2011 lecture 6 start

1 Maximum Flow

1.1 Definitions

Tarjan: *Data Structures and Network Algorithms*
Ford and Fulkerson, *Flows in Networks*, 1962 (paper 1956)
Ahuja, Magnanti, Orlin *Network Flows*. Problem: do min-cost.
Problem: in a graph, find a *flow* that is *feasible* and has maximum *value*.
Directed graph, edge *capacities* $u(e)$ or $u(v, w)$. Why not c? reserved for costs, later.
source s, *sink* t
Goal: assign a *flow* value to each edge: (raw/gross flow form)

- *conservation:* for every vertex v, $\sum_w g(v, w) - g(w, v) = 0$, unless $v = s, t$
- *capacity:* $0 \leq g(e) \leq u(e)$ (flow is *feasible/legal*) (sometimes lower bound $l(e)$)
- Flow *value* $|f| = \sum_w g(s, w) - g(w, s)$ (in gross model).

Water hose intuition. Also routing commodities, messages under bandwidth constraints, etc. Often "per unit time" flows/capacities.
Maximum flow problem: find flow of maximum value.
Path decomposition (another perspective):

- claim: any s-t flow can be decomposed into paths/cycles with quantities
- proof: induction on number of edges with nonzero flow
- if s has out flow, traverse flow outward from s
- till find an s-t path or cycle (why can we? conservation) and kill
- if t has in-flow (won't any more, but no need to prove) work backwards to s or cycle and kill
- if some other vertex has outflow, find a cycle and kill
- corollary: flow into t equals flow out of s (global conservation)
- Note: every path is a flow (balanced).
- Note: at most m paths/cycles (zero out one edge's flow each time)

Point A. 15 min to here.
Decision question: is there nonzero feasible flow

- Suppose yes. consider one.

- decompose into paths
- proves there is a flow path
- Suppose no
- then no flow path
- consider vertices reachable from s
- they form a **cut** ($s \in S, t \in T = \overline{S}$)
- no edge crossing cut

2012 lecture 8 start
What about max-flow?

- Need some upper bound to decide if we are maximal
- Consider any flow, and cut (S, T)
- decompose into paths
- every path leaves the cut (at least) once
- So, outgoing cut capacity must exceed flow value
- **min cut** is upper bound for **max flow**

Suppose have some flow. How can we send more?

- Might be no path in original graph
- Instead need **residual graph:**
  - Suppose flow $f(v, w)$
  - set $u_f(v, w) = u(v, w) - f(v, w)$ (decrease in available capacity)
  - set $u_f(w, v) = u(w, v) + f(v, w)$ (increase in capacity indicates can remove (v, w) flow
- If **augmenting path** in residual graph, can increase flow
- What if no augmenting path?
- Implies zero cut (S, T) in residual graph
- Implies every S-T edge is saturated
- Implies every T-S edge is empty
- i.e. cut is saturated
- consider path decomposition

- each path crosses the cut exactly once (cannot cross back)
- so flow crossing cut equals value of flow
- i.e. value of cut equals value of flow
- but (again by path decomposition) value of flow cannot exceed value of cut (because all paths cross cut)
- so flow is maximum

We have proven Max-flow Min-cut **duality**:

- Equivalent statements:
  - f is max-flow
  - no augmenting path in G_f
  - $|f| = u(S)$ for some S

Proof:

- if augmenting path, can increase f
- let S be vertices reachable from s in G_f. All outgoing edges have $f(e) = u(e)$ and incoming have $f(e) = 0$
- since $|f| \leq u(S)$ always, equality implies maximum

Another cute corollary:

- Net flow across any cut (in minus out) equals flow value
- True for a path as it must go out once more than in
- So true for a sum of paths

Note that max-flow and min-cut are witnesses to each others' optimality.

1.2 Algorithms

1.2.1 Augmenting paths

Can always find one.
If capacities integral, always find an integer.

- So terminate
- Running time $O(mf)$, $O(m)$ to find each augmenting path, each augmenting path increases flow by 1.
- **Lemma:** flow integrality
- Polynomial for unit-capacity graphs

- Also terminate for rationals (but not polynomial)
- might not terminate for reals.

Note: complex greedy algorithm

- always have 'greedy' augmentation
- but augmentations may undo previous, adding flow to an edge and then removing
- diamond example (s, x), (s, y), (y, t), (x, t), (x, y).

1.2.2 max capacity augmenting path

Running time:

- recall flow decomposition bound
- Get $1/m$ of flow each time.
- So $m \log f$ flows suffice for integer f

How find one? Prim.
Overall, $O(m^2 \log f)$
weakly vs. strongly polynomial bounds

- pseudopoly in magnitudes of numbers
- poly (specifically weakly poly) is poly in input size including bits
- strongly is poly in input size ignoring bits

Works for rational capacities too.
2011 Lec 7 start

1.3 Scaling

Idea of max-capacity augment was to be greedy

- get most of solution as quick as possible.
- decrease residual flow quickly

Another approach: scaling.

- Gabow '85.
- Also [Dinitz '73], but appeared only in Russian so, as often the case in this area, the american discovery was much later but independent.

General principle for applying "unit case" to "general numeric case".
Idea: number is bits; bits are unit case!
Scaling is reverse of rounding.

- start with rounded down value (drop low order bits)
- put back low order bits, fixup solution

big benefit: aside from scaling phase, often as simple as unit case (eg no data structures!)
Basic approach:

- capacities are $\log U$ bit numbers
- start with all capacities 0 (max-flow easy!)
- roll in one bit of capacity at a time, starting with high order and working down.
- after rollin, update max-flow by finding max-flow in residual graph
- effect of rollin:
  - double all capacities and flow values
  - double all residual capacities
  - add 1 to some residual capacities
- after $\log U$ roll-ins, all numbers are correct, so flow is max-flow

Algorithm: repeatedly

- Shift in one bit at a time
- Run plain augmenting paths

Analysis:

- before bit shift, some cut has capacity 0
- after bit shift, cut has capacity at most m
- so m augmentations suffice.

Running time: $O(m^2 \log U)$.
Discuss relation to max capacity algorithm.
weakly polynomial

1.4 Problem Variants

Multiple sinks
Explicit supplies/demands
Edge lower bounds

- send flow along lower bounds (creating residual arcs)
- creates additional demands and supplies in residual.

- solve
- add flow on lower bound edges to cancel imbalance of new supplies and demands
- relies on linearity: feasible flow in residual graph can be added to feasible flow in G to get another feasable flow in G.

Bipartite matching.
Vertex capacities.