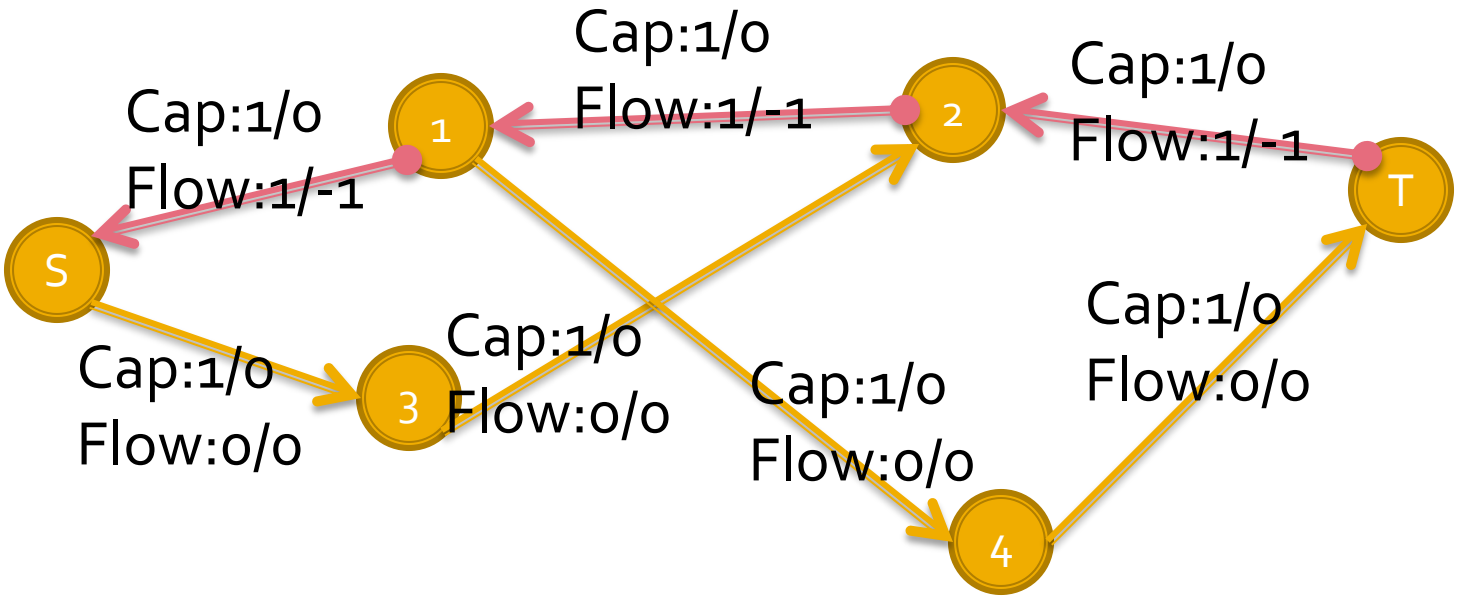
Maxflow, how to cancel?

- residue flow = cap – flow
- Now…
- residue flow: node 1 to 2 = 1-1 = 0 < can't use
- residue flow: node 2 to 1 = 0-(-1) = 1 < can use



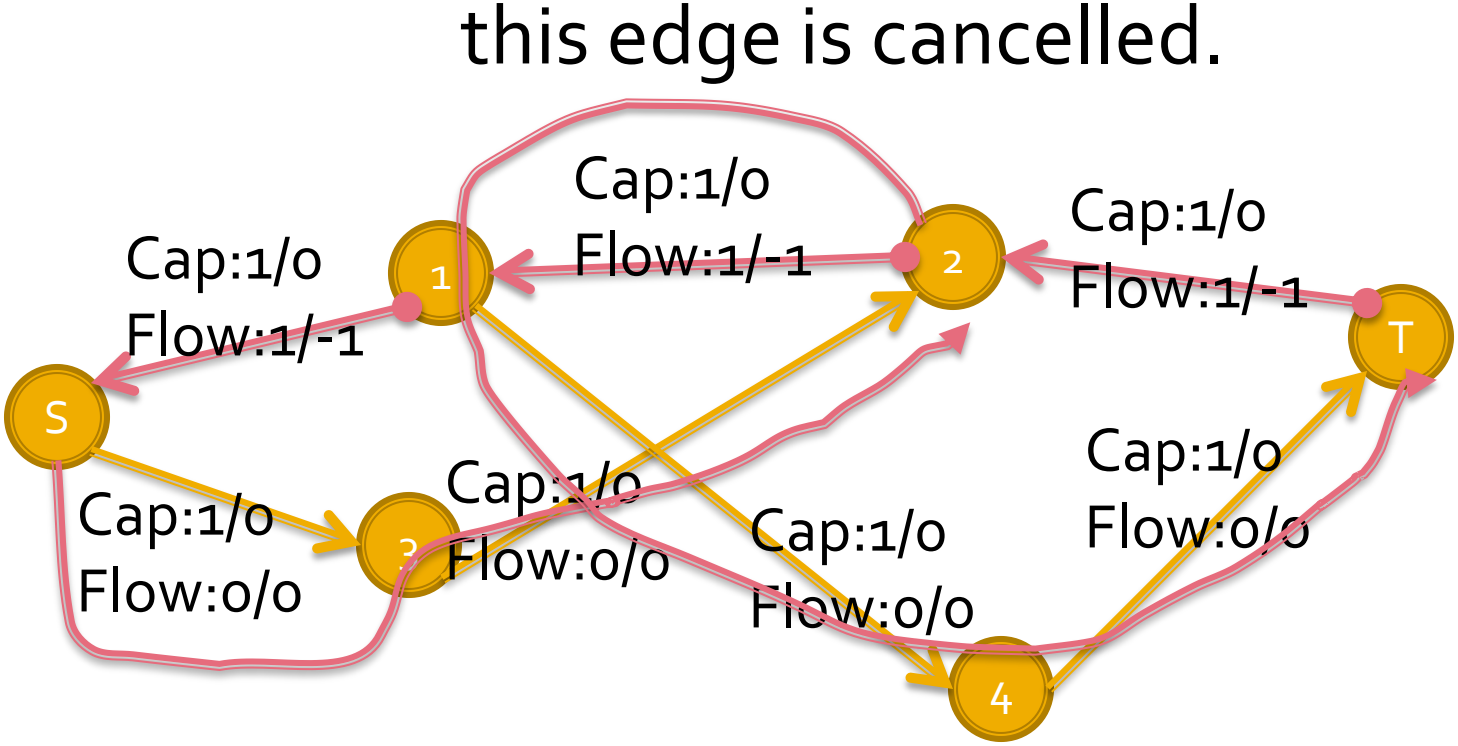
Maxflow, how to cancel?

- So, it becomes:



Maxflow, how to cancel?

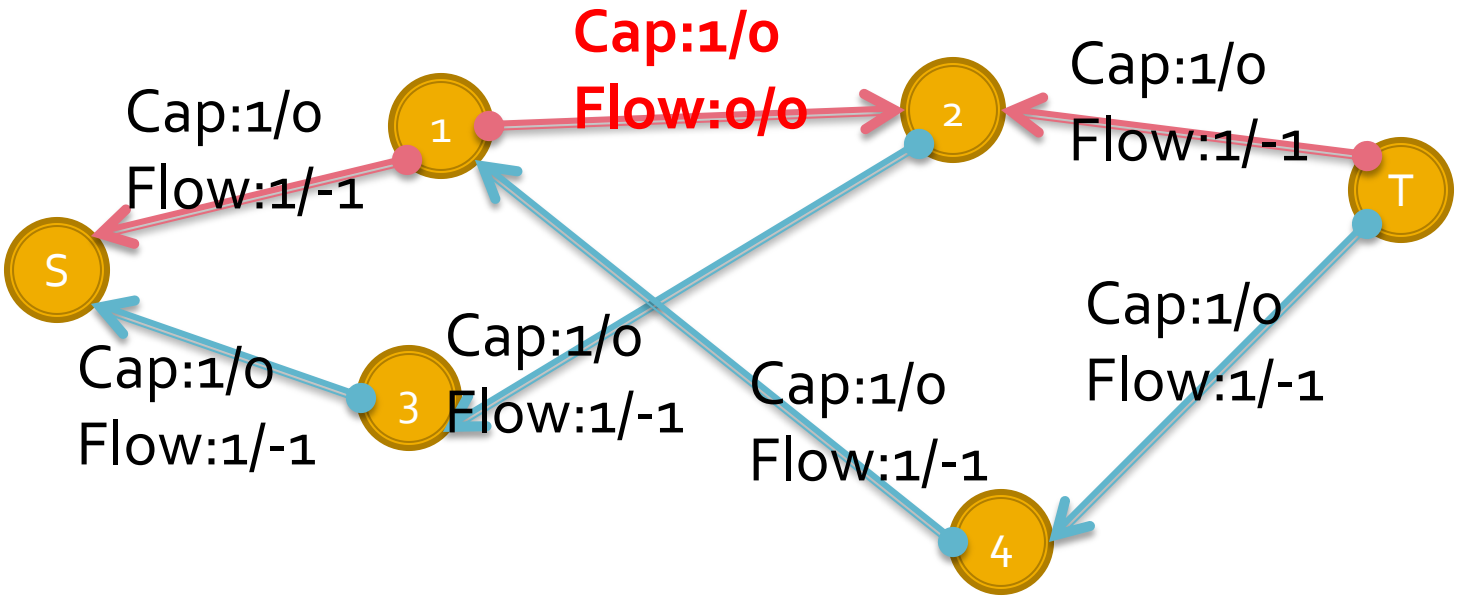
- Cancellation



Maxflow, how to cancel?

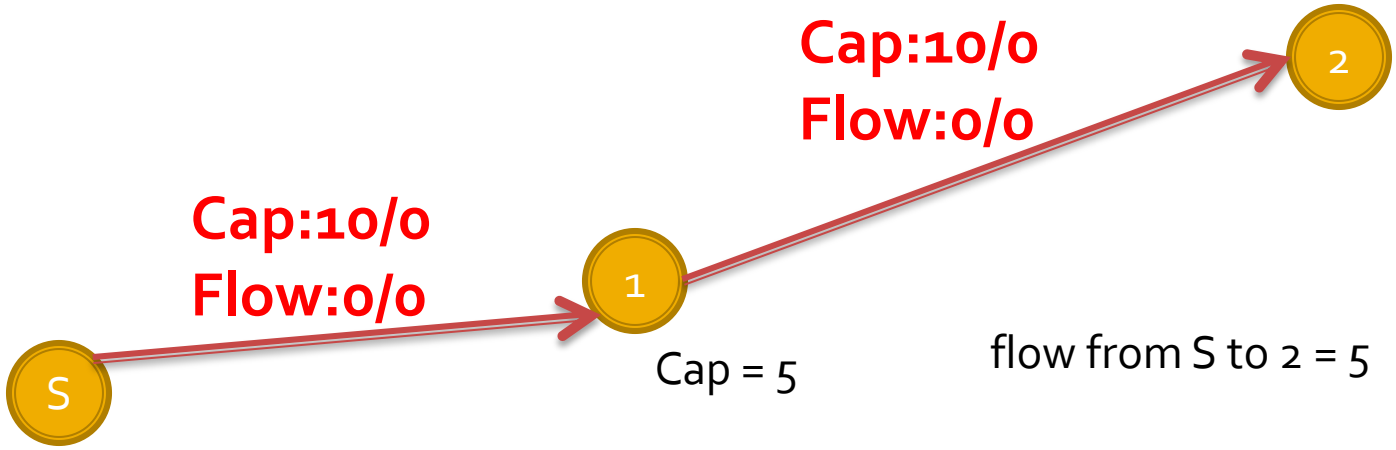
- Cancellation

this edge is cancelled.



Vertex with capacity…?

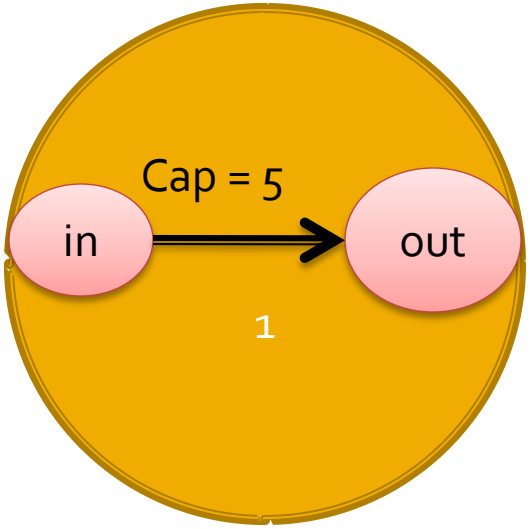
- How to handle this…?



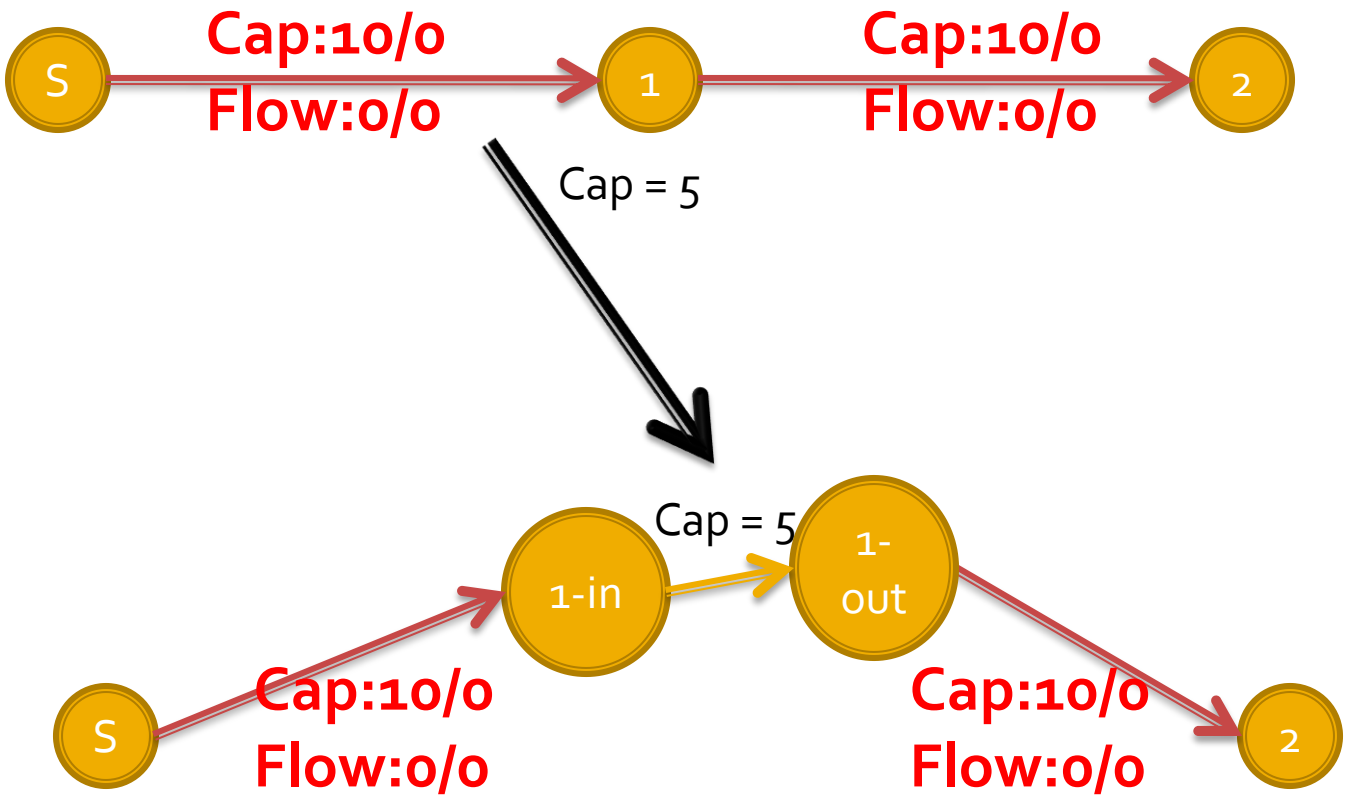
Vertex with capacity…?

Cap = 5

Transform =>

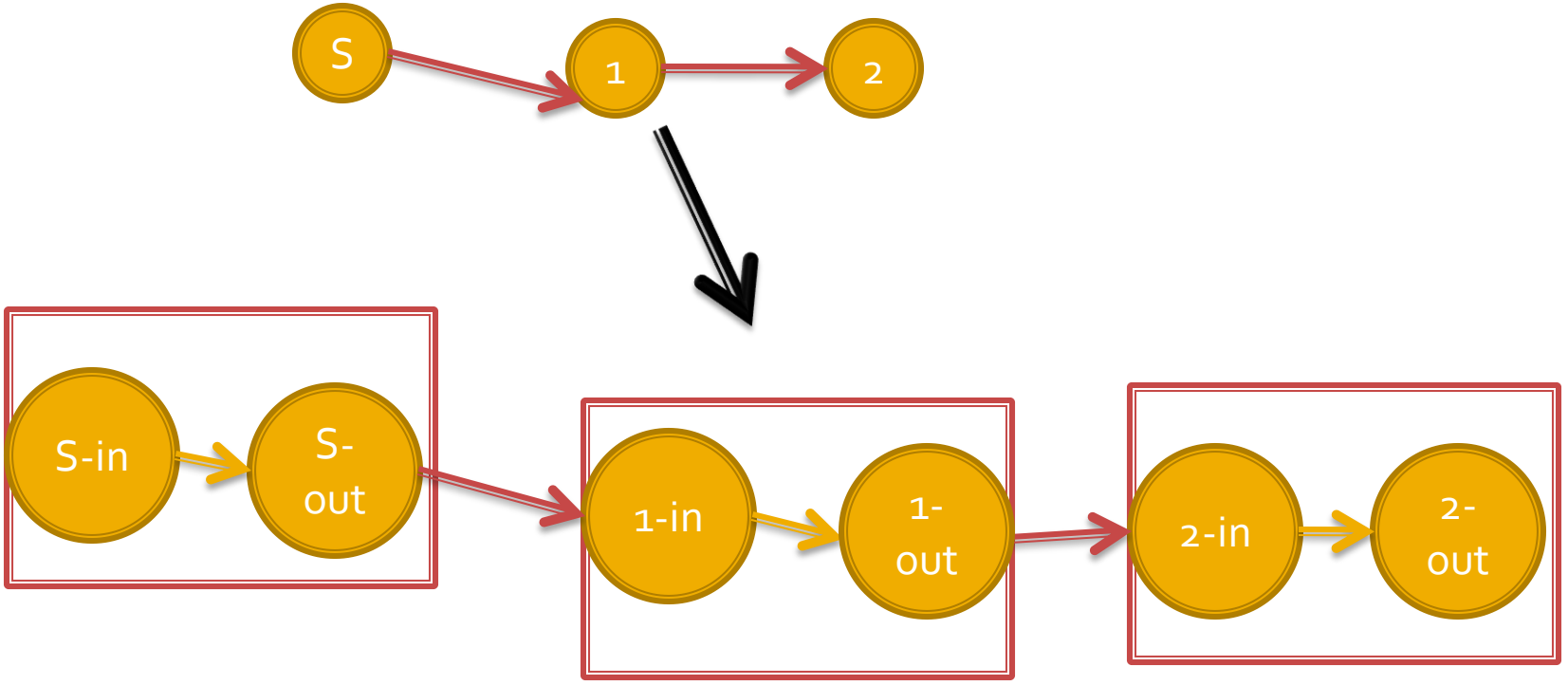


Vertex with capacity…?



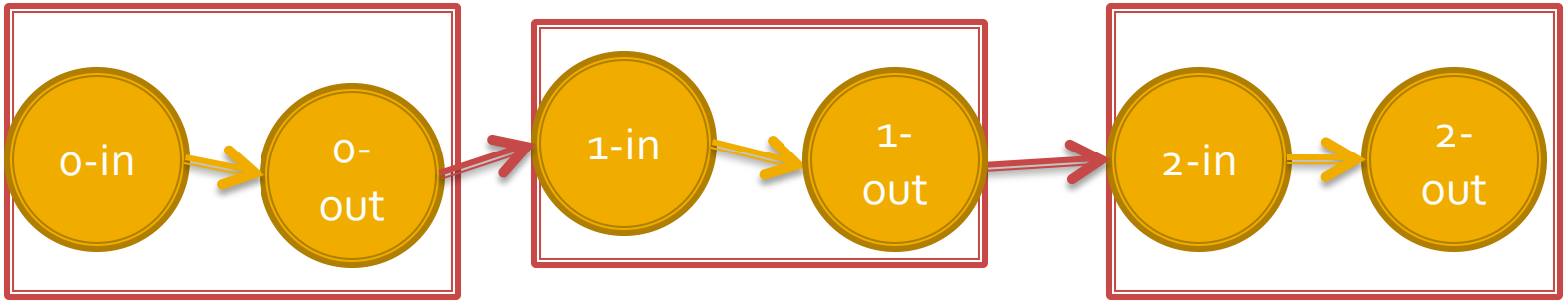
Vertex with capacity…?

- Using "split vertex", the size of graph will increase = (n*2)



Vertex with capacity…?

- How to code?



- In node: (original node)
- Out node: (original node)+n
- or
- In node: (original node)*2
- Out node: (original node)*2+1

Edge disjoint path

- Suppose you want to send k large files from s to t but never have two files use the same network link (to avoid congestion on the links).

k Edge-disjoint Paths

- Given directed graph G , and two nodes s and t, find k paths from s to t such that no two paths share an edge.

k Edge-disjoint Paths
Reduce the problem to maxflow

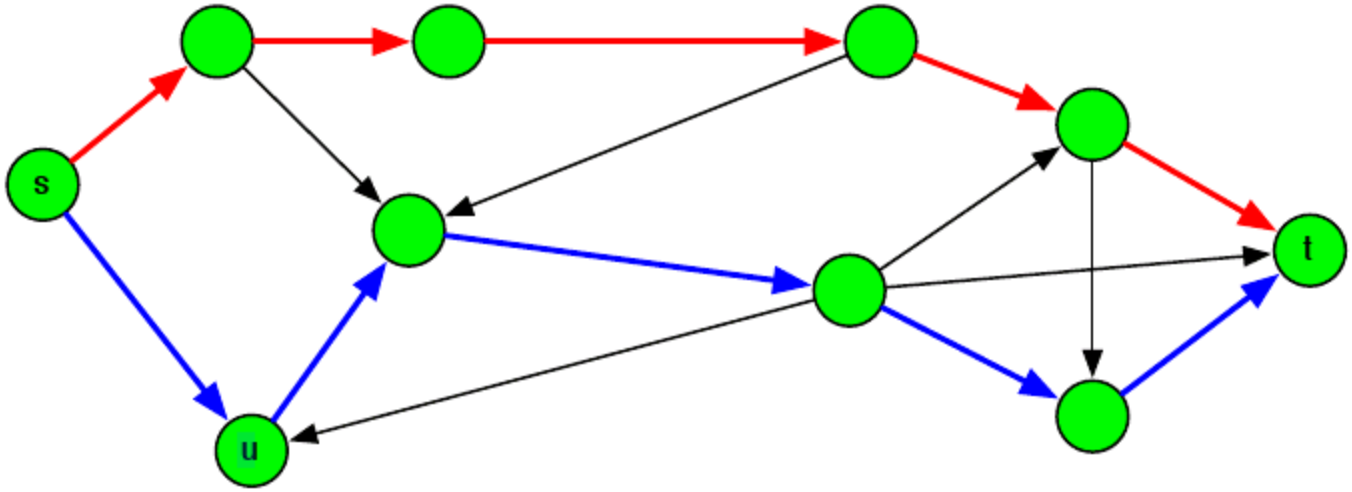
1. Given an instance of k-Edge-Disjoint Paths,
2. Create an instance of Maximum Network Flow.
3. The maximum flow will used to find the k edge disjoint paths.

k Edge-disjoint Paths
Reduce the problem to maxflow

- Suppose we had k edge-disjoint s – t paths.
- We could sent 1 unit of flow along each path without violating the capacity constraints.

- If there are k edge-disjoint s – t paths in directed, unit-weight graph G , then the maximum s – t flow is ≥ k.

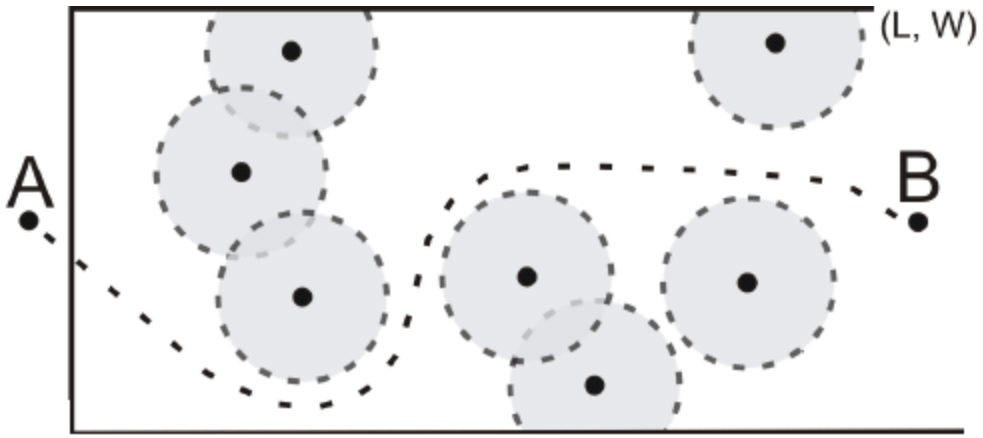
k Edge-disjoint Paths
Reduce the problem to maxflow



k=2

What!? This is maxflow!?

- CityU OJ 261



What!? This is maxflow!?

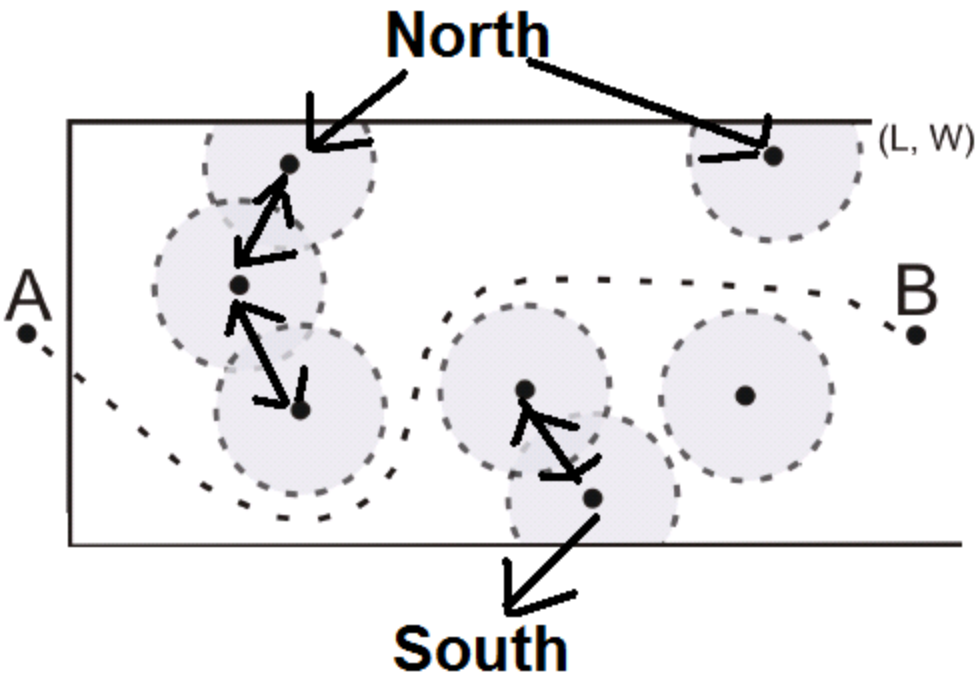
- If this problem just asks you whether a winger can go to L without being tackles or not…
- Reduce to graph problem!

Reduce to graph problem

- Let robots, border of North and South be vertex

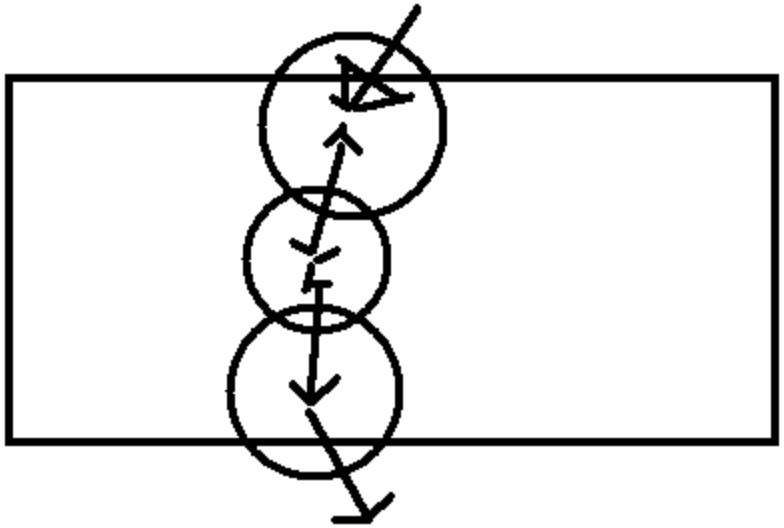
1. Connect North -> robot if the robot touches/overlap with border of North
2. Connect robot -> South if the robot touches/overlap with border of South
3. Connect robot_1 <-> robot_2 with undirected edge if two robots touches/overlap each other

Reduce to graph problem



Reduce to graph problem

- If there is a path from north to south, it means a winger need to "eat some tackles"



Transform to maxflow

- The problem now is transformed to "what is the **minimum** number of vertex should be remove in order to **disconnect** North and South"

- "Disconnect" = cut
- "minimum… disconnect" = minimum cut!
- minimum cut = maximum flow!!!

Transform to maxflow, another version

- The problem now is transformed to "How many paths going from North to South, if vertex can be only used once in the path?"

- similar to Edge disjoint Path
- But now is **Vertex disjoint Path**

Transform to maxflow

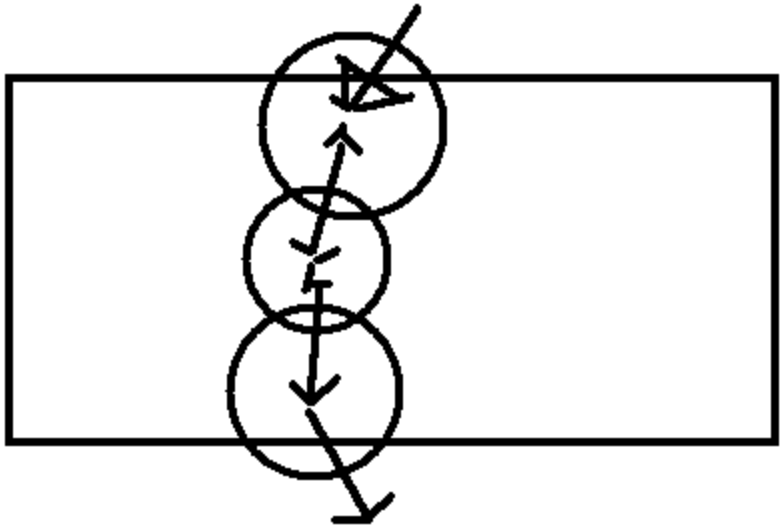
- Either of these will lead to same algorithm

- Cut vertex, not edge
- => Capacity of vertex is a main concern
- => Capacity of edge can be ignored

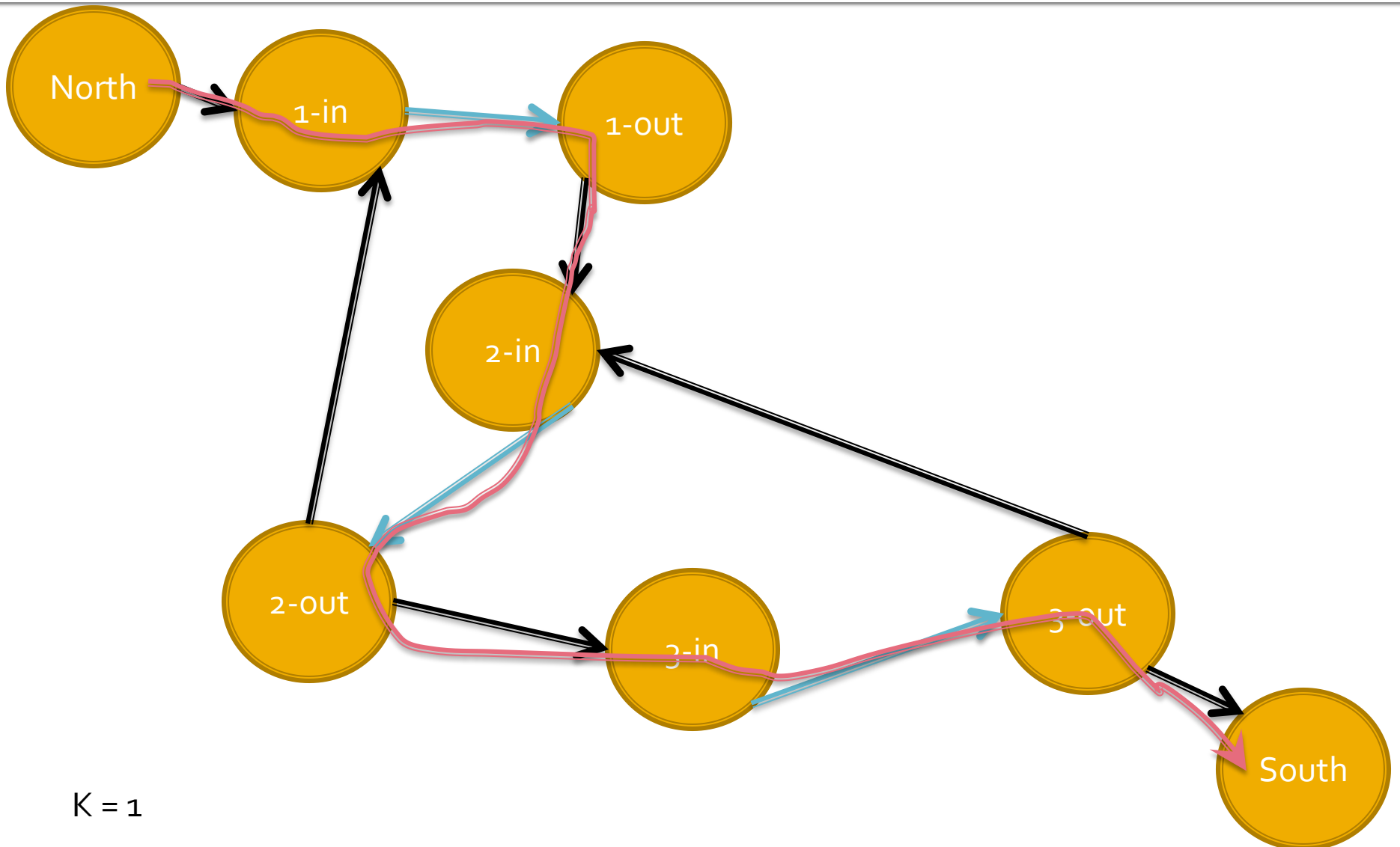
- Try to link up with previous topic

Transform to maxflow

- Set capacity of vertex be 1
  - Be the "bottleneck" of the network graph
- Set capacity of edge be INF



Transform to maxflow



K = 1

Transform to maxflow

- How to know a circle touch/overlap with
  - North border?
  - South border?
  - Another Circle?

- Q & A