# Greedy Algorithms

Note Title 8/30/2015

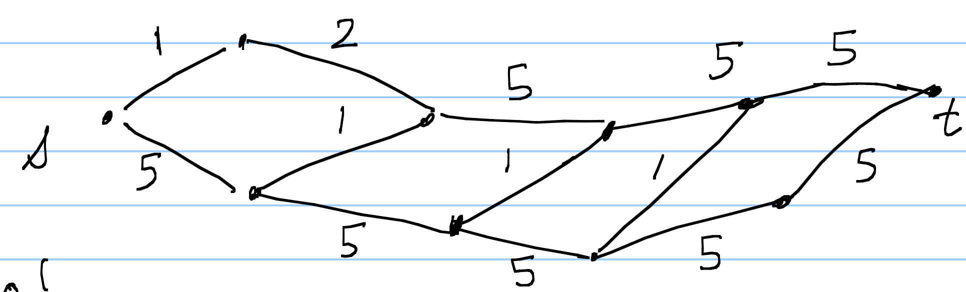
Many computational problems are about optimization. E.g.

- shortest path
- maximum flow
- minimum congestion
- minimum makespan

etc.

A natural, widely-applicable paradigm for these and many other problems is GREEDY: At each step, make a choice that provides the most benefit given choices made so far.

E.g. for shortest path routing, "send to neighbor that is closest to destination." Will this work?




No!

Graph $G = (V, E)$ vertices,

edges (pairs of vertices)

$w : E \to \mathbb{R}$ weights on edges

Problem Find minimum (max) subset of edges that satisfies property P.

P: "A single cycle"

"Planar"

"clique"

"connected" (i.e. contains a path between each pair of vertices).

Claim Minimal subset of edges that connects every pair of vertices must be

(1) acyclic (no induced cycles)

(2) of size $|V|-1 = n-1$

(3) Any connected subgraph with $n-1$ edges must be a tree.

Pf (1) Let $F \subseteq E$ be a connected subgraph.

Suppose $C$ is a cycle induced by $F$.


Remove any edge of $C$. Any pair of vertices is still connected. So $F$ is not minimal

(2) $|F| = n-1$. True for $n = 1, 2$.

Remove any edge. $F \setminus \{e\}$ has 2 components

(else $F$ is not minimal)

Each has $< n$ vertices

$n_1 - 1 + n_2 - 1 + 1 = n - 1$. ✓ $F$ is a TREE.

Problem (MST): Given $G = (V, E)$ $w: E \to \mathbb{R}$

find a minimum weight spanning tree of $G$.

(alternatively: find maximum wt spanning tree).

Algorithm (KRUSKAL) $n = |V|$, $m = |E|$

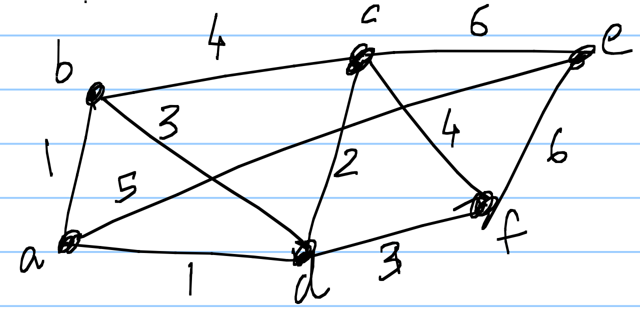
1. Sort edges in increasing order of weight

$e_1 \quad e_2 \quad e_3 \ldots \quad e_i \quad \ldots \quad e_m$

$w(e_1) \leq w(e_2) \quad \ldots \quad w(e_m)$

2. Add edges in this order that do not create cycles.

E.g.




$(a,b), (a,d), (c,d), (d,f), (b,d), (b,c)\ (c,f), (a,e), (c,e)$

✓ ✓ ✓ ✓ × × × ✓

$1 + 1 + 2 + 3 + 5 = 12.$

Q. Does this Greedy algorithm for MST always work?

Thm 1: Kruskal's algorithm always produces an MST.

Lets order edges by weight, and edges of equal weight lexicographically, i.e. $(a,b) < (a,c)$
$(a,b) < (c,b)$.

Then Kruskal produces a unique Spanning tree.

Cut $(S,\bar{S})$ is defined by a subset $S \subseteq V$ and its complement $\bar{S} = V \setminus S$, as all edges between $S$ and $\bar{S}$

$$(S,\bar{S}) = \{ (i,j) : i \in S,\ j \in \bar{S} \}$$

Lemma 1. For any edge $e$ whose weight is the minimum in some cut $(S,\bar{S})$, $\exists$ MST $T$ containing $e$.

Pf. Suppose $e$ is not in any MST.
Take MST $T$. Add $e$
then $e$ induces a cycle $C$ (since $T \cup \{e\}$ has two paths between endpoints of $e$)
Consider cut $(S,\bar{S})$ for which $e$ is minimum weight

$C$ must cross $(S, \bar{S})$ in some other edge $f$.

$wt(e) \leq wt(f)$.

Let $T' = T \cup \{e\} \setminus \{f\}$

$wt(T') \leq wt(T)$

$T'$ is a spanning tree (since it is connected and has $n-1$ edges) $\times$

---

Lemma 2. Let $T_K$ be the tree found by Kruskal's algorithm. For each edge $e \in T_K$

$\exists$ cut $(S, \bar{S})$, (1) $e$ is the min edge of $(S, \bar{S})$

(2) $e$ is the unique edge in $T_K \cap (S, \bar{S})$.

(only edge from $(S, \bar{S})$ in $T_K$).

Pf. Consider $T_K \setminus \{e\}$  $e = (u, v)$

Then $u, v$ are in different components $S, \bar{S}$ and $e$ is the only edge of $(S, \bar{S})$ in $T_K$.

At the point when $e$ was added,

$comp(u)$ and $comp(v)$ were not connected

So $e$ has smaller wt than all edges of $(S, \bar{S})$.

---

Pf. (of Thm 1.) Suppose $T_K$ is not an MST.

Let $T$ be an MST. We will show that $w(T_K) \leq w(T)$

Order the edges of $T_K$. Take the next edge $e$ of $T_K$

that is not in $T$. $T \cup \{e\}$ has a cycle $C$.
Let $(S, \bar{S})$ be the cut for $e$ given by lemma 2.
Now remove $f \in C \cap T$ that crosses $(S, \bar{S})$

$w(e) \leq w(f)$

and $T \cup \{e\} \setminus \{f\}$ is a tree.

Repeat for all edges of $T_K$ not in $T$.

We never remove an edge of $T_K$ since edge added to form $T_K$ and no other edge of $T_K$ is in the same cut. #

---

When does Greedy work?

We don't know the complete answer.

But we know some nice conditions under which it is guaranteed to work.

$\boxed{\text{Matroid}}$ $M$ has a base set of elements $U$ and a collection of subsets of $U$ called <u>independent</u> sets. $M = \{U, \mathcal{I}\}$. The sets in $\mathcal{I}$ satisfy

1. $\phi \in \mathcal{I}$
2. $A \subseteq B$ and $B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$
3. $A, B \in \mathcal{I}$, $|A| < |B| \Rightarrow$
   $\exists\, e \in B \setminus A$ s.t. $A \cup \{e\} \in \mathcal{I}$.

<u>Problem</u>: Given weights for elements, FIND

# Maximum weight independent set of $M$.

Algorithm (Greedy): Add heaviest element that maintains an independent set.

Thm 2: Greedy finds max ind. set of any matroid.

Pf. Let $X_1, X_2 \ldots$ be Greedy choices

$Y_1 \; Y_2 \quad \ldots$ — OPT ind. set

order s.t. $w(X_1) \geq w(X_2) \geq \ldots w(X_k) \geq$

$w(Y_1) \geq w(Y_2) \geq \ldots w(Y_k) \geq$

Let $k$ be the first place where $w(X_k) < w(Y_k)$

Consider $A = \{X_1 \ldots X_{k-1}\}$ $B = \{Y_1 \ldots Y_k\}$

$\exists \; Y_j \in B$ s.t. $A \cup \{Y_j\}$ is ind. and

$w(Y_j) \geq w(Y_k) > w(X_k)$.

So greedy would have chosen $Y_j$ ! $\quad \star$

Example Given a set of vectors

$V = v_1, v_2, \ldots\ v_m \in \mathbb{R}^n$

with weights $\omega_1, \omega_2, \ldots\ \omega_n$

Find a max weight linearly indepent subset of vectors of $V$.

Solution. Show that linearly ind. subsets form a matroid.

---

Back to MST.

PRIM'S algorithm

- Start with any vertex
- add minm wt edge to some new vertex
- Repeat till no vertices left.

By cut property (Lemma 1 + Lemma 2), PRIM gives an MST

Lemma 3 (cut property). $X \subseteq T$, $T$ is an MST

For any $e$ s.t. $e$ is min wt edge of some $(S, \bar{S})$ and $X \cap (S, \bar{S}) = \emptyset$, $\exists$ MST containing $X \cup \{e\}$

## Running time of Kruskal..

SORT $O(n \log n)$

For each edge $e = (u, v)$, need to check if $u, v$ are already in one components.

For each vertex $v$, keep a component name (initially just name of vertex).

When two components merge, update component name for all vertices in one component to the name of the other component.

Time? $n^2$?

Update only smaller component

so component of a vertex DOUBLES each time its name is updated.

At most $\log_2 n$ updates per vertex

Total $= O(n \log n)$.

Q. Can this be done faster?

A. YES! $O(n \log^* n)$

Also: There is an $O(n)$ RANDOMIZED algorithm for MST.