Minimum Spanning Trees

Connecting a Graph

Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

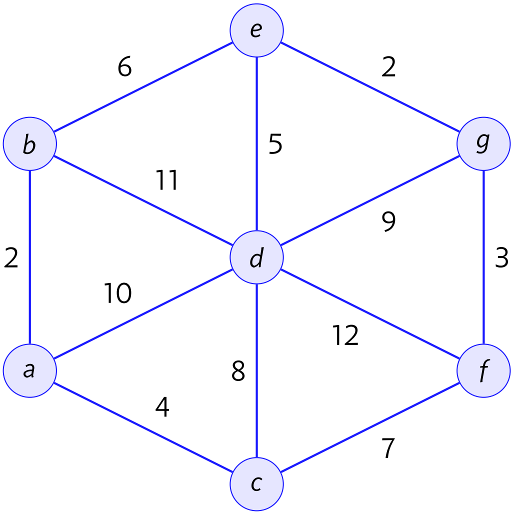
Fall, 2022

Outline

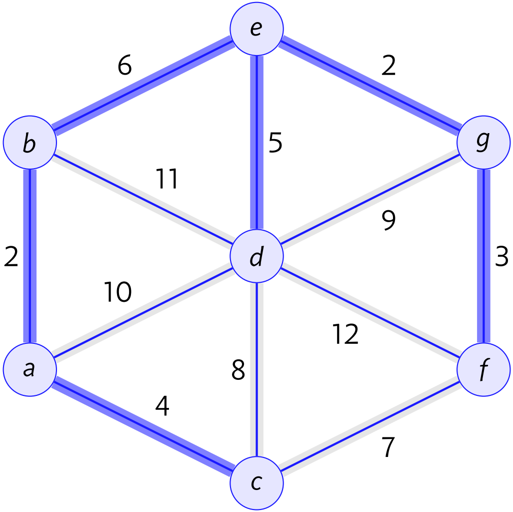
Objectives

- Define minimum spanning tree and explain its properties
- List some problems that MSTs solve
- Explain some variations of the MST
- Implement Kruscal's Algorithm

A Minimum Spanning Tree



A Minimum Spanning Tree



Properties of MSTs

- All connected graphs have one.
- May have more than one.
- $|E| = |V| - 1$
- Other tree properties hold…

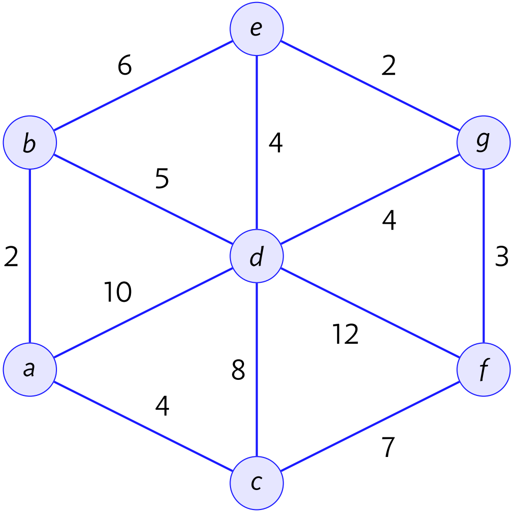
Variations

- Maximum spanning tree
- "minimum" spanning subgraph
- Minimum spanning forest
- Second minimum spanning tree: Compute MST, then try again $|E|$ times, removing a different edge from the MST each time.
- In contests: this algorithm is easy, so contest problems will try to disguise the fact that MST will solve it.

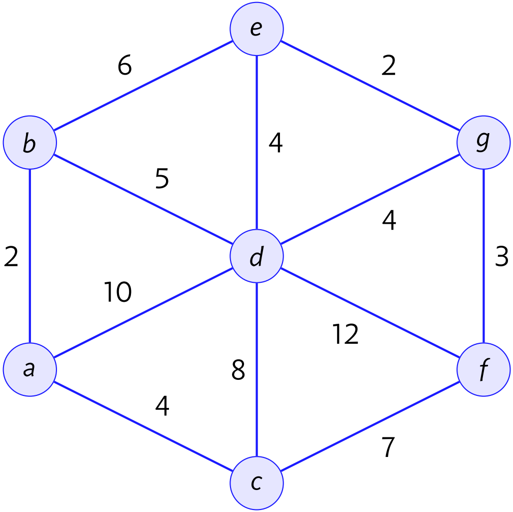
Outline

- Insert all edges into a priority queue
- Initialize a disjoint set with all the edges
- While there are fewer than $|V| - 1$ edges in your MST:
  - Dequeue an edge.
  - If the incident vertices are not both part of the MST already, add the edge. (Use the disjoint set to keep track)

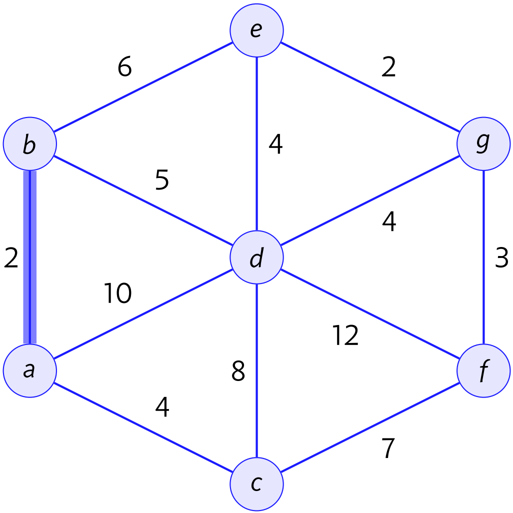
Example Run



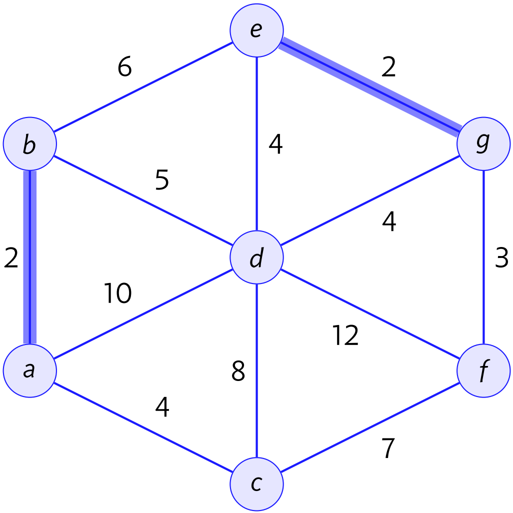
Example Run



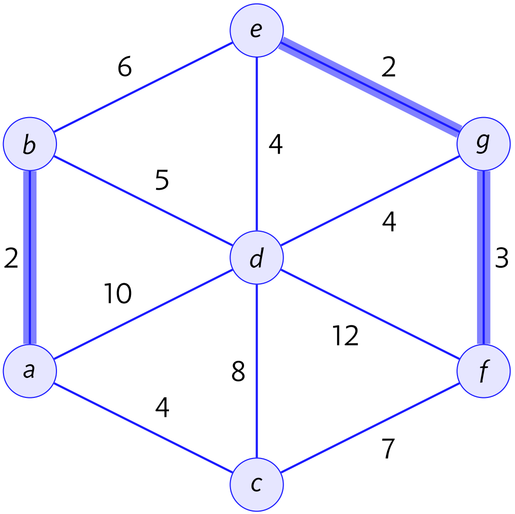
Example Run



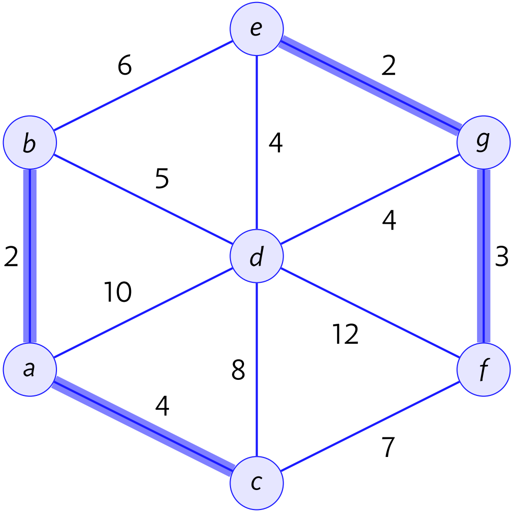
Example Run



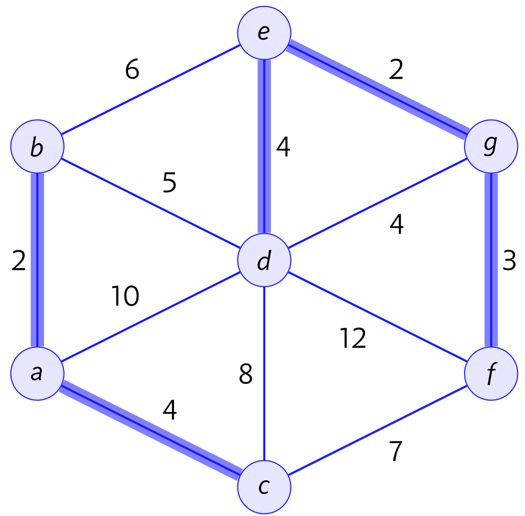
Example Run



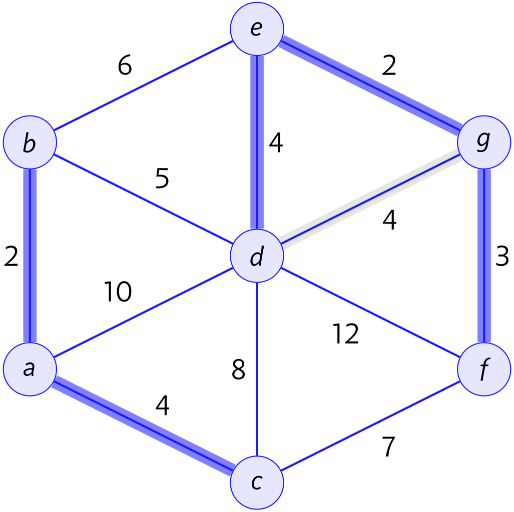
Example Run



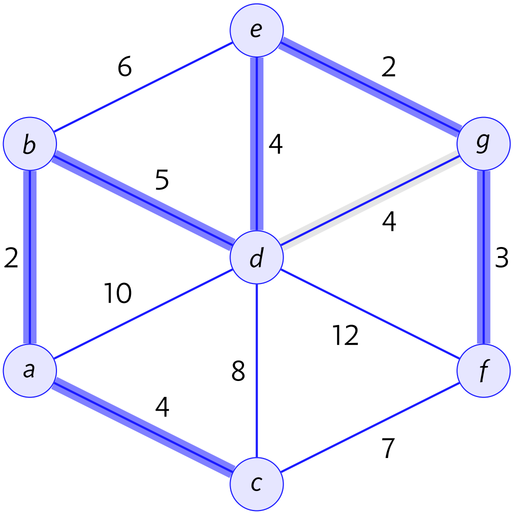
Example Run



Example Run



Example Run



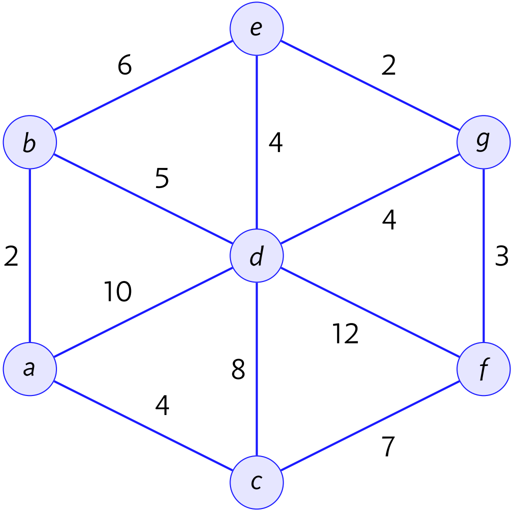
Code

```
vector< pair<int, ii> > EdgeList;
for (int i = 0; i < E; i++) {
    cin >> u >> v >> w;
    EdgeList.push_back(make_pair(w, ii(u, v)));
}
sort(EdgeList.begin(), EdgeList.end());
int mst_cost = 0;
UnionFind UF(V);
for (int i = 0; i < E; i++) {
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first
                      ,front.second.second)) {
        mst_cost += front.first;
        UF.unionSet(front.second.first, front.second.second)
} }
```

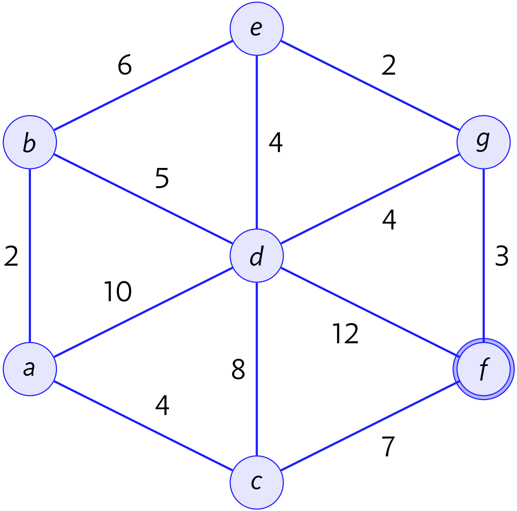
Idea

- In Kruscal's algorithm, you add edges by order of weight.
- In Prim's algorithm, you extend your current tree by adding a least-cost node.

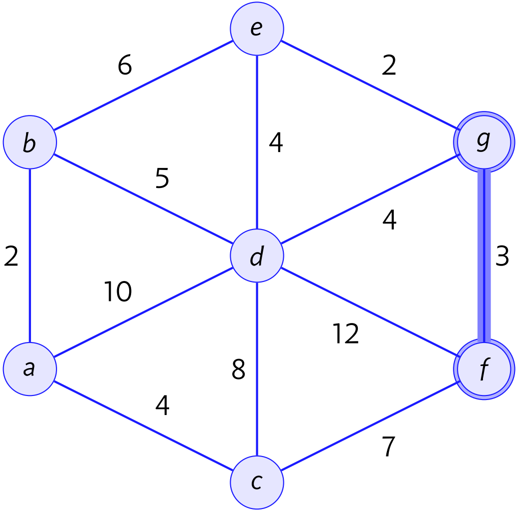
Example Run



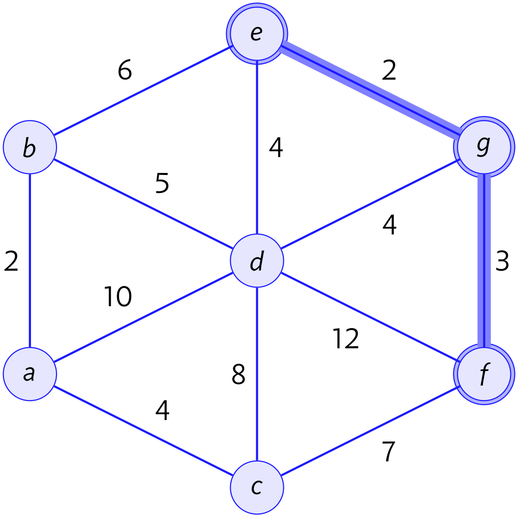
Example Run



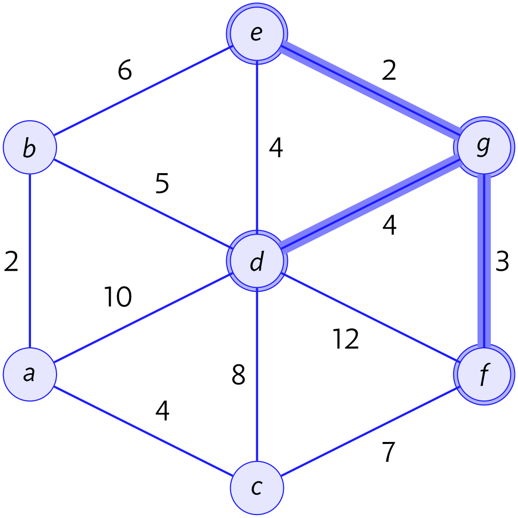
Example Run



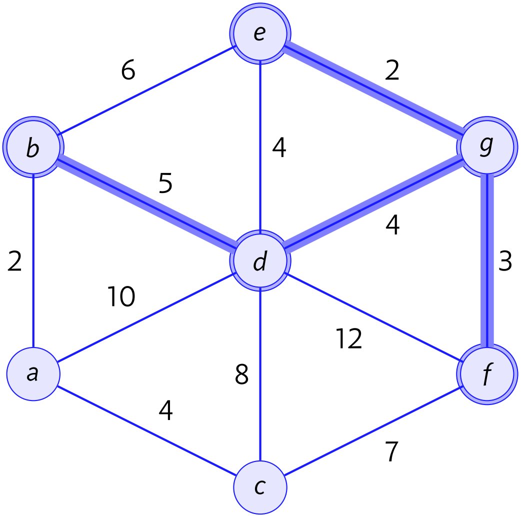
Example Run



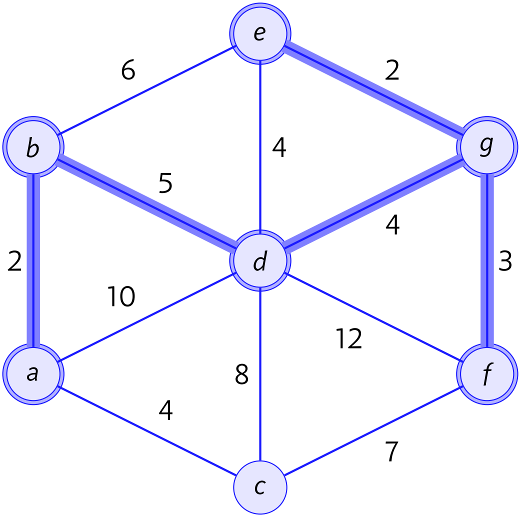
Example Run



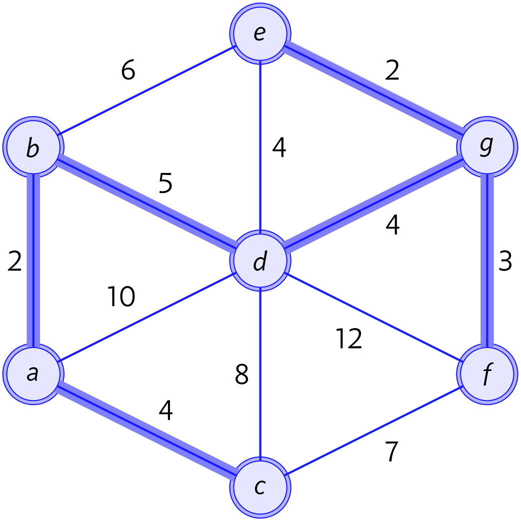
Example Run



Example Run



Example Run



Code from CP book 4

```
vector<vii> AL; // the graph stored in AL
vi taken; // to avoid cycle
priority_queue<ii> pq; // to select shorter edges
// C++ STL priority_queue is a max heap, we use -ve sign to
void process(int u) { // set u as taken and enqueue neighbo
  taken[u] = 1;
  for (auto &[v, w] : AL[u])
    if (!taken[v]) pq.emplace(-w, -v); // sort by non-dec w
}
```

Code, part 2

- in main, setting up...

```
int V, E;
cin >> V >> E;
AL.assign(V, vii());
for (int i = 0; i < E; ++i) {
  int u, v, w;
  cin >> u >> v >> w; // read as (u, v, w)
  AL[u].emplace_back(v, w); AL[v].emplace_back(u, w);
}
taken.assign(V, 0); // no vertex is taken
```

Code, part 3

- The main loop

```
process(0); // take+process vertex 0
int mst_cost = 0, num_taken = 0; // no edge has been taken
while (!pq.empty()) { // up to O(E)
  auto [w, u] = pq.top(); pq.pop(); // C++17 style
  w = -w; u = -u; // negate to reverse order
  if (taken[u]) continue; // already taken, skipped
  mst_cost += w; // add w of this edge
  process(u); // take+process vertex u
  ++num_taken; // 1 more edge is taken
  if (num_taken == V-1) break; // optimization
}
cout << "MST cost = " << mst_cost << " (Prim's)" << endl;
```