Greedy Algorithms: Shortest Paths and Minimum Spanning Trees
CMSC 451, Summer 2009
Sorelle Friedler

Updates
- Survey says... we’re done with graph review

Review: Dijkstra’s Algorithm
Numb3rs Clip

Dijkstra’s Shortest Path Algorithm

Find shortest path from s to t.

Let S = {} and d(s) = 0
For each u in S, store a distance d(u)
While S ≠ V:
   Select a node v not in S with at least one edge from S to minimize d(v)
   Add v to S and set d(v) = \min_{u \in S} d(u) + l(e)

S = {}  \hspace{1cm} PQ = {s, 2, 3, 4, 5, 6, 7, t}
Dijkstra's Shortest Path Algorithm

$S = \{s\}$
PQ = \{2, 3, 4, 5, 6, 7, t\}

distance label

$S = \{s, 2\}$
PQ = \{3, 4, 5, 6, 7, t\}

distance label

$S = \{s, 2, 6\}$
PQ = \{3, 4, 5, 7, t\}

distance label
Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]

Dijkstra’s Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]

\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ PQ = \{ t \} \]

\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]

\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]

\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]
\[ \infty \]

Dijkstra's Algorithm: Proof of Correctness

**Invariant:** For each node \( v \in S \), \( d(v) \) is the length of the shortest \( s-v \) path.

(Applying for \( v=t \) immediately gives the proof of optimality.)

**Proof:** (by induction on \( |S| \))

**Base case:** \(|S| = 1, d(s)=0, d(t)=0\), which is the shortest it could be.

**Inductive hypothesis:** Assume true for \(|S| = k \geq 1\).

**Induction step: Proof for \(|S|=k+1\)**

- Let \( v \) be next node added to \( S \), and let \( u-v \) be the chosen edge.
- The shortest \( s-u \) path plus \( (u,v) \) is an \( s-v \) path of length \( \pi(v) \).
- Consider any \( s-v \) path \( P \). We'll see that it's no shorter than \( \pi(v) \).
- Let \( x-y \) be the first edge in \( P \) that leaves \( S \), and let \( P' \) be the subpath to \( x \).
- \( P \) is already too long as soon as it leaves \( S \).
- \( d(y) \) is set to \( \pi(v) \).
- \( d(y) = \pi(v) \) is optimal.
- \( \pi(v) \leq \pi(y) \) (shown).

\[ \pi(y) \geq d(y) \]

\[ \pi(y) \geq d(x) + l(x,y) \geq \pi(x) \geq \pi(y) \]

\[ \pi(x) \geq \pi(y) \]

\[ \pi(x) \geq \pi(y) \geq \pi(y) \]

\[ \pi(x) \geq \pi(y) \geq \pi(y) \]
Dijkstra’s Algorithm Greedy Perspective

What is the “step” in our step-by-step creation of a solution?
What is the greedy choice being made?
Note on proof: The analysis was in the “stay ahead” form

Spanning Tree

Given an undirected, connected graph G:
A spanning tree of G is a tree containing all vertices of G and some subset of the edges of G.
• maximal subset of edges of G with no cycle
• minimal subset of edges of G that connect all vertices

Minimum Spanning Tree

Minimum spanning tree. Given a connected graph G = (V, E) with real-valued edge weights \( c_e \), an MST is a subset of the edges \( T \subseteq E \) such that T is a spanning tree whose sum of edge weights is minimized.

Cayley’s Theorem. There are \( n^{n-2} \) spanning trees of \( K_n \).

Applications

MST is fundamental problem with diverse applications.

• Network design. - telephone, electrical, hydraulic, TV cable, computer, road
• Cluster analysis.

Developing a Greedy Algorithm to Find a Minimum Spanning Tree

Options:
• Look at all nodes and consider their adjacent edges
• Look at all edges and their weights
Make greedy decisions to:
• Minimize total edge cost
• Maximize cost of edges not chosen

Greedy Algorithms

Kruskal’s algorithm. Start with T = φ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prim’s algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Remark. All three algorithms produce an MST.
Kruskal’s Algorithm

Find the minimum spanning tree using Kruskal’s algorithm:
Start with T = φ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

Reverse-Delete Algorithm

Find the minimum spanning tree using the Reverse-Delete algorithm:
Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prim’s Algorithm

Find the minimum spanning tree using Prim’s algorithm:
Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.

c is in the MST
f is not in the MST

Cycles and Cuts

Cycle. Set of edges the form a-b, b-c, c-d, ..., y-z, z-a.

Cutset. A cut is a subset of nodes S. The corresponding cutset D is the subset of edges with exactly one endpoint in S.

Cycle-Cut Intersection

Claim. A cycle and a cutset intersect in an even number of edges.

Pf. (by picture)
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST $T^*$ contains $e$.

Pf. (exchange argument)
- Suppose $e$ does not belong to $T^*$, and let's see what happens.
- Adding $e$ to $T^*$ creates a cycle $C$ in $T^*$.
- Edge $e$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S$.
- $T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost($T^*) < cost(T^*)$.
- This is a contradiction.

Cycle property. Let $C$ be any cycle in $G$, and let $f$ be the max cost edge belonging to $C$. Then the MST $T^*$ does not contain $f$.

Pf. (exchange argument)
- Suppose $f$ belongs to $T^*$, and let's see what happens.
- Deleting $f$ from $T^*$ creates a cut $S$ in $T^*$.
- Edge $f$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S$.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost($T^*) < cost(T^*)$.
- This is a contradiction.

Implementation: Prim's Algorithm

Implementation. Use a priority queue ala Dijkstra.
- Maintain set of explored nodes $S$.
- For each unexplored node $v$, maintain attachment cost $a[v] = \text{cost of cheapest edge } v \text{ to a node in } S$.
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

Prim($G, c$) {
  foreach ($v \in V$) $a[v] \leftarrow \infty$
  Initialize set of explored nodes $S \leftarrow \emptyset$
  while $(Q$ is not empty) {
    $u \leftarrow \text{delete min element from } Q$
    $S \leftarrow S \cup \{u\}$
    Initialize set of explored nodes $S \leftarrow \emptyset$
    while $(Q$ is not empty) {
      $u \leftarrow \text{delete min element from } Q$
      $S \leftarrow S \cup \{u\}$
      $\text{if } (v \notin S \text{ and } (c_e < a[v]))$
      $a[v] \leftarrow c_{e}$
      $\text{decrease priority } a[v] \text{ to } c_{e}$
    }
  }
  return $T$
}

Prim's Algorithm: Proof of Correctness (Sketch)

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]
- Only edges belonging to the minimum spanning tree are added (by application of the cut property).
- A spanning tree is created since
  - No cycles exist since each added edge must have exactly one endpoint in $S$.
  - All nodes are added, since this is the criterion that the algorithm stops.

Kruskal's Algorithm

Kruskal's algorithm. [Kruskal, 1956]
- Consider edges in ascending order of weight.
- Case 1: If adding $e$ to $T$ creates a cycle, discard $e$ according to cycle property.
- Case 2: Otherwise, insert $e = (u,v)$ into $T$ according to cut property where $S = \text{set of nodes in } u \text{'s connected component}$.

Kruskal($G, c$) {
  Sort edge weights so that $c_1 \leq c_2 \leq \ldots \leq c_m$.
  $T \leftarrow \emptyset$
  foreach ($u \in V$) {
    $\text{make a set containing singleton } u$
    for $i = 1$ to $m$
      are $u$ and $v$ in different connected components?
        $(u,v) = e_i$
      if $(u$ and $v$ are in different sets)$ {
        $T \leftarrow T \cup \{e_i\}$
        merge the sets containing $u$ and $v$
      }
    return $T$
}
Knapsack's Algorithm: Proof of Correctness

Proof:
- Only edges in the MST are added since it must span the cut since it doesn't create a cycle, and it is the cheapest (Cut property)
- A spanning tree is created
- It contains no cycles by design of the algorithm
- It is connected since otherwise there would be some edge that could be added without creating a cycle

Implementation: Kruskal's Algorithm

```javascript
Kruskal(G, c) {
    sort edge weights so that c_1 ≤ c_2 ≤ ... ≤ c_m
    T ← Ø
    foreach (u ∈ V) make a set containing singleton u
    for i = 1 to m
        (u,v) = e_i
        if (u,v are in different sets) {
            T ← T ∪ {e_i}
            merge the sets containing u and v
        }
    return T
}
```

**Union-Find Data Structure**

- `MakeUnionFind(S={a,b})` Create singleton trees for all items in the set
- `Union(A={a},B={b})` Merge two connected components by creating a pointer from the root of the smaller tree to the root of the larger tree. Store the size of its tree with each root.
- `Find(a)` Traverse up the tree until finding the root. The name of the root is the name of the tree.

**Time analysis**

- `MakeUnionFind(S={a,b})` Create singleton trees for all items in the set
  - `O(n)`
- `Union(A={a},B={b})` Merge two connected components by creating a pointer from the root of the smaller tree to the root of the larger tree. Store the size of its tree with each root.
  - `O(1)`
- `Find(a)` Traverse up the tree until finding the root. The name of the root is the name of the tree.
  - `O(log n)` Takes time on the order of the number of times the name changed. Since the larger set keeps its name, a name change implies that the set of at least doubled. It can be of size at most n.