Spanning trees

What we do today:
- Talk about modifying an existing algorithm
- Calculating the shortest path in Dijkstra's algorithm
- Minimum spanning trees
- 3 greedy algorithms (including Kruskal & Prim)

Assignment A7 available soon
Due close to prelim 2

Implement Dijkstra's shortest-path algorithm.
Start with our abstract algorithm, implement it in a specific setting. Our method: 36-40 lines, including extensive comments.
We give you all necessary test cases.
We will make our solution to A6 available after the deadline for late submissions.
Previous semester: median: 4.0, mean: 3.84. But our abstract algorithm is much closer to the planned implementation than during that semester, and we expect a much lower median and mean.

Backpointers

Shortest path requires not only the distance from start to a node but the shortest path itself. How to do that?
In the graph, red numbers are shortest distance from S.

S, 0
1

Backpointers

Shortest path requires not only the distance from start to a node but the shortest path itself. How to do that?
In the graph, red numbers are shortest distance from S.

S, 0
null

In each node, store (a pointer to) previous node on the shortest path from S to that node. Backpointer
Backpointers

When to set a backpointer? In the algorithm, processing an edge \((f, w)\): If the shortest distance to \(w\) changes, then set \(w\)’s backpointer to \(f\). It’s that easy!

Each iteration of Dijkstra’s algorithm

dist: shortest-path length calculated so far

\(f\): node in Frontier with min dist; Remove \(f\) from Frontier; for each neighbor \(w\) of \(f\):

- if \(w\) in far-off set
  - then \(w.\text{spl} = f.\text{dist} + \text{weight}(f, w)\); Put \(w\) in the Frontier;
- else if \(f.\text{dist} + \text{weight}(f, w) < w.\text{dist}\)
  - then \(w.\text{dist} = f.\text{dist} + \text{weight}(f, w)\);
  - \(w.\text{backPointer} = f\);

Undirected trees

- An undirected graph is a tree if there is exactly one simple path between any pair of vertices

Root of tree? It doesn’t matter. Choose any vertex for the root

Facts about trees

Consider a graph with these properties:

1. \(|E| = |V| - 1\)
2. connected
3. no cycles

Any two of these properties imply the third—and imply that the graph is a tree

A spanning tree of a connected undirected graph \((V, E)\) is a subgraph \((V, E')\) that is a tree

- Same set of vertices \(V\)
- \(E' \subseteq E\)
- \((V, E')\) is a tree

Spanning trees: examples

http://mathworld.wolfram.com/SpanningTree.html
Finding a spanning tree

A subtractive method

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles

Use: Maximal set of edges that contains no cycle

Minimum spanning trees

- Suppose edges are weighted (> 0), and we want a spanning tree of minimum cost (sum of edge weights)
- Some graphs have exactly one minimum spanning tree. Others have several trees with the same cost, any of which is a minimum spanning tree

Use: Minimal set of edges that connects all vertices

Finding a spanning tree: Additive method

- Start with no edges
- While the graph is not connected: Choose an edge that connects 2 connected components and add it – the graph still has no cycle (why?)

Use: Minimal set of edges that connects all vertices

Tree edges will be red.
Dashed lines show original edges.
Left tree consists of 5 connected components, each a node

Minimum spanning trees

- Suppose edges are weighted (> 0), and we want a spanning tree of minimum cost (sum of edge weights)
- Useful in network routing & other applications
- For example, to stream a video
Greedy algorithm

A greedy algorithm: follow the heuristic of making a locally optimal choice at each stage, with the hope of finding a global optimum.

Example. Make change using the fewest number of coins. Make change for \( n \) cents, \( n < 100 \) (i.e. < $1).

Greedy: At each step, choose the largest possible coin.

- If \( n \geq 50 \), choose a half dollar and reduce \( n \) by 50;
- If \( n \geq 25 \), choose a quarter and reduce \( n \) by 25;
- As long as \( n \geq 10 \), choose a dime and reduce \( n \) by 10;
- If \( n \geq 5 \), choose a nickel and reduce \( n \) by 5;
- Choose \( n \) pennies.

Greediness doesn’t work here

You’re standing at point \( x \), and your goal is to climb the highest mountain.

Two possible steps: down the hill or up the hill. The greedy step is to walk up hill. But that is a local optimum choice, not a global one. Greediness fails in this case.

Construct minimum spanning tree (greedy)

As long as there is a cycle:

Find a black max-weight edge – if it is on a cycle, throw it out otherwise keep it (make it red).

We mark a node red to indicate that we have looked at it and determined it can’t be removed because removing it would unconnect the graph (the node is not on a cycle).

Maximal set of edges that contains no cycle

Nondeterministic algorithm

Maximal set of edges that contains no cycle

Non more cycles: done
Construct minimum spanning tree (greedy)

As long as there is a cycle:
Find a black max-weight edge – if it is on a cycle, throw it out otherwise keep it (make it red)

Nobody uses this algorithm because, usually, there are far more edges than nodes. If graph with n nodes is complete, O(n^2) edges have to be deleted!

It’s better to use this property of a spanning tree and add edges to the spanning tree. For a tree with n nodes, n-1 edges have to be added.

Two greedy algorithms for constructing a minimum spanning tree

- **Kruskal**
- **Prim**

Both use this definition of a spanning tree and in a greedy fashion:

- Maximal set of edges that contains no cycle
- Minimal set of edges that connect all vertices

Both are nondeterministic, in that at a point they may choose one of several nodes with equal weight.

Kruskal’s algorithm: greedy

At each step, add an edge (that does not form a cycle) with minimum weight

One of the 4’s

Edge with weight 2

The 5

Edge with weight 3

Dashed edges: original graph

Red edges: the constructed spanning tree

Prim’s algorithm. greedy

Have start node.

Edge with weight 3

One of the 4’s

Edge with weight 5

Prim’s algorithm (n nodes, m edges)

prim(s) {
    D[s] = 0; //start vertex
    D[i] = ∞ for all i ≠ s;
    while (a vertex is unmarked) {
        v = unmarked vertex
        mark v;
        for (each w adj to v) {
            D[w] = min(D[w], c(v,w));
        }
    }
}

- O(m + n log n) for adj list
- Use a priority queue PQ
- Regular PQ produces time O(n + m log m)
- Can improve to O(m + n log n) using a fancier heap

- O(n^2) for adj matrix
  - while-loop iterates n times
  - for-loop takes O(n) time

Tree greedy spanning tree algorithms

1. Algorithm that uses this property of a spanning tree: Maximal set of edges that contains no cycle
2. Algorithms that use this property of a spanning tree: Minimal set of edges that connect all vertices
   (a) Kruskal   (b) Prim

When edge weights are all distinct, or if there is exactly one minimum spanning tree, all 3 algorithms construct the same tree.
Application of MST

Maze generation using Prim’s algorithm

More complicated maze generation

Greedy algorithms

- These are Greedy Algorithms
- Greedy Strategy: is an algorithm design technique
  - Like Divide & Conquer
  - Greedy algorithms are used to solve optimization problems
    - Goal: find the best solution
    - Works when the problem has the greedy-choice property:
      - A global optimum can be reached by making locally optimum choices

Example: Making change

Given an amount of money, find smallest number of coins to make that amount

Solution: Use Greedy Algorithm:

Use as many large coins as you can.

Produces optimum number of coins for US coin system

May fail for old UK system

Traveling salesman problem

Given a list of cities and the distances between each pair, what is the shortest route that visits each city exactly once and returns to the origin city?

- The true TSP is very hard (called NP complete)… for this we want the perfect answer in all cases.
- Most TSP algorithms start with a spanning tree, then “evolve” it into a TSP solution. Wikipedia has a lot of information about packages you can download…

Similar code structures

while (a vertex is unmarked) {
    v= best unmarked vertex
    mark v;
    for (each w adj to v)
        update D[w];
}

- Breadth-first-search (bfs)
  - best: next in queue
  - update: D[w] = D[v]+1
- Dijkstra’s algorithm
  - best: next in priority queue
  - update: D[w] = min(D[w], D[v] + c(v,w))
- Prim’s algorithm
  - best: next in priority queue
  - update: D[w] = min(D[w], c(v,w))

$c(v,w)$ is the $v \rightarrow w$ edge weight