* Dynamic programming
* Max-flow algorithm
* Application of max-flow min algorithm, look at the examples (Bipartite matching), Perfect matching
* Master decision problems

Test Questions:

1. Dynamic programming
   1. Optimal function
   2. Repeating function
   3. Time complexity
2. Max flow
3. Max flow application
4. P and np
5. Transformation
6. Bonus question

OPTa(n) = Ca + min(OPTa(n-1) + OPTb(n-1))

OBTb(n) = Cb + min(OPTb(n-1) + OPTb(n-1))

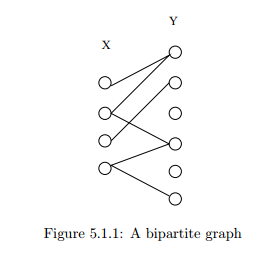
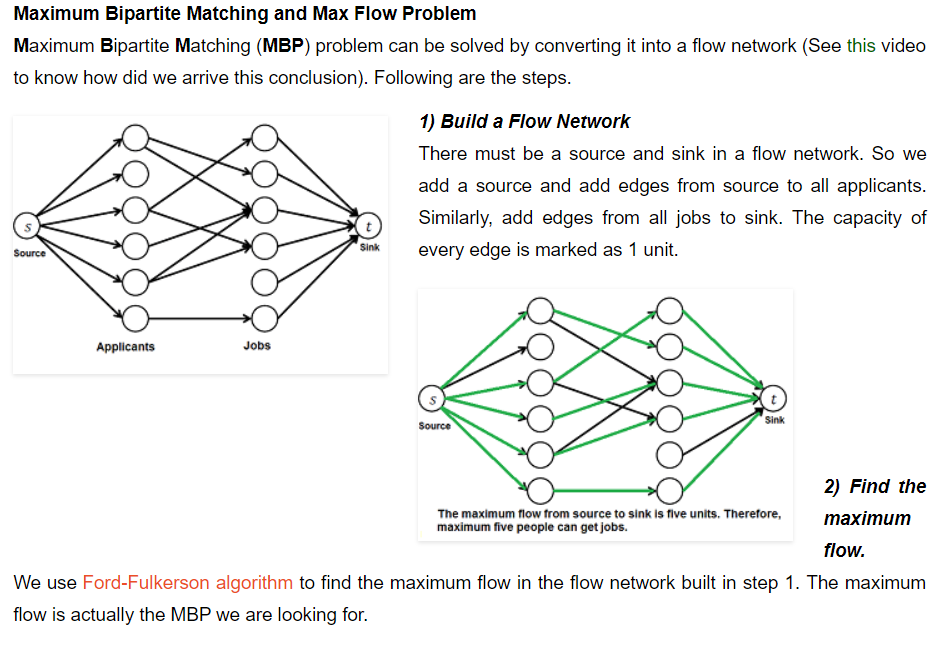
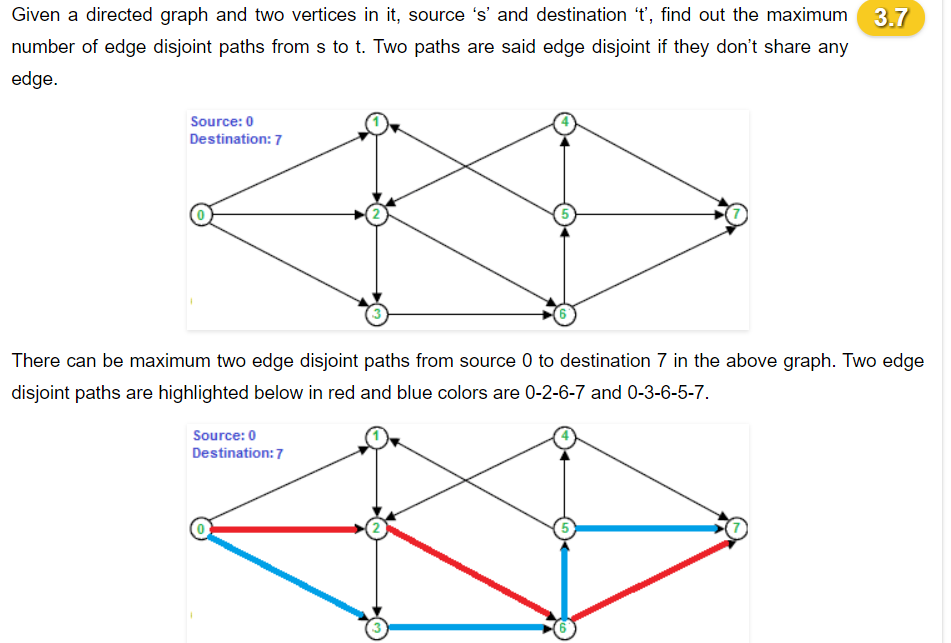
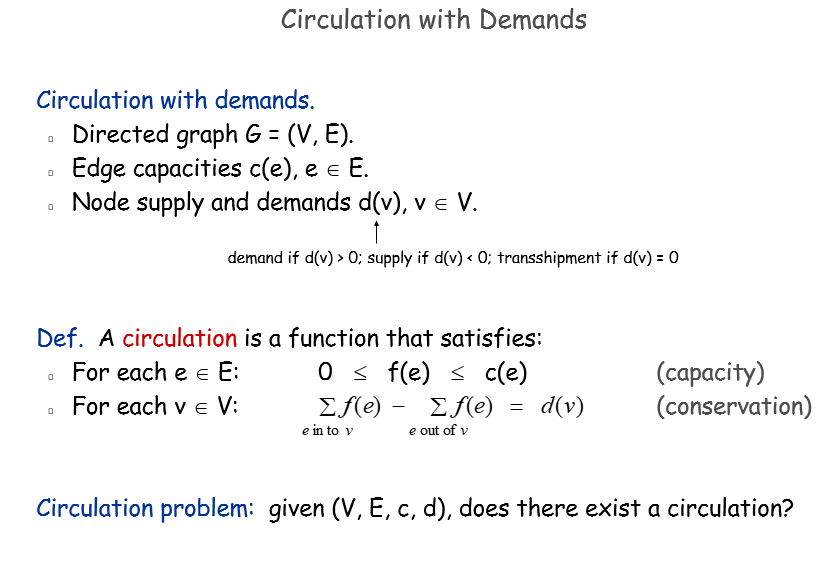
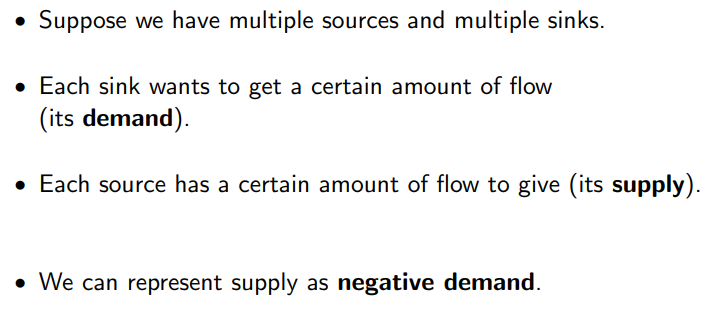
Dynamic Programming

* Dynamic programming for interval scheduling
  + Go through all jobs
  + if j = 0 set to 0
  + compare the weight of the current job + the OPT of the previous compatible job and just the OPT of the last job
  + 
  + Know runtime
    - O(nlogn)
* Dynamic programming for Knapsack
  + Have items with weight and value
  + Try to maximize for the highest amount of value that can fit in the bag

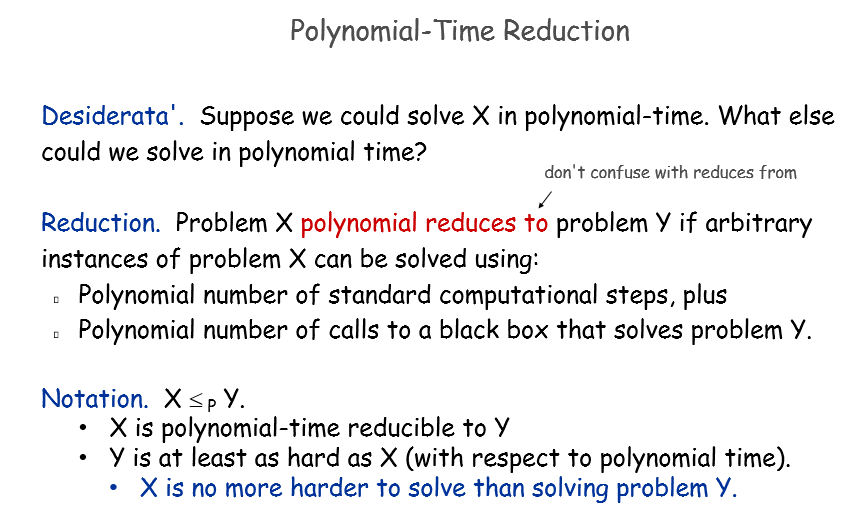
Max Flow

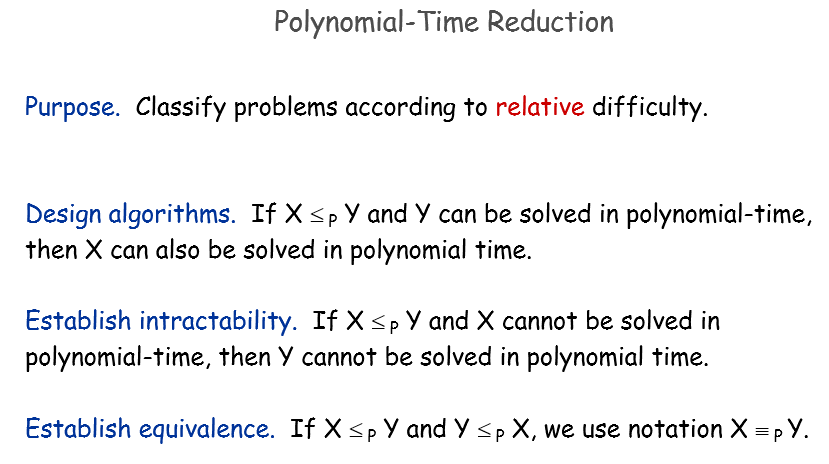
* Minimum cut problem
* Ford Fulkersin
* Capacity Scaling
  + Always choosing the path with the highest bottleneck capacity increases flow by max possible amount delta
  + The subgraph of the residual graph is only arcs with capacity of at least delta

Max Flow Applications

* Bipartite matching
  + 
  + A graph where every edge has one point in x and one point in y
  + 
    - Make all edges 1 for max flow
    - Equal to maxflow
* Perfect matching
  + Each node appears in exactly one edge. So basically each of the above graph every vertex on the left has one vertex on the right
* Disjoint matching
  + 
* Circulation with demands
  + 
  + Add the Amount in on every node, and the given demand
  + 
  + <https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/flowext.pdf>
  + All nodes with negative value (supply) will be attached to the source (s) all with a positive value (demand) will be attached to a sink (t)
  + Run max flow algorithm
  + Is circular is input = output

Polynomial-Time Reduction

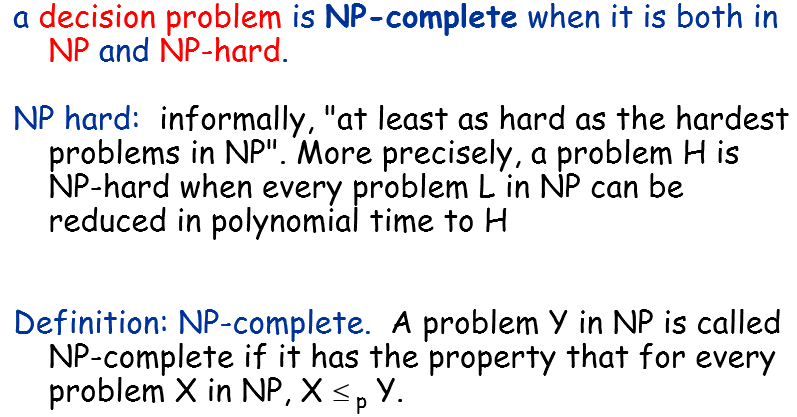




NP Complete

* P. Decision problems for which there is a poly-time algorithm.

-🡪 Finding solution is easy

* NP. Finding solution is hard, but certifying that a solution is correct is relatively easy. poly-time certifier
* EXP. Decision problems for which there is an exponential-time algorithm.
* P = NP. Is the decision problem as easy as the certification problem?
* If an NP-complete problem can be solved quickly for an equation, then all Np-complete problems can be solved quickly
* 
* 