Math 38 — Graph Theory
Optimization and weighted graphs

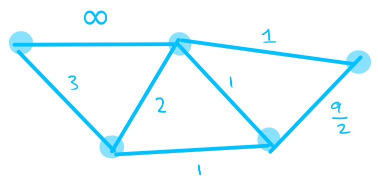
Nadia Lafrenière

04/25/2022

We consider two problems, and use weighted graphs to solve them: The first one is the problem of the minimal spanning tree (where minimal refers to the weight on the edges), and the second one is the shortest path.

Weighted graphs

A <u>weighted graph</u> is a graph with edges labeled by numbers (called weights). In general, we only consider nonnegative edge weights. Sometimes, ∞ can also be allowed as a weight, which in optimization problems generally means we must (or may not) use that edge.

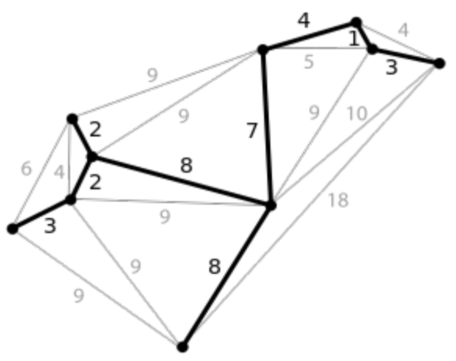


Minimum spanning tree

<u>Problem:</u> Given any weighted graph, find the spanning tree with the minimum weight, where the weight of a tree is the sum of the weights of its edges.

<u>Example</u>

An internet provider wants to wire cable in a new housing development and wants to reach every house. However, due to certain weather conditions and due to the distance between houses, the cost of reaching houses might not be the same from every path. The graph below illustrates the potential cost of every section: that is the weight of the edges, and the houses correspond to vertices.
How can they reach every house at minimum cost?



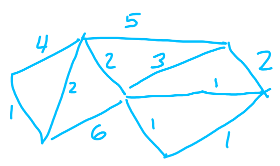
To make sure they connect every house, they must build a spanning tree. To find the spanning tree with minimal cost, they can use, for example, Kruskal's algorithm. They have no incentive to create a cycle.

Kruskal's algorithm: Given a connected weighted graph $G=(V,E)$, find its minimal spanning tree.

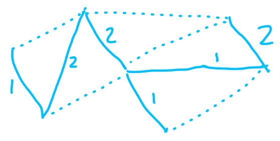
Idea: At every step, we have a forest H. We add edges from G to H until H is a spanning tree. The subgraph always stay acyclic.
To ensure H is minimal, we consider edges to be added in increasing order of their weight.

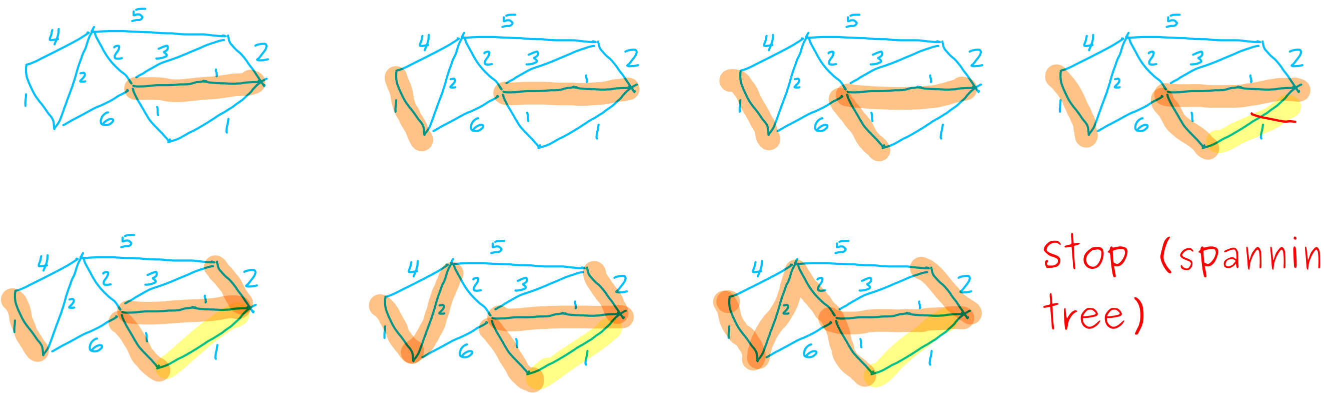
Initialization: H has $|V|$ isolated vertices (no edge). The edges of G are sorted in increasing oder of their weight.
Iteration: Consider the next smallest edge of G. If adding it to H reduces the number of components of H, we do so. (Otherwise, it creates a cycle, so we do not add it).
Stop: When we get at the end of the list of edges, or when H is a spanning tree, whichever comes first.

Example



has minimal spanning tree of weight 9





Stop (spanning tree)

Theorem (Kruskal, 1956)
In a connected weighted graph, Kruskal's algorithm constructs a minimum-weight spanning tree.

Proof
We prove the following two things:
1) The result is always a spanning tree.
2) There can't be any spanning tree with smaller weight.

1) We must show that the result is acyclic, connected and reaching every vertex.

- It is acyclic, since we only add edges that reduce the number of connected components. These edges cannot create cycles.
- Obviously, if we stop because the graph is a spanning tree, it is connected and reaches every vertex. Otherwise, we stop because we considered adding every edge; we did not add them only if they did not reduce the number of components. So the number of components in the forest is the same as in the original (connected) graph. Since we started with all the vertices of G, a tree is always spanning.

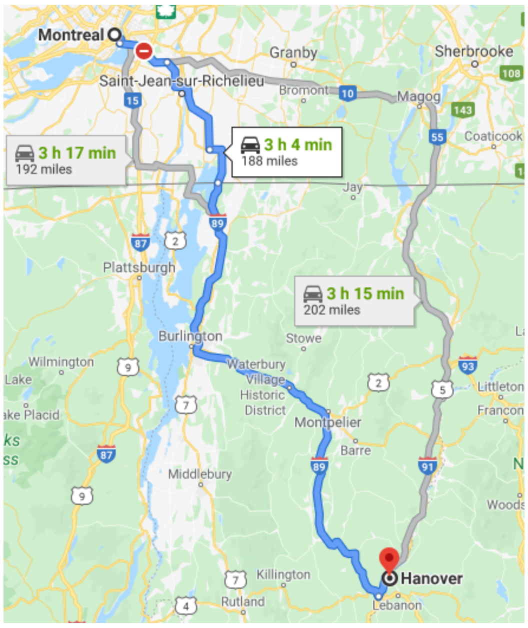
2) We prove it by contradiction. Assume T is a spanning tree with lower weight than H (obtained by the algorithm). They both have the same number of edges (since they are spanning trees), so there is at least one edge e in T but not in H. Conversely, there is an edge e' of H that is not in T. Since T has lower weight than H, we can choose $e<e'$. In fact, take the smallest such e. We considered e before e' and did not add it to H. Necessarily, e would have created a cycle in H, so there is a cycle in T (because e is the smallest edge in T but not in H). A contradiction. So $T=H$, and H is the minimal spanning tree.

Kruskal's algorithm is not the only algorithm that does so. See, for example, Prim's algorithm, where you grow a tree from one single vertex.

Shortest paths

Given two vertices in a labeled graph, what is the shortest path?

Dijkstra's algorithm gives all the distances from a given vertex u to other vertices in the graph.

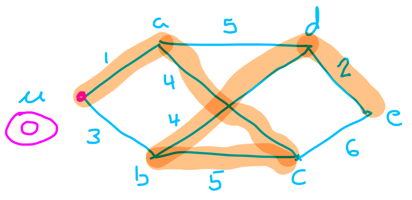


Observation: This cannot be done using a greedy algorithm.

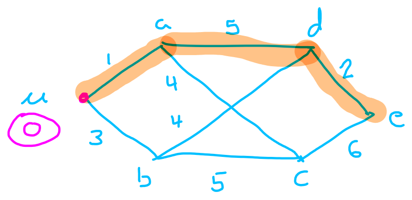
greedy = informed by local data

Example

Shortest path from u to e



With a greedy algorithm: $d(u,e)=16$?



A (much) shorter path: $d(u,e) \leq 8$

Dijkstra's algorithm

We need a weighted graph, and we compute the minimum-weight path from one specific vertex u to every other vertices. An edge that does not exist is equivalent to an edge with weight ∞.

Idea: The distance $d(u,v)$ is the weight of the edge between u and v if they are adjacent. We give tentative distance $d(u,w)$ for every vertex w not adjacent to u, and that distance never increases during the process.

Initialization: The set of visited vertices is $\{u\}$, $d(u,u)=0$, the tentative distance from u to x, $t(x)$, is the weight of the edge between u and x (∞ if it does not exist).

Iteration: Take the vertex x with shortest value of $t(x)$ amongst the non-visited vertices. The distance to u is $d(u,x)=t(x)$.

"Visit it" by modifying $t(y)$ for all its neighbors, by the minimum of these:

- what $t(y)$ was already; the tentative distance does not change
- $d(u,x)$ + the weight of the edge xy; there is a shorter path

Stop when you visited every vertex (for a connected graph).

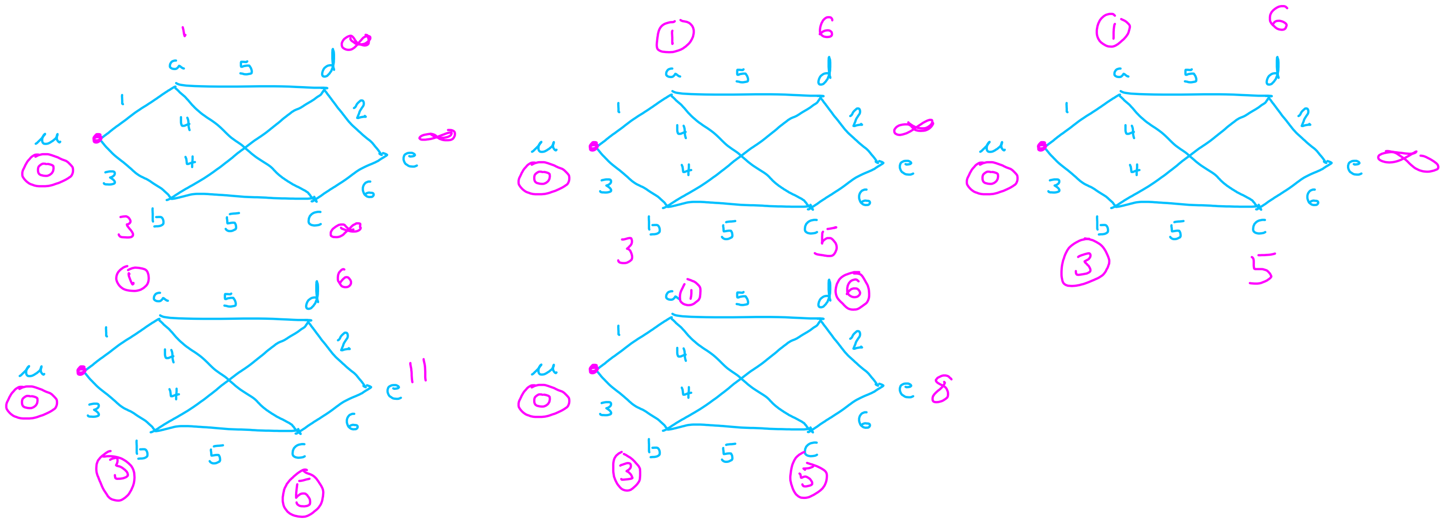
Theorem

Dijkstra's algorithm computes the distance $d(u,x)$ for every vertex x in a connected graph.

Sketch of proof

- Previously visited vertices cannot see their distance increase.
- When a vertex is visited, there cannot be a shorter path passing through a non-visited vertex.

Example



Reference: Douglas B. West. Introduction to graph theory, 2nd edition, 2001. Section 2.3