Basic Problems: I: Task Scheduling

n tasks \( t_1, t_2, \ldots t_n \)

n people \( P_1, P_2, \ldots P_n \)

Each person capable of doing some subset of tasks \( S_i \subseteq \{ t_1, \ldots t_n \} \).

Q1: can we assign 1 task per person so that all tasks are done?

Q2: if not, how many tasks can we concurrently do
   (with at most 1 task per person)?

Q3: Each person also has a score for each task.
   \[ W_{ij} = \text{"quality of work if person } i \text{ does task } j. } \]

Now: assign tasks to people to maximize total weight.

Note: if \( W_{ij} > 1 \) if \( i \) can do task \( j \)

\[
\text{maximizing } \sum W_{ij} = \text{maximizing } \# \text{tasks } \]

So Q3 generalizes Q2.

Which generalizes Q1.

How to solve these problems?
II: Airline Schedule:

1. k planes at some locations. (say identical for now)
2. have m flights to handle.
   - each flight has start time, end time, start location and end location:
     \[ f = (\text{start}, \text{endtime}, \text{sloc}, \text{endloc}) \]
   - a plane can be used for flight \( f_1 \) and \( f_2 \)

\[
\text{if } \text{etime}_1 + (\text{service delay}) + (\text{travel time from } \text{eloc}_1 \text{ to } \text{sloc}_2) \\
\leq \text{stime}_2.
\]

Can \( k \) planes handle all \( m \) flights?

Baseball Elimination:

Given set of teams, each with \( w_x = \# \) of wins already.

and \( x, y \) teams, \( g_{xy} = \# \) of games \((x,y)\) still play.

\( \text{and target team } z = \text{Pirates}, \text{say} \)

Is it possible to have outcomes of the

\[ G = \sum_{x,y} g_{xy} \]  games

such that \( z \) is the top team

(i.e. if \( w_z \) is wins in these outcomes for \( z \))

\[ \geq w_x + \# \text{ wins for } x \quad \forall x \neq z \)
**Project Selection:**

\[ \{n \} = \{ 1, 2, \ldots, n^3 \} \]

- Set of projects each with profit \( p \in \mathbb{Z} \) (both positive and negative possible)
- Precedences/dependencies
  \( (i, j) \) or \( i \rightarrow j \)
  means must do \( i \) before doing \( j \).

- Pick set of projects so that
  \( S \subseteq \{ n \} \) s.t. \( \forall j \in S, \exists i \in S \) such that \( i \rightarrow j \) in dependencies
  then \( i \in S \) as well.

\[
\max \sum_{i \in S} p_i = \text{profit}
\]

And many more... (see HWs etc).

Can be solved using the same abstraction:--

- **Max Flow in a flow network**
  - Given a "flow network" with a source and sink,
    find the maximum flow in this network.

- Efficient algorithms for this
- Notion of "duality" and the max-flow/min-cut theorem.
- Integer vs. fractional solutions.
Flow network:

Directed graph \( G = (V, E) \) with special nodes \( s = \text{source} \)
\( t = \text{target or sink} \)
all others called internal nodes

Every edge has a capacity \( \text{cap} \geq 0 \).

Assume:
[No edge enters \( s \), no edge leaves sink \( t \)]

Flow: map \( f : E \rightarrow \mathbb{R}_{\geq 0} \) such that

(a) \( 0 \leq f(e) \leq \text{cap}(e) \).
(b) \( \sum_{e \in \text{in} u} f(e) = \sum_{e \in \text{out} u} f(e) \) \((\ast)\)

Fact: total flow leaving \( s \) = total flow entering \( t \).

Pro: sum \((\ast)\) over all internal nodes. then every edge between internal nodes cancels out. left with

\[ \sum_{e \in \text{in} t} f(e) = \sum_{e \in \text{out} t} f(e) \]

This is the \("\text{value}\) of the flow.

Problem (Max Flow): Find flow that has maximum value.
Idea #1: Greedy.
Find a path from $s$ to $t$. Push flow, repeat.

[Diagrams of flow networks with arrows indicating flow and capacities.]

if $f$ < $\sqrt{2}$ then after this step, used 1 unit on that edge. So left with capacity $\frac{1}{\sqrt{2}}$. What now?

Didn't find max flow = 2.

Idea #2: Greedy, but with possibility of undoing.

Residual graph: Given flow network $G = (V, E)$ and a flow $f$.
get new network $G_f = (V, E_f)$ as follows.

[Diagrams of residual graphs with arrows indicating residual capacities and flows.

Now in example above:

[Diagrams of flow networks showing $G_0$, $G_{f_1}$, and $G_{f_1+f_2}$ with arrows indicating flows and capacities.

Now no path from $s$ to $t$ in residual graph. Stop.
Claim: $f_1 + f_2 = \text{maxflow in } G$.}
Wait, what? Why is \( f + f \leq \text{even a flow?} \) And why max?

(Just proving for this example now, will do general proof soon). 

**Fact 1**: if \( f \) in flow in \( G \), and \( g \) is flow in \( G_f \) then \( f + g \) is flow in \( G \).

**Pf**: \( 0 \leq f(e) \leq \text{cap}(e) \) if \( f \) in flow in \( G \).

\[-f(e) \leq g(e) \leq \text{cap}(e) - f(e) \] if \( g \) in flow in \( G_f \).

\( 0 \leq (f + g)(e) \leq \text{cap}(e) \).

Also: \( f \circ g \) have flow conservation \( \Rightarrow \) \( f + g \) does too.

Similarly can prove.

**Fact 1b**: if \( f \) in flow in \( G \), and \( \underline{f} \) is max-flow in \( G \) \( \Rightarrow \) \( f + \underline{f} \) is max-flow in \( G \).

**Pf**: sps \( \underline{f} \) is max-flow in \( G \)

\( f \) is any flow

\( \Rightarrow (\underline{f} - f) \) is a flow in \( G_f \)

sats flow conservation, so only need to handle capacities.

\[ e \left[ 0, \text{cap}(e) - f(e) \right] \]

\[ \text{if } F(e) \geq f(e) \text{ then use } F(e) - f(e) \text{ on forward edge}. \]

\[ \text{if } F(e) < f(e) \text{ then use } f(e) - F(e) \text{ on back edge}. \]

For example.

\[ F \]

\[ f \quad \text{and} \quad \Rightarrow \quad \text{and} \quad \Rightarrow \]

\[ F - f \]
Great. So the flow we found $f_1 + f_2$ was a feasible flow in $G$.

Why isn't max flow for that graph $G$?

**Fact 2:** Capacity of any cut separating $(s,t)$ is max flow value.

Partitions of vertices $A, B$ s.t. $s \in A$ \\
$t \in B$.

Capacity of cut = total capacity of edges from $A$ to $B$ \\
(do not look at edges from $B$ to $A$)

So by Fact 2, applied to cut $E_{s3}, E_{a,b,e3}$

max flow value $\leq$ $\text{cap}(E_{s3}) + \text{cap}(s,b) = 1 + 1 = 2$.

Note: We have a feasible easy to verify certificate of optimality.

If I give you a flow $f$, you can quickly verify it is a flow.

And if I give you a cut $(A,B)$ separating $s \in A$ whose capacity equals value of $f$ \\
$\Rightarrow$ know that $f$ is maximum flow.
OK: here's the general algorithm (works for graphs with integer capacities).

**Algorithm Ford-Fulkerson** (Graph G, caps).

1. Find $s \rightarrow t$ path in $G$ with every edge having capacity $> 0$.
2. Augmenting path.
   - Push flow on path of value $\min_{e \in P} \text{cap}(e)$, call this $f$.
   - Update residual graph $g \leftarrow FF(G_f)$.
3. Return $g + f$.

**Fact 3:** If caps are integers, FF terminates in at most $\sum_{e \in E} \text{cap}(e)$ iterations.

In fact, if max flow $= F^*$, terminates in $\leq F^*$ iterations.

**Proof:** Each time max flow in residual graph drops by at least 1. (by Fact 1b).

**Corollary:** Runtime $= O(mF^*)$. Since each iteration uses DFS to find a path.

That's it, we're done:

- Max flow in $O(mF^*)$ time.
- (at least for integer capacity networks).
So application #1

maximum cardinality matching in bipartite graphs

(aka allocating jobs to people)

Bipartite graph

\[ B \]

Want a matching \( M \) of largest size

\[ \text{set of edges not sharing endpoints} \]

\[ \Rightarrow \text{exactly \ one\ job\ per\ person} \]

\[ \Rightarrow \text{exactly \ one\ person\ per\ job} \]

Claim:

\( \exists \text{a matching } M \text{ in } B \text{ of size } K \)

\( \iff \text{exists flow in } G \text{ of value } K \)

with integer \( f(e) \) values.

Proof:

[Diagram of bipartite graph and flow network, showing matching and flow]
can find max cardinality matching in bipartite graphs in time $O(mF^*) \leq O(mn)$

can do better. (not in this course)
do weighted matching — later lecture (on min cost flow)
do non-bip matchings (also not in this course)
Example: Multiple Source sinks. (with demands)

Graph: each node has number \( b_v \in \mathbb{Z} \) (say) \( \Leftarrow \) demand

some negative: \( b_a = -10 \) means want flow that has 10 units 4 flow leaving a (in total)

\( b_v = 7 \) means want flow that has 7 units 4 flow entering v (in total)

is a nice solution

\( 17 - 7 = 10 \).

Hence, since each unit 4 flow enters some node and leaves another

must have \( \sum b_v = 0 \).

How to find such a flow? Easy. (if one exists)

'Make a new source s, new sink t.'

Find a max flow.

Claim: Max flow of value = \( \sum u: b_u < 0 \)

\( \iff \) flow satisfying these demands. [Pf: ev]

new edges

all nodes u with bu < 0

all nodes v with bv > 0
Extensin: Flow with lower bounds. (and demands)

In addition to previous setting $G = (V, E)$ demands $d_v$ cap $c(e)$

Also have some edges with lower bounds: $\text{lower}(e) > 0$

Want at least $\text{lower}(e)$ flow on edge $e$.

Does $I$ such a flow?

Easy: "send the flow in advance". So

Claim:
Any flow solution in old graph with lower bounds and demands

$\iff$

Any flow solution in new graph with these new demands.

[Pf: exercise]

Problem II: Airline Scheduling

Recall from beginning of lecture:

Graph: one node per start location of $k$ planes: demand $b_u = -1$.
(can send out a plane).

two nodes for each flight.

(start node) cap = 1
lower bound = 1
end node

? must satisfy this flight!
End node $t$: demand = $k$ (does not matter where end up).

Edges (apart from the per-flight edges).

- edge from start node to plane $i$ to start node $i$
- or from end node $i$ to start node $j$

If the same plane can go from one to the other satisfying the constraints, (enough time, etc.), cap = 1.

- finally: from all planes $i$ and all end node $i$ to $t$, cap = 1

Claim: A flow satisfying these demands (and using integer flows on all edges) can give an itinerary for all the $k$ planes.

[Again: exercise]

⇒ can solve flight scheduling problem using Ford Fulkerson.

(+ reductions from previous parts).