Greedy Algorithms: Minimum Spanning Trees

Reading:
CLRS Chapter 23
Appendix B.4, B.5
Section 16.2

Greedy algorithm —
- Overall problem solved in series of steps.
- Choice made at each step "looks best at the moment" without explicit reference to overall problem. ("locally optimal")

Example:
We need to make 99¢ in change with minimum number of coins. We automatically do this with a greedy algorithm.

\[
99\text{¢} = \underbrace{\text{25¢}}_7 \times 3 + \underbrace{24\text{¢}}_3 + \underbrace{\text{10¢}}_2 + \underbrace{\text{4¢}}_4 + \underbrace{\text{4¢}}_2 + \underbrace{\text{4¢}}_2 + \underbrace{\text{0¢}}_0
\]

\[
3\text{ quarters} + 2\text{ dimes} + 4\text{ pennies} \Rightarrow 9\text{ coins}
\]

This greedy algorithm gives correct solution. Starting with largest coin, take as many as possible without going over.
But...

1. If we started with smallest coin (pennies):
\[ 99¢ = (1¢) \times 99 \rightarrow 99 \text{ coins} \]

2. If dime replaced by 11¢ piece, need to make 15¢
   - Greedy: \[ 15¢ = (11¢) \times 1 + (5¢) \times 0 + (1¢) \times 4 \rightarrow 5 \text{ coins} \]
   - Optimal: \[ 15¢ = (11¢) \times 0 + (5¢) \times 3 + (1¢) \times 0 \rightarrow 3 \text{ coins} \]

So greedy algorithms
- sometimes give correct solution (globally optimal)
- sometimes do not
  - change is made "correctly"
  - number of coins not optimal

Often greedy algorithms, even when not correct, are useful because provide "good solutions" efficiently.

Depends on
- structure of algorithm (forward/reverse)
- structure of problem (coin values)
Graphs (CLRS Appendix B.4, B.5)

Undirected graph: \( G = (V, E) \)
- vertices
- edges (unordered pairs of vertices)

Weighted undirected graph: \( G = (V, E) \)
- with weight function \( w: E \to \mathbb{R} \)

Tree: Graph that is acyclic and connected

Spanning Tree: (of graph \( G = (V, E) \))
- a subset \( T \subseteq E \) that forms a tree
- and spans the graph (touched all vertices)

Observation: Spanning tree has \( |V| - 1 \) edges

\[ G = (V, E) \]
- 8 edges
- 5 vertices

\( T \rightarrow \)
- One particular spanning tree

\( \rightarrow \) Adding any additional edge to \( T \)
- produces a cycle, which would be connected
- and spanning but no longer a tree

\( \rightarrow \) Other spanning trees exist here \( \square \)
Minimum Spanning Tree = MST

So let's put weights on the edges and think about finding minimum spanning trees.

**Input:** Connected, undirected graph \( G = (V,E) \) with weight function \( w : E \rightarrow \mathbb{R} \)

**Output:** A spanning tree \( T \) of minimum weight

\[
W(T) = \sum_{(u,v) \in T} w(u,v)
\]

sum over edges of MST

Problem is useful for connecting cities with minimum-length fiber-optic networks, pipelines, or other infrastructure.

- Other applications
  - Clustering through cutting largest edges in MST
  - Finding high-order correlational relationships in large datasets through mutual information graph and finding MST

- not minimal
- minimal
- minimal

- clustering
Heuristics

- Avoid large weights: 14, 15
- Include small weights: 3, 5
- Some edges are "inevitable": 9, 15 (only route to rest of graph)
- Avoid cycles, because then not a tree

Thoughts

- Would a greedy algorithm be likely to work here? Is there something about the structure of the problem where local choices that are best can be informed enough about the global solution?
- Should I start with all the edges present and remove them, or should I start with no edges present and add them?
- Is the MST unique? Or are there multiple equivalents?
- If close a cycle, can get back to MST by removing "heaviest" edge
Theorem: Let $G = (V,E)$ be connected graph with cost function defined on edges. Let $U$ be proper subset of $V$. If $(u,v)$ is an edge of lowest cost ("light edge") with $u \in U$ and $v \in V-U$, then there is an MST containing $(u,v)$.

Proof: (By cut-and-paste) Assume theorem false:
There is no MST that includes $(u,v)$. Let $T$ be an MST.

1. Adding $(u,v)$ introduces a cycle, because $T$ is an MST and already has a path connecting $U \cup V$.
2. There must be at least one other edge connecting $U \cap V$ in the cycle, WLOG call this $(u', v')$.
3. Deleting edge $(u', v')$ breaks cycle, giving tree $T'$.
4. $w(T') \leq w(T)$ because $(u,v)$ was light edge, so $w(u,v) \leq w(u', v')$.
5. Thus, our assumption was wrong and theorem is true:
$(u,v)$ is in an MST.
Now do you imagine greedy algorithm might work?

We can use local information to include and exclude edges from growing MSTs.

**Kruskal's Algorithm**

- Initially $T = (V, \emptyset)$ → vertices but no edges
- Examine edges of $E$ in increasing weight order (arbitrary order for ties)
  - If edge connects two unconnected components then add edge to $T$
  - Else discard edge and continue (forms cycle)
- Can terminate when all edges in simple connected tree, or can continue through all edges

Example graph with vertices and edges labeled.
Correctness of Algorithm: loop invariant

Prior to each iteration, T is a subset of an MST

Initialization: T has no edges, so trivially satisfied

Maintenance: Edges that are accepted by the
loop are light edges crossing a partition ("cut")
of V (that has U" being one of the unconnected
components). By MST Property, edges are in MST.

Termination: All edges are examined and added to T
if in MST, so T must be MST.

---

Note: For distinct weights, MST is unique

Implementation and Run-time analysis
require thinking through the data
structures that will be used and
the run time of operations on them.

→ What if we want a maximum, rather than
  a minimum?