Spanning trees

Trees are the minimal connected graphs. Spanning trees are minimal subgraphs that contain all the vertices and are connected.

Definition

Let G be a connected graph. A *spanning tree* of G is a subgraph $T \subseteq G$ such that T is a tree, and T contains every vertex of G.

Side point: Kirkchoff's Matrix Tree Theorem

Spanning trees of K_n are the same thing as labelled trees on n vertices.
As a generalization of Cayley's formula, can compute the number of spanning trees of any graph G as the determinant of a matrix.

Weighted graphs

Edges often have a “cost” associated to them – the time, money, or distance of the corresponding route/connection.

Definition

Weighted graph A *weighted graph* is a graph G together with a weight function $w : E(G) \to \mathbb{R}$. Normally we assume $w(e) \geq 0$ for all edges e.

Weighted graphs are often encoded in tables:

A						
3	B					
6	7	C				
7	9	5	D			
6	8	9	4	E		
8	7	9	8	9	F	
9	8	6	7	5	7	G

Minimal spanning trees

Motivating problem:

Suppose that the vertices of a weighted graph G represented cities, and the weight $w(e)$ of an edge was the cost of building a road between the cities. What's the cheapest way to connect all the cities?

Definition

Let $T \subseteq G$ be a spanning tree of a weighted graph. The weight of T is the total weight of all its edges:

$$w(T) = \sum_{e \in T} w(e)$$

Problem becomes: find the minimal weight spanning tree

Checking every spanning tree too slow: K_n has n^{n-2}

Many solutions. Two: Kruskal and Prim

Loose concept: Greedy Algorithms

A *greedy algorithm* doesn't plan ahead, but just does the best it can at each stage.

Definition (Kruskal's algorithm)

Start with T having no edges. Iteratively:

- Look at cheapest remaining edge e
- If adding e to T creates a loop, discard e
- Otherwise, add e to T

Fairly clear: produces a spanning tree

But it's *not* clear this spanning tree is *minimal*.

Another approach: Prim

Kruskal: a global view, "avoid cycles"

- Kruskal's algorithm looks at all edges at start
- T may be disconnected at intermediate steps

Prim: local view, "build tree"

Start at one vertex and explore out

Definition (Prim's algorithm)

Start $T = v$, a single vertex. Iteratively:

- Find the cheapest edge $e = vw$ from $v \in T$ to $w \notin T$
- Add e and w to T

Fairly clear: produces a spanning tree

But it's *not* clear this spanning tree is *minimal*.

Why do Kruskal and Prim work?

Exchange principle:

Let T be a spanning tree of G, and $e = xy$ an edge not in T. Then:

- Unique path P from x to y using only edges of T
- If f any edge in P, then $T' = T \setminus f \cup e$ a spanning tree

i.e., can exchange edges in P for e.

Basic idea of proofs:

- Let T be spanning tree produced by algorithm
- Let T_m be a minimal spanning tree
- Transform T_m to T edge by edge using exchange principle
- Show each step is a minimal spanning tree

Key: always add cheapest edge in T but not T_m.

Finding *all* minimal spanning trees

All edges have distinct weights:

- Never have to make an arbitrary choice
- Unique minimal weight spanning tree

All edges have the same weight:

- Any spanning tree is minimal
- Probably too many to write down

A few edges have repeated weights:

- Only a few places we have to make a choice
- Can find them with case by case analysis breaking ties