L25: Minimum Spanning Trees (MST) and Disjoint Sets

Alex Steiger
CompSci 201: Spring 2024
4/15/2024

Logistics, coming up

• This Wednesday, 4/17
  • Midterm exam 3
    • APT 9 (last APTs) - extended to Thursday 4/18
  • This Friday, 4/19
    • Semester / Final review in discussion
  • Next Monday, 4/22
    • Project P6: Route (last project) due
  • Tuesday after next, 4/30
    • Final exam, 9 am

Exploring a node with Dijkstra’s Algorithm, Pseudocode

While unexplored nodes remain
• Explore current = the closest unexplored node
• For each neighbor:
  • Update shortest path to neighbor if shorter to go through current

Just like BFS (explore closer nodes first) except... now we need to account for weights.
Practical Dijkstra Initialization

Add vertices to the queue once they are actually reached/visited.

Don’t need to add anything for all nodes yet.

Practical Dijkstra search loop

Keep searching while there are unexplored nodes.

Choose to explore from the next closest (to start) unexplored node to start at each iteration.

Details: Checking each neighbor

All neighbors of current node

Distance to neighbor through current = distance to current + weight on edge from current to neighbor

If neighbor newly discovered:
- Record new distance
- Add to priority queue

If neighbor already discovered, update:
- Remove from PQ
- Record new distance
- Add back to PQ
Implementing `decreasePriority`

- Most standard library binary heaps (including java.util) don’t support an efficient update/decrease priority operation.

```java
else if (newDist < distance.get(neighbor)) {
  // implement decreasePriority by removal and re-insertion
tobeprocessed.remove(neighbor);
distance.put(neighbor, newDist);
tobeprocessed.add(neighbor);
}
```

- Our code works, but is O(N) time
  - Java’s PQ takes O(N) to remove a given node (O(log N) for smallest)
  - Other PQ implementations support O(log N)-time `decreasePriority`, but they are not in Java library

---

Is Dijkstra’s algorithm guaranteed to be correct? (Informal)

- **Claim.** Distance is correct shortest path distance for all nodes explored so far, and shortest path distance through explored nodes for all others.

- Formal proof is **by induction**, see CompSci 230.
  - Assume the property is true up to some point in the algorithm, then:
    - Consider the next node we explore:

```plaintext
Is Dijkstra’s algorithm guaranteed to be correct? (Informal)
```

Suppose we explore from `C` this iteration

The shortest path distance so far goes through explored nodes

```
can’t be another shorter path through an unexplored node! There would be a node that would be explored/removed from the PQ before C.
```

<table>
<thead>
<tr>
<th>Start</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>Explored nodes</th>
</tr>
</thead>
</table>

The shortest path distance so far goes through explored nodes

Putting edge weights in parentheses
L25-WOTO1-Dijkstra-Sp24

Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

1

NetID *

solutions

2

At each iteration, Dijkstra's algorithm will explore the next unexplored node that... *
Suppose every edge in the graph has equal weight. Then Dijkstra's algorithm would... *

- Explore nodes in the order of a depth-first search (DFS)
- Explore nodes in the order of a breadth-first search (BFS)
- Explore nodes in an order different than DFS or BFS

The best explanation of the if statement on line 39 is... *
If neighbor is newly discovered or is closer to start than current

If neighbor is newly discovered or the path through current to neighbor is shorter

If neighbor is newly discovered

If neighbor is not in the graph or is closer to start than current

If neighbor is not in the graph or the path through current to neighbor is shorter

If the path through current to neighbor is shorter

The best explanation of the if statement on line 43 is...
for (char neighbor : aList.get(current)) {
    int newDist = currDist + getWeight(current, neighbor);
    if (!distance.containsKey(neighbor)) {
        distance.put(neighbor, newDist);
        toExplore.add(neighbor);
    } else if (newDist < distance.get(neighbor)) {
        // implement decreasePriority by removal, update prio, then reinsert
        toExplore.remove(neighbor);
        distance.put(neighbor, newDist);
        toExplore.add(neighbor);
    }
}
6

Consider the weighted undirected graph diagrammed with edges labeled by their weight.

Starting from A, in what order will Dijkstra's algorithm explore (remove from the priority queue) the nodes in the graph? *

- A, B, C, D, E, F
- A, B, C, F, E, D
- A, B, D, C, E, F
- A, B, D, E, C, F

7

Again starting from A in the same graph, which path from A to E will Dijkstra's algorithm record as the shortest path? Hint: See the if statements above in problems 4-5. *

- A, B, E
- A, D, E
Might be either, depends on tie breaking in the priority queue
Runtime Complexity of Dijkstra’s Algorithm (with N nodes, M edges) assuming $O(\log N)$ decreasePriority

Like BFS, consider each node once and each edge twice, takes $O(\log N)$ time for each: $O((N+M)\log(N))$

Minimum Spanning Tree (MST) and Greedy Graph Algorithms

Minimum Spanning Tree (MST) Problem

• Given N nodes and M edges, each with a weight/cost...
• Find a set of edges that connect all the nodes with minimum total cost (will be a tree)
Motivating/Applying Minimum Spanning Tree

• Create a connected cable/data network with the least cable/cost/energy possible.

• City planning: Connect several metro stops with least tunneling.

• Image Segmentation

• Clustering

Example MST Problem

leetcode.com/problems/min-cost-to-connect-all-points

You are given an array points representing integer coordinates of some points on a 2D plane, where points[i] = [xi, yi].

The cost of connecting two points [xi, yi] and [xj, yj] is the manhattan distance between them: |xi - xj| + |yi - yj|, where |val| denotes the absolute value of val.

Return the minimum cost to make all points connected. All points are connected if there is exactly one simple path between any two points.

Intuitive Inductive Reasoning

• Suppose we have the MST on N-1 vertices.

• We consider the next vertex to get the MST on N vertices.

• Must use the cost 2 or the cost 5 edge regardless of the rest of the MST.

• Might as well use the cheaper cost 2 edge.
Greedy Optimization: Prim’s Algorithm

- Initialize?
  - Choose an arbitrary vertex

- Partial solution?
  - MST connecting subset of the vertices.

- Greedy step?
  - Choose the cheapest / least weight edge that connects a new vertex to the partial solution.

Visualizing Prim’s Algorithm

In the visualization:
- Edges between all pairs of vertices
- Weights are implicit by distances
- Algorithm greedily grows by choosing closest unconnected vertex

More Intuitive Inductive Reasoning

- Suppose we have chosen some spanning trees so far.
- Must connect all of them, might as well choose the cheapest edge connecting two trees.
Greedy Optimization Again: Kruskal’s Algorithm

- Initialize?
  - All nodes in *disjoint sets*

- Partial solution?
  - Forest of spanning trees in disjoint sets

- Greedy step?
  - Choose the cheapest / least weight edge that connects two disjoint sets / trees, connect them.

Visualizing Kruskal’s Algorithm

In the visualization:
- Edges between all pairs of vertices
- Weights are implicit by distances
- Algorithm greedily grows by cheapest edge that connects disjoint sets/trees.

Kruskal’s Algorithm in Pseudocode

Input: N node, M edges, M edge weights
- Initialize MST as empty set
- Let S be a collection of N disjoint sets, one per node
- While S has more than 1 set:
  - Let (u, v) be the minimum cost remaining edge
  - Find which sets u and v are in. If different sets:
    - Union the sets together
    - Add (u, v) to MST
- Return MST
Kruskal’s Algorithm Runtime?

Input: N node, M edges, M edge weights
- Initialize MST as empty set
- Let S be a collection of N disjoint sets, one per node
- While S has more than 1 set:
  - Let (u, v) be the minimum cost remaining edge
  - Find which sets u and v are in. If different sets:
    - Union the sets together
    - Add (u, v) to MST
  - Return MST

Looping over (worst case) all M edges:
Remove from binary heap, \(O(\log(M))\)

Overall: \(O(M(\log(M)+C))\) where \(C\) is time for Union/Find

Disjoint Sets and Union-Find

- AKA Disjoint-Set Data Structure
- Start with N distinct (disjoint) sets
  - Consider them labeled by integers: 0, 1, ...
- Union two sets: create set containing both
  - Label with one of the numbers
- Find the set containing a number
  - Initially self, but changes after unions

DIYDisjointSets implementation viewable here:
coursework.cs.duke.edu/cs-201-spring-24/diydisjointsets

Union-Find Data Structure
Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

1

NetID *

solutions
For the weighted undirected graph pictured below, what is the total cost of the minimum spanning tree?  
*  

- 3  
- 5  
- 8  
- 10  

Suppose we are running Prim's algorithm, and we have so far selected the edge between nodes 3 and 4 shown in bolded red. What edge will we greedily select next?  
*  

- Edge between nodes 0 and 1
Suppose we are running Kruskal's algorithm. So far we have selected the edges shown in bolded green: (A, D) and (C, E).

How many disjoint sets/trees are there remaining at this point? * 4
Suppose we are running Kruskal's algorithm. So far we have selected the edges shown in bolded green: (A, D), (A, B), (D, F), and (C, E).

Which edge will the algorithm select next? *

- (B, C)
- (B, E)
- (D, E)
- (E, G)
- (F, E)

True or false: In a given iteration, Kruskal's algorithm will always select/add to the MST the next lowest weight edge.*

- True
False