CSCI 136
Data Structures &
Advanced Programming

Lecture 31

Fall 2017

Instructors: Bills

Last Time

- Greedy Algorithms for Optimization
- Lab 11 : Exam Scheduling
- Adjacency List Implementation Details

Today's Outline

- GraphList Time/Space Complexity
- An Important Algorithm: Minimum-cost spanning subgraph
- More Core Algorithms: Directed Graphs
  - Dijkstra's Algorithm
  - Time permitting
    - Cycle Detection
    - Topological Sorting

Efficiency Revisited

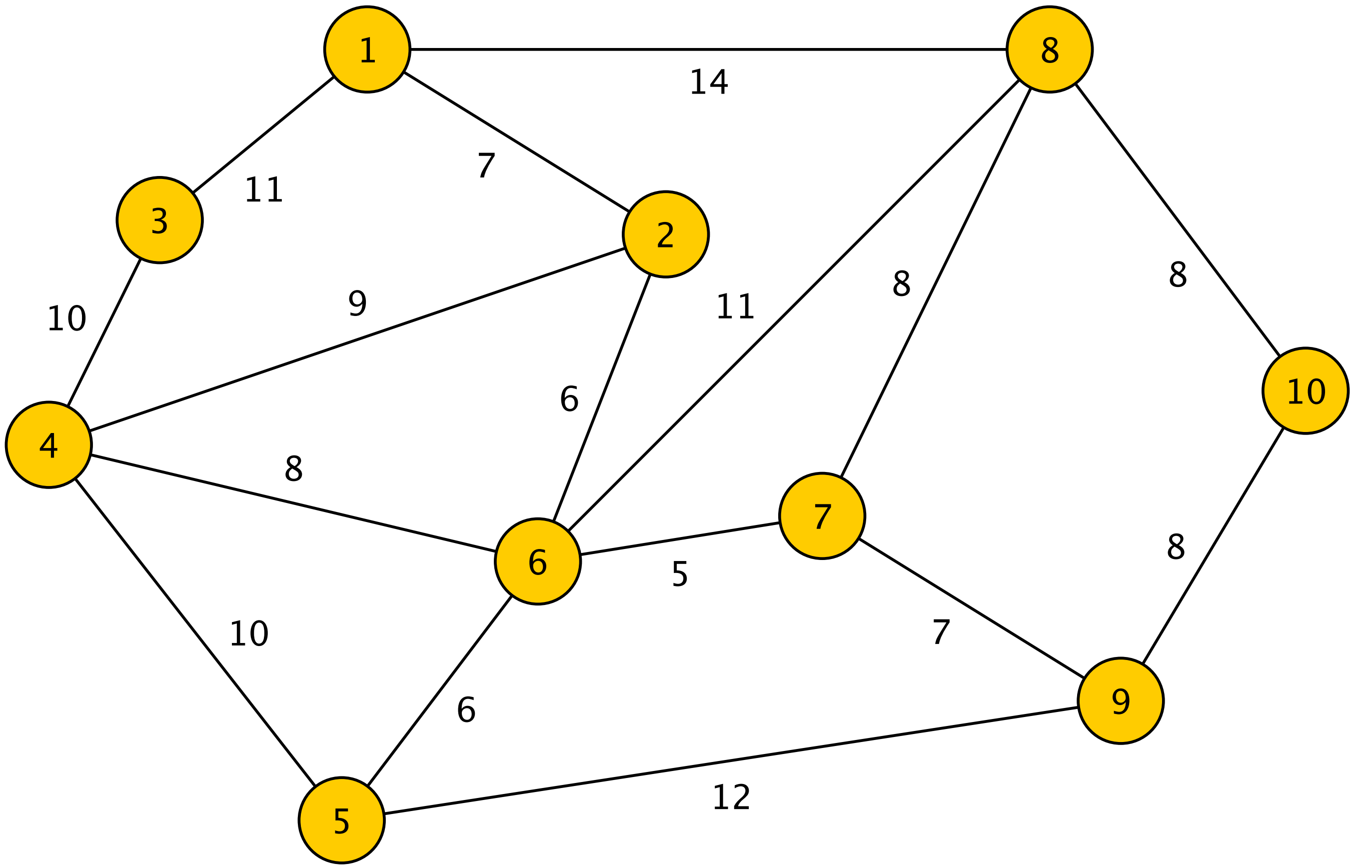
- Assume Map operations are O(1) (for now)
  - |E| = number of edges
  - |V| = number of vertices
- Runtime of add, addEdge, getEdge, removeEdge, remove?
- Space usage?
- Conclusions
  - Matrix is better for dense graphs
  - List is better for sparse graphs
  - For graphs "in the middle" there is no clear winner

Efficiency : Assuming Fast Map

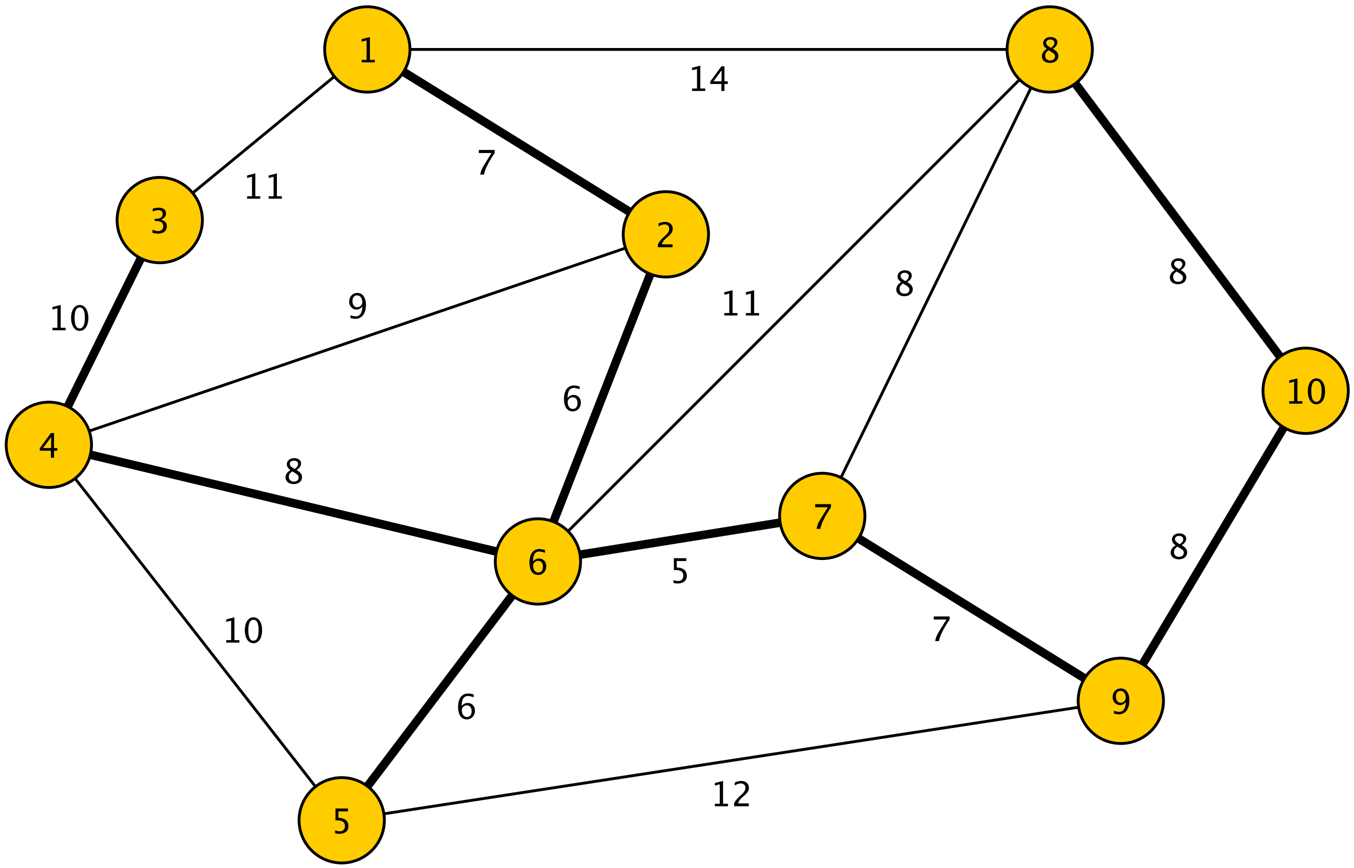
	Matrix	GraphList
add	O(1)	O(1)
addEdge	O(1)	O(1)
getEdge	O(1)	O(|V|)
removeEdge	O(1)	O(|V|)
remove	O(|V|)	O(|E|)
space	O(|V|^2)	O(|V|+|E|)

Applications

Minimum-Cost Spanning Trees



Minimum-Cost Spanning Trees



Basic Graph Properties

- A *subgraph* of a graph G=(V, E) is a graph G'=(V',E') where
  - V' $\subseteq$ V
  - E' $\subseteq$ E, and
  - If e $\in$ E' where e = {u,v}, then u, v $\in$ V'
- If E' contains *every* edge of E having both ends in V', then G' is called the subgraph of G *induced by* V'
- If V' = V, then G' is called a *spanning subgraph of G*

Basic Graph Properties

- Recall: An undirected graph G=(V,E) is *connected* if for every pair u,v in V, there is a path from u to v (and so from v to u)
- The maximal sized connected subgraphs of G are called its *connected components*
  - Note: They are induced subgraphs of G
- An undirected graph without cycles is a *forest*
- A connected forest is called a *tree.*
  - Not to be confused with the data structure!

Facts About Graphs

Thm: If G=(V,E) is a forest with |E| > 0, then G has at least one vertex v of degree 1 (a *leaf*)

- Let's prove this: Consider a longest simple path in G…

Thm: If G=(V,E) is a tree then |E| = |V| - 1.

- Hint: Induction on v: delete a leaf

Thm: Every connected graph G=(V,E) contains a spanning subgraph G'=(V,E') that is a tree

- That is, a *spanning tree*

Proof idea:

- If G is not a tree, then it contains a cycle C
- Removing an edge from C leaves G connected (why)
- Repeat until no more cycles remain

Edge-Weighted Graphs

- An *edge-weighting* of a graph G=(V,E) is an assignment of a number (weight) to each edge of G
  - We write the weight of e as w(e) or w_e
- The weight w(G') of any subgraph G' of G is the sum of the weights of the edges in G'
- We will focus on edge-weights that are non-negative, so if G' is a subgraph of G, then w(G')≤w(G)

A Famous Problem

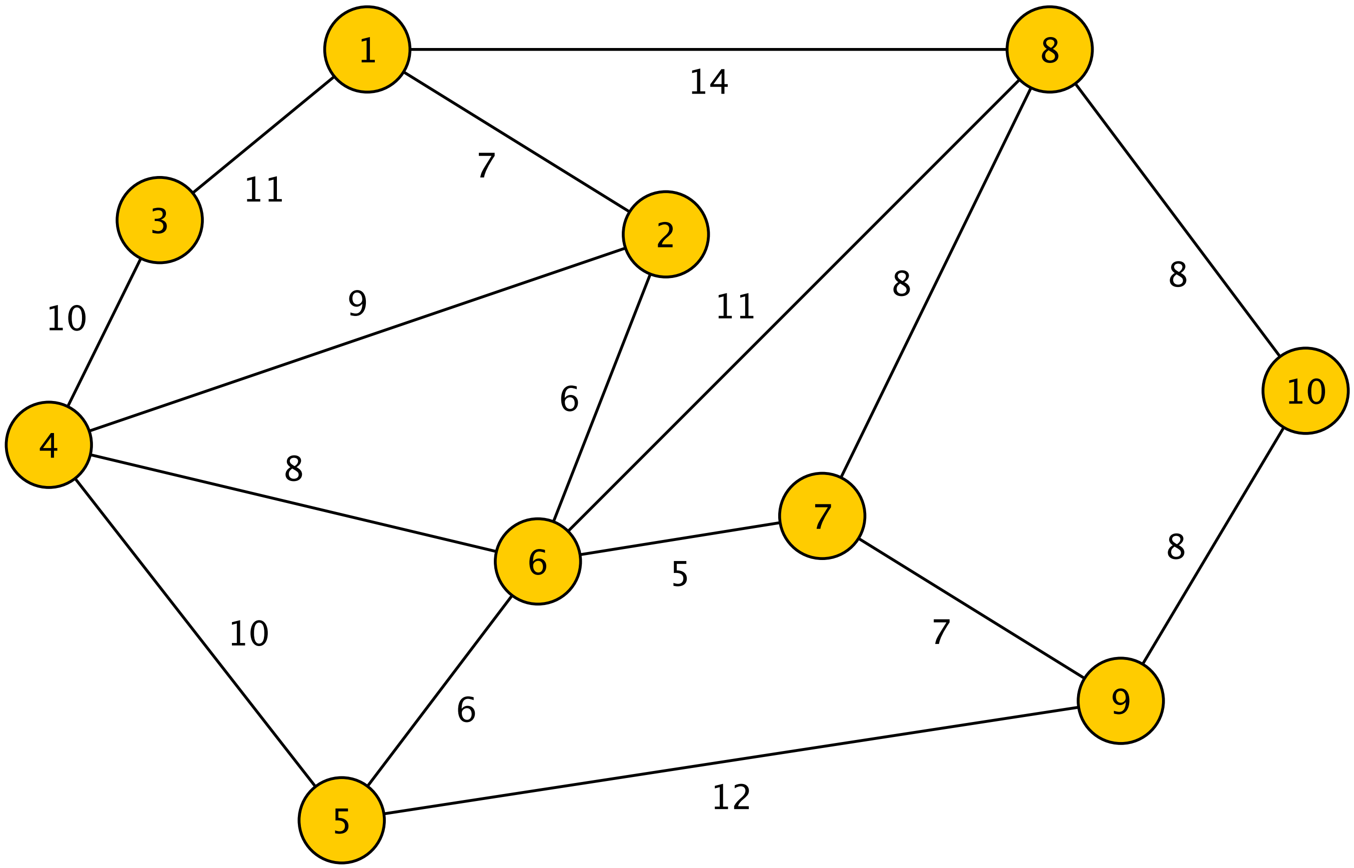
Given a connected, undirected graph G=(V,E) with non-negative edge weights, find a minimum-weight, connected, spanning subgraph of G.

Note: Such a subgraph must be a spanning tree!

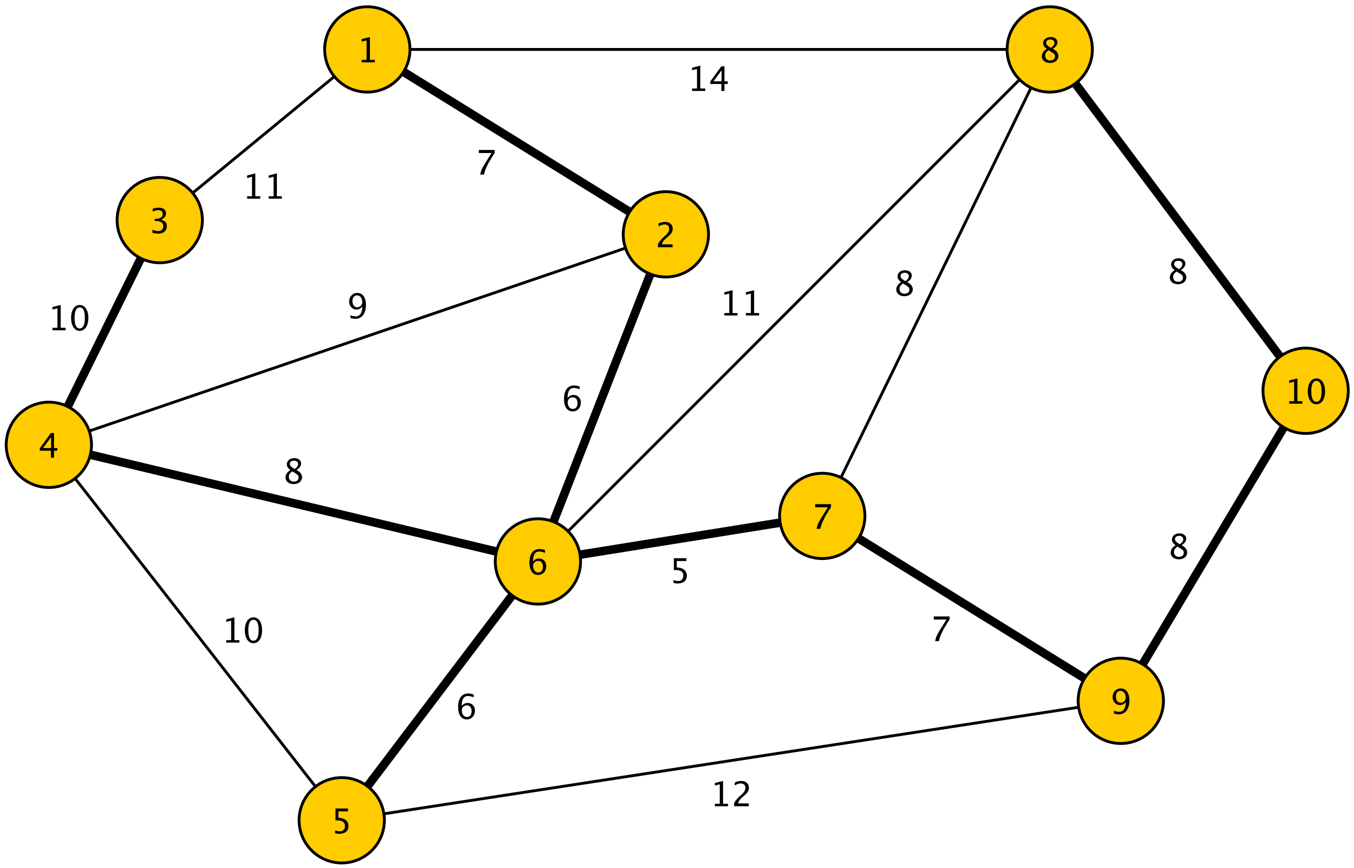
Frequently, we refer to the edge weights as *costs* and so this problem becomes:

Given an undirected graph G with edge costs, compute a minimum-cost spanning tree of G.

Minimum-Cost Spanning Trees



Minimum-Cost Spanning Trees



Finding a MCST

Suppose we just wanted to find a PCST (pretty cheap spanning tree), here's one idea:

Grow It Greedily!

- Pick a vertex and find its cheapest incident edge. Now we have a (small) tree
- Repeatedly add the cheapest edge to the tree that keeps it a tree (connected, no cycles)
- This method is called *Prim's Algorithm*
- How close might this get us to the MCST?

An Amazing Fact

Thm: (Prim 1957) The greedy tree-growing algorithm always finds a minimum-cost spanning tree for any connected graph.

Contrast this with the greedy exam scheduling algorithm, which does *not* always find a minimum schedule (coloring)

Why does this work?

The Key

Def: Sets V_1 and V_2 form a *partition* of a set V if

$$V_1 \cup V_2 = V \text{ and } V_1 \cap V_2 = \varnothing$$

Lemma: Let G=(V,E) be a connected graph and let V_1 and V_2 be a partition of V. *Every* MCST of G contains a cheapest edge between V_1 and V_2

- Let e be a cheapest edge between V_1 and V_2
- Let T be a MCST of G. If $e \notin T$, then $T \cup \{e\}$ contains a cycle C and e is an edge of C
- Some other edge e' of C must also be between V_1 and V_2; e is a cheapest edge, so w(e') = w(e) [Why?]

Using The Key to Prove Prim

We'll assume all edge costs are distinct

Otherwise proof is slightly less elegant

Let T be a tree produced by the greedy algorithm and suppose T* is a MCST for G

Claim: T = T*

Idea of Proof: Show that every edge added to the tree T by the greedy algorithm is in T*

Clearly the first edge added to T is in T*

Why? Use the key!

Using The Key

Now use induction!

- Suppose, for some $k \geq 1$, that the first k edges added to T are in T^*. These form a tree T_k
- Let V_1 be the vertices of T_k and let $V_2 = V - V_1$
- Now, the greedy algorithm will add to T the cheapest edge e between V_1 and V_2
- But any MCST contains the (only!) cheapest edge between V_1 and V_2, so e is in T^*
- Thus the first $k+1$ edges of T are in T^*