CSE 100
Minimum Spanning Trees
Union-find
Announcements

• Midterm grades posted on gradesource
• Collect your paper at the end of class or during the times mentioned on Piazza.
• Re-grades
  ‣ Submit your regrade request in writing on Wed during Lecture or during office hours
Q1: (True or False): Some spanning trees may have cycles

A. True

B. False
Q2: Cycle checks are required in the implementation of which of the following algorithms

A. Depth First Search
B. Breadth First Search
C. Dijkstra’s Shortest Path
D. Kruskal’s Minimum Spanning Tree
Q3: In the implementation of the union operation provided in the textbook, what does the array named ‘root’ keep track of?

A. The set that each vertex belongs to
B. The number of elements in a particular set
C. The sets with exactly one vertex
Q4: In the implementation of the union operation provided in the textbook, merging two distinct sets involves updating the 'root' value of vertices of which set?

A. The set with the larger number of vertices
B. The set with the smaller number of vertices
C. Size of the set is not considered
Spanning trees

• We will consider spanning trees for undirected graphs
• A spanning tree of an undirected graph $G$ is an undirected graph that...
  • contains all the vertices of $G$
  • contains only some edges of $G$
  • has no cycles
  • is connected

• A spanning tree is called “spanning” because it connects all the graph’s vertices

• A spanning tree is called a “tree” because it has no cycles (recall the definition of cycle for undirected graphs)

• What is the root of the spanning tree?
  • you could pick any vertex as the root; the vertices adjacent to that one are then the children of the root; etc.
Spanning trees: examples

• Consider this undirected graph $G$: 

```
V0 ---- V1
  |    |
  |    |
  V2 ---- V3
  |    |
  |    |
  V5 ---- V6
```

$V0$, $V1$, $V2$, $V3$, $V4$, $V5$, $V6$
Spanning tree? Ex. 1

Is this graph a spanning tree of G?

A. Yes
B. No because it has a cycle
Spanning tree? Ex. 2

Is this graph a spanning tree of $G$?

A. Yes  
B. No
Is this graph a spanning tree of G?

A. Yes  
B. No because it is disconnected
Minimum Spanning tree: Spanning tree with minimum total cost

Cost = Sum of the weights of all the edges

\[ 10 + 1 + 8 + 1 + 4 + 3 = 27 \]

Is this graph a minimum spanning tree of G?

A. Yes
B. No; we can construct a lower cost spanning tree by replacing the edge \((v_0, v_1)\) by \((v_2, v_0)\).
Minimum spanning trees in a weighted graph

• All spanning trees have the same number of edges, but if it is a weighted graph, they may differ in the total weight of their edges.

• Of all spanning trees in a weighted graph, one with the least total weight is a minimum spanning tree (MST).

• It can be useful to find a minimum spanning tree for a graph: this is the least-cost version of the graph that is still connected, i.e. that has a path between every pair of vertices.

• How to do it? Prim's or Kruskal's.

\[|E| \geq |V| - 1\]

In general, if a graph is connected, \(|E| \geq |V| - 1\).

A spanning tree has the min. no. of edges required for it to be connected: \(|E| = |V| - 1\).
Prim’s MST Algorithm

- Start with any vertex and grow like a mold, one edge at a time
- Each iteration choose the cheapest crossing edge
Prim’s MST Algorithm vs Dijkstra’s Shortest Path

Q: What is the major difference between the implementation of Prim’s Algorithm and Dijkstra’s shortest path algorithm

A. In Prim’s, among the edges that cross the frontier, we select those with minimum edge weight while in Dijkstra’s we select the edge with the minimum score (distance from source)

B. They need different supporting data structures for a fast implementation
Fast Implementation of Prim’s MST Algorithm

1. Initialization:
   a. Create an empty graph T.
   b. Initialize the vertex vector for the graph. Set all “done” fields to false. Pick an arbitrary start vertex s. Set its “done” field to true.

2. Iterate through the adjacency list of s, and put those edges in the priority queue.

3. While the priority queue is not empty or until the done flag of all vertices are set:
   • Remove from the priority queue the edge \((v, w, \text{cost})\) with the smallest cost.
   • Is the “done” field of the vertex \(w\) marked true?
     • If Yes: this edge connects two vertices already connected in the spanning tree, and we cannot use it. Go to 3.
     • Else accept the edge:
       • Mark the “done” field of vertex \(w\) true, and add the edge \((v, w)\) to the spanning tree \(T\).
       • Iterate through \(w\)’s adjacency list, putting each edge that connects \(w\) to a vertex whose done flag is not set into the priority queue.
   • The resulting spanning tree \(T\) is the minimum spanning tree.
Finding a minimum spanning tree: Prim’s algorithm

• As we know, minimum weight paths from a start vertex can be found using Djikstra’s algorithm

• At each stage, Djikstra’s algorithm extends the best path from the start vertex (priority queue ordered by total path cost) by adding edges to it

• To build a minimum spanning tree, you can modify Djikstra’s algorithm slightly to get Prim’s algorithm

• At each stage, Prim’s algorithm adds the edge that has the least cost from any vertex in the spanning tree being built so far (priority queue ordered by single edge cost)

• Like Djikstra’s algorithm, Prim’s algorithm has worst-case time cost O(|E| log |V|)

• We will look at another algorithm: Kruskal’s algorithm, which also is a simple greedy algorithm

• Kruskal’s has the same big-O worst case time cost as Prim’s, but in practice it can be made to run faster than Prim’s, if efficient supporting data structures are used
Weighted minimum spanning tree: Kruskal’s algorithm

• Prim’s algorithm starts with a single vertex, and grows it by adding edges until the MST is built: it builds the MST like growing a mold.

• Kruskal’s algorithm starts with a forest of single-node trees (one for each vertex in the graph) and joins them together by adding edges until the MST is built;

Why Kruskal? New data structures → union-find
Kruskal’s algorithm:
Naïve Implementation of Kruskal’s algorithm:

1. Sort edges in increasing order of cost (priority queue)

2. Set of edges in MST, $T = \emptyset$

3. For $i = 1$ to $|E|$
   
   If $T \cup \{e_i\}$ has no cycles
   
   Add $e_i$ to $T$

Ref: Tim Roughgarden (stanford)
Naïve Implementation of Kruskal’s algorithm:

1. Sort edges in increasing order of cost
2. Set of edges in MST, \( T = \{ \} \)
3. For \( i = 1 \) to \( |E| \)
   
   If \( T \cup \{ e_i \} \) has no cycles
   
   Add \( e_i \) to \( T \)

What is the work that we repeatedly do in Kruskal’s Algo?

Ref: Tim Roughgarden (stanford)
Towards a fast implementation of Kruskal’s Algorithm

Q: Which of the following algorithms can be used to check if adding an edge (v, w) to an existing Graph creates a cycle?

A. DFS
B. BFS
C. Either A or B
D. None of the above
Running Time of Naïve Implementation of Kruskal’s algorithm:

1. Sort edges in increasing order of cost
   \[ T(\log_2 |E|) = O(1E \log_2 |V|) \]

2. Set of edges in MST, \( T = \{ \} \)
   \[ O(1) \]

3. For \( i = 1 \) to \( |E| \)
   - If \( T \cup \{e_i\} \) has no cycles
     - Add \( e_i \) to \( T \)
       \[ O(1) \]
   - Using BFS/DFS \[ O(|V| + |E|) \]
     - In worst case \[ |E_t| \leq (|V|-1) \]
       - Cost of cycle check \( = O(|V|) \)

Ref: Tim Roughgarden (stanford)
Towards a Fast Implementation of Kruskal’s MST Algorithm

• What is the work that we are repeatedly doing with Kruskal’s Algo?
  – Checking for cycles: Linear with BFS, DFS \( O(V) \)
  – Union-Find Data structure allows us to do this in near constant time! \( O(1) \)
Graph algorithm time costs: a summary

• The graph algorithms we have studied have fast algorithms:
  • Find shortest path in unweighted graphs
    • Solved by basic breadth-first search: $O(|V|+|E|)$ worst case
  • Find shortest path in weighted graphs
    • Solved by Dijkstra’s algorithm: $O(|E| \log|V|)$ worst case
  • Find minimum-cost spanning tree in weighted graphs
    • Solved by Prim’s or Kruskal’s algorithm: $O(|E| \log|V|)$ worst case

• The “greedy” algorithms used for solving these problems have polynomial time cost functions in the worst case
  • since $|E| \leq |V|^2$, Dijkstra’s, Prim’s and Kruskal’s algorithms are $O(|V|^3)$

• As a result, these problems can be solved in a reasonable amount of time, even for large graphs; they are considered to be ‘tractable’ problems

• However, many graph problems do not have any known polynomial time solutions...!