Greedy Method
Minimum Spanning Trees – Kruskal's Algorithm

Textbook Reading

Chapter 6, pp. 261-266

Minimum Spanning Tree (MST) Problem

In this course we cover two classical algorithms for computing a minimum spanning tree (MST),

Kruskal's algorithm – based on growing a forest

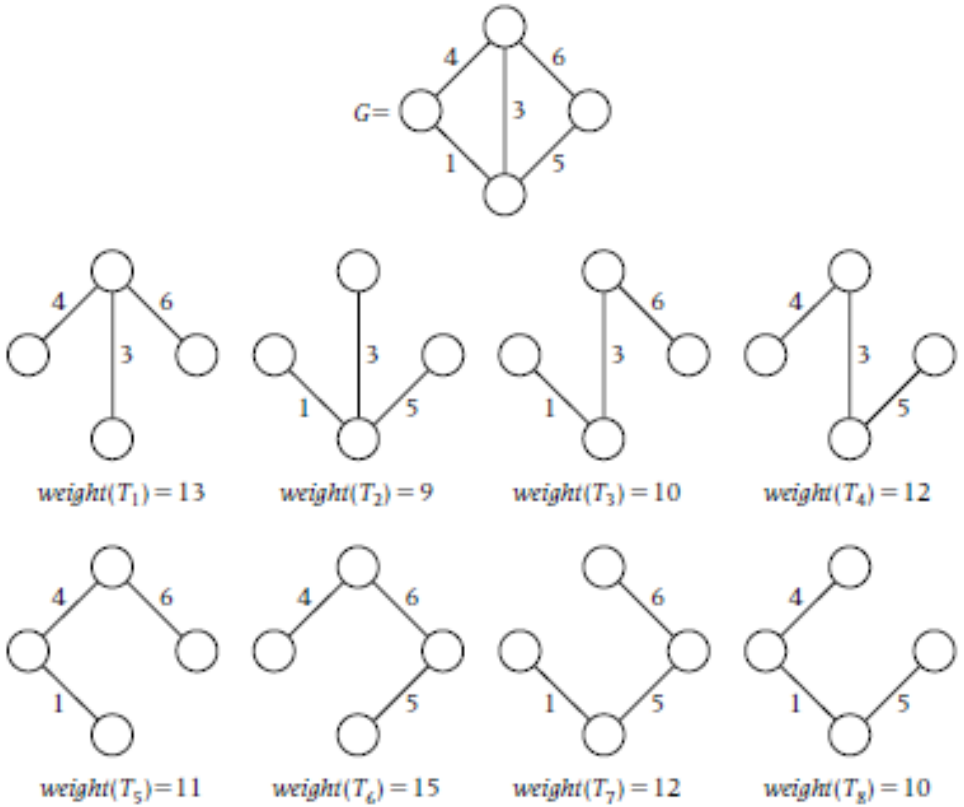
Prim's algorithm – based on growing a tree

The textbook reading for minimum spanning trees is Chapter 6, Section 5. Kruskal's algorithm will be covered in here (textbook reading Subsection 6.5.1) and Prim's will be covered in the video (textbook reading Subsection 6.5.2).

Definition of a minimum spanning tree

- Given a spanning tree *T* in a graph *G* with a weighting *w* of the edges *E* of *G*, the *weight* of *T*, denoted *weight*(*T*), is the sum of the *w*-weights over all its edges.
- If *T* has minimum weight over all spanning trees of *G*, then we call *T* a **minimum spanning tree**.

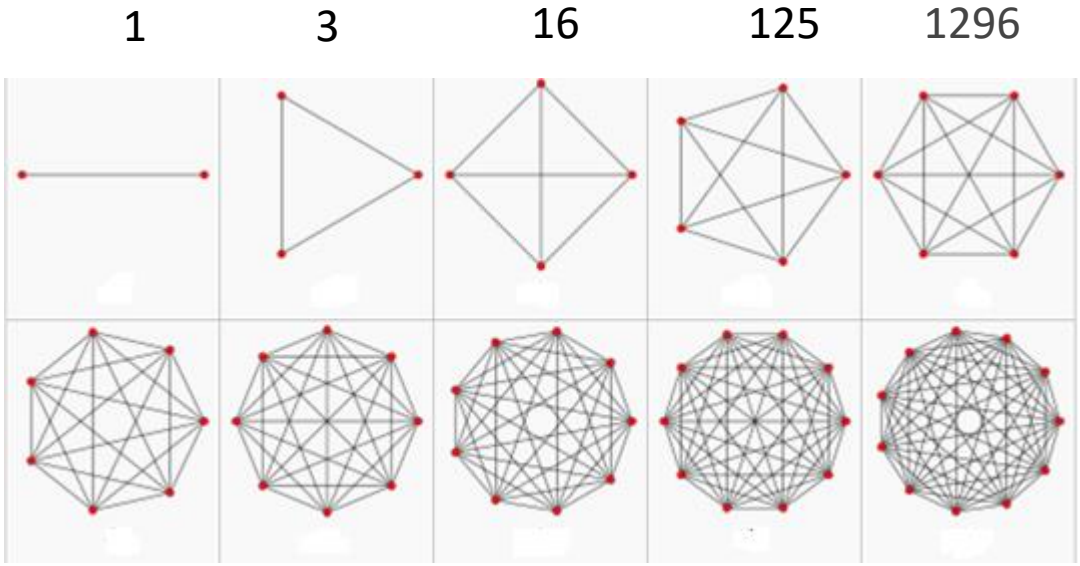
Minimum Spanning tree for Sample Graph *G*



All 8 spanning trees of sample graph G are shown above. The minimum spanning tree has weight 9.

Enumerating Spanning Trees

Clearly, the complete graph K_n has the most spanning trees for a graph on n vertices



Can you guess the formula for the number of spanning trees of K_n?

Cayley's Theorem

By a theorem of Caley the number of spanning trees of K_n, the complete graph on n vertices is

$$n^{n-2}$$

Therefore, it is infeasible even for relatively small n to use brute force to find the spanning tree having minimum weight.

High-Level Description Kruskal's Algorithm

Input: Connected graph $G = (V,E)$, weighting w of the edges

Output: Minimum Spanning Tree T, i.e., weight(T) is minimum

Design Strategy employed: the greedy method

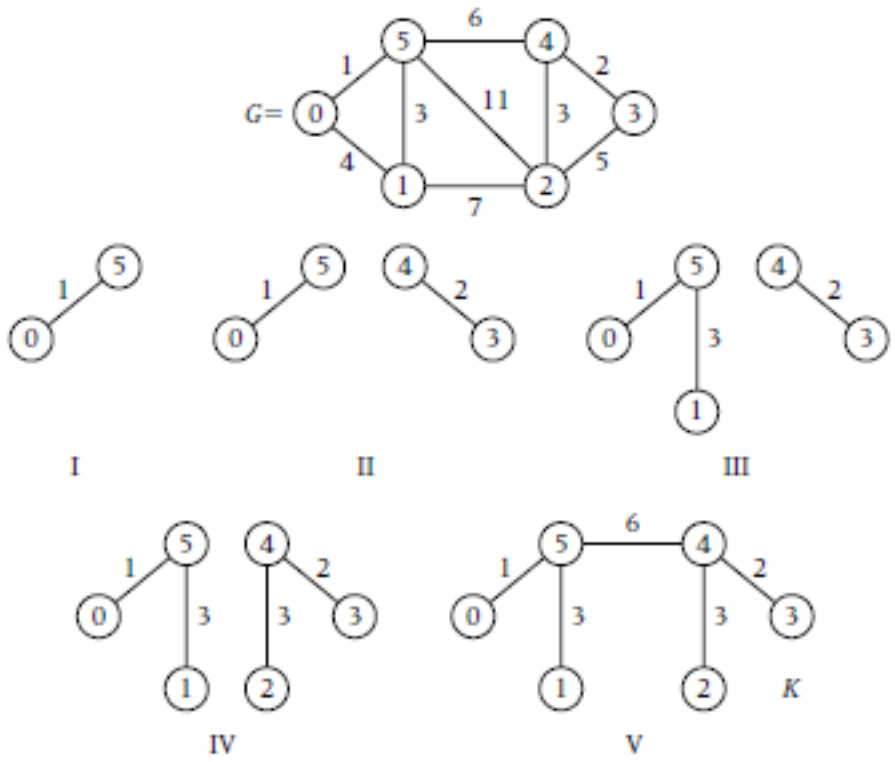
Notation: As usual we let $n = |V|$ (order of G) and $m = |E|$ (size of G)

Compute a sequence of n forests $F_0, F_1, \ldots, F_{n-1}$, where **$F_0$ is the empty forest, i.e., consists of a set of n isolated nodes** and F_i is obtained from F_{i-1} by adding a single edge e_i, denoted

$$F_i = F_{i-1} + e_i \text{ , } i = 1, 2, \ldots, n-1,$$

where e_i is chosen so that it has **minimum weight** among all the edges not belonging to F_{i-1} and **doesn't form a cycle** when added to F_{i-1}.

Action of Kruskal's for sample graph *G*



Testing for cycles

It is important to be able to test for cycles efficiently, otherwise this testing could dominate the complexity of the algorithm. We do this efficiently by using the **Disjoint Set ADT** where the **collection of disjoint sets is the vertex sets of the trees in the forest** being grown.

Let e = {*u*,*v*} be the next edge to be considered.

Perform operations

call to *Find2*(*Parent*[0:*n* – 1],*u*,*r*)
call to *Find2*(*Parent*[0:*n* – 1],*v*,*s*)

If *r* = *s* then *u* and *v* belong to the same set in the collection of disjoint sets, which implies they belong to the same tree, which in turn implies that a cycle is formed; otherwise a cycle is not from. Thus,

if *r* = *s* then REJECT edge e
else
    add edge e to current forest
    perform the union of sets containing r and s, respectively

Pseudocode for Kruskal's Algorithm

procedure *Kruskal*(*G*,*w*,*MSTree*)
Input: *G* (a connected graph with vertex set *V* of cardinality *n*
and edge set $E = \{e_i = u_i v_i \mid i \in \{1,...,m\}\}$)
w (a weight function on *E*)
Output: *MSTree* (a minimum spanning tree)
Sort the edges in nondecreasing order of weights $w(e_1) \le ... \le w(e_m)$
Forest ← ∅
Size ← 0
for *i* ← 0 **to** *n* – 1 **do** //initialize a disjoint collection of single vertices
Parent[*i*] ← – 1
endfor
j ← 0
while *Size* ≤ *n* – 1 **.and.** *j* < *m* **do**
j ← *j* + 1
Find2(*Parent*[0:*n* – 1],u_j,*r*) //$e_j = u_j v_j$
Find2(*Parent*[0:*n* – 1],v_j,*s*)
if *r* ≠ *s* **then** //add edge e_j to *Forest*
Forest ← *Forest* ∪{e_j}
Size ← *Size* + 1
Union(*Parent*[0:*n* – 1],*r*,*s*) //combine sets containing u_j and v_j
endif
endwhile
MSTree ← *Forest*
end *Kruskal*

Complexity Analysis of Kruskal's algorithm

Sorting the edges in order of their weights using an algorithm such as Mergesort or Heapsort, which have worst-case complexity $\Theta(m \log n)$. Note that since m is less than n^2 (i.e., the complete graph has $n(n-1)/2$ edges) $\log m \leq \log n^2 = 2 \log n$. Thus,

$$\Theta(m \log m) = \Theta(m \log n).$$

Instead of sorting a priority queue implement using a **min-heap** can be used. The worst-case complexity over the entire algorithm of choosing the next smallest edge is still $\Theta(m \log n)$.

Complexity Analysis cont'd

- **Testing for cycles using the Disjoint Set ADT** takes time $O(m \quad (m,n))$ where $\alpha(m,n)$ is the inverse Ackermann function. See textbook pp. 189-190. $\alpha(m,n)$ is slower growing than log n, so that $O(m \quad (m,n)) \subset O(m \log n)$.
- It follows that Kruskal's algorithm has worst-case complexity

$$W(m,n) \in \Theta(m \log n) .$$

- Note that the complete graph K_n realizes the worst-case when only one parameter n is uses in which case we have

$$W(n) \quad \Theta(n^2 \log n) .$$

PSN. Give transformations showing

a) The MST problem for general weights can be reduced to the MST problem for positive weights.

b) The MST problem with some weights repeated can be reduced to the MST problem where all the weights are disjoint.

Proof of Correctness of Kruskal's Algorithm

We may assume **without loss of generality** that the edge weights are **distinct**.

As usual we use Proof by Contradiction.

Assume the tree K computed by Kruskal's algorithm is not a MST.

Let $x_1, x_2, \ldots, x_{n-1}$ be its edges listed in increasing value of their weights, i.e.,

$$w(x_1) < w(x_2) < \cdots < w(x_{n-1})$$

Let T be a MST with edges $y_1, y_2, \ldots, y_{n-1}$ listed in increasing value of their weights, i.e.,

$$w(y_1) < w(y_2) < \cdots < w(y_{n-1})$$

Let *j* be the first index where K and T disagree, i.e.,

$$x_i = y_i, i = 1, \ldots, j-1 \text{ and } x_j \neq y_j.$$

Since x_j was chosen to be the next smallest edge by the Greedy Method

$$w(x_j) < w(y_j)$$

Correctness Proof cont'd

- Consider the subgraph $T + x_j$, i.e., the subgraph obtained by adding x_j to the tree T.
- It is easily verified that $T + x_j$ contains a cycle C.
- Further, one of the edges y_k has index at least j, i.e., $j \leq k$. Otherwise, all the edges of C would have indices strictly less than j, so they would all belong to K, since T agrees with K in the first $j - 1$ edges. But, this would mean that K contains the cycle C, a contradiction.
- Let $T' = T + x_j - y_k$ be the subtree obtained from T by adding edge x_j and removing edge y_k . It is easily verified that T' is a spanning tree.

Conclusion of Proof

Now,

$$w(T') = w(T) + w(x_j) - w(y_k).$$

But

$$w(x_j) < w(y_j) \leq w(y_k)$$

Therefore,

$$w(T') < w(T).$$

This contradicts our assumption that T is minimum spanning tree. Q.E.D.

What did the beaver say to the tree?

It’s been nice gnawing you!