Efficient Algorithms and Problem Complexity

– Greedy Algorithms –

Frank Drewes
Department of Computing Science
Umeå University
Today’s Menu

1. The Greedy Approach
2. Recalling Minimal Spanning Trees
3. Example: Kruskal’s Minimal Spanning Tree Algorithm
4. Example: Prim’s Minimal Spanning Tree Algorithm
The greedy approach

When does the greedy approach apply?

- Greedy algorithms usually seek to solve optimization problems.
- An optimization problem asks for a solution that satisfies some minimality or maximality requirement.
- Often, the desired result is a “subset” of the input (e.g. a subgraph).

The greedy pattern...

- Start with an “empty” solution.
- Extend the current solution stepwise by picking the best of the remaining items.
- Stop when nothing can be added anymore.
The greedy approach

How do we know that it works?
- In general, we don’t!
- We are not automatically guaranteed to find the optimum.
- Some greedy algorithms are merely good heuristics.
- Greedy algorithms are usually very efficient (about $n$ iterations, no backtracking) and easy to implement.

What is the “best item” that should be picked?
- The choice of the “best item” determines the algorithm.
- We talk about a greedy rule when referring to this choice.
- It must
  - make progress towards a final solution,
  - preserve the property of being a partial solution, and
  - be efficiently computable.
Minimal Spanning Trees

A spanning tree of a (connected undirected) weighted graph \( G = (V, E, w) \) is a subgraph \( T = (V', E') \) of \( G \) such that

- \( V' = V \) and
- \( T \) is a tree.

A minimal spanning tree (MST) of \( G \) is a spanning tree of \( G \) whose weight \( w(T) = \sum_{e \in E'} w(e) \) is minimal.

The MST problem

- Input: A connected weighted graph \( G \).
- Output: An MST of \( G \).
MST Construction Theorem

Consider

- a subgraph $G' = (V, E')$ of an MST of $G = (V, E, w)$,
- a connected component $C$ of $G'$, and
- the set $S \subseteq E$ of edges connecting nodes in $C$ with nodes outside $C$.

If $e$ is an edge of minimal weight in $S$, then $(V, E' \cup \{e\})$ is also a subgraph of an MST of $G$. 

Frank Drewes (Umeå University)
Proof of the MST Construction Theorem

Let \( e = (u, v) \), where \( u \) belongs to \( C \), and let \( T = (V, E'') \) be an MST of \( G \) that contains \( G' \).

Case 1: \( e \in T \)

\[ \Rightarrow (V, E' \cup \{e\}) \] is contained in \( T \) as well. – Done.

Case 2: \( e \notin T \)

\[ \Rightarrow (V, E'' \cup \{e\}) \] contains a cycle \( u_1 u_2 \cdots u_n \) with \( (u_1, u_2) = e \).

Let \( e' = (u', v') \) to be the first edge on this cycle such that \( v' \in C \).

\[ \Rightarrow (V, E'' \setminus \{e'\} \cup \{e\}) \] is an MST containing \( (V, E' \cup \{e\}) \).  [WHY?]
Kruskal’s algorithm, rough description

The input is given as a list of edges (and associated weights).

1. Start by defining $E' = \emptyset$.

2. **Greedy rule:** While $|E'| < |V| - 1$, choose $e = (u, v) \in E$ such that
   - $u, v$ do not belong to the same connected component of $(V, E')$ and
   - $w(u, v)$ is minimal among all edges with this property.

3. Add $e$ to $E'$ and repeat.

4. Return $(V, E')$. 
Correctness of Kruskal’s algorithm

Why does it work?

1. Initially $G' = (V, E')$ is a subgraph of an MST of $G$.
2. By the MST Construction Theorem, this property is preserved
   $\Rightarrow$ by induction, $G' = (V, E')$ is always contained in an MST of $G$.
3. Finally $|E'| = n - 1$, meaning $G'$ is equal to the MST it is contained in
   $\Rightarrow G'$ itself is the MST.
Kruskal’s algorithm, filling in some details

\[
\text{Kruskal}(E, n) \\
\text{sort}(E') \quad // \text{sort according to edge weights} \\
\text{for } v = 1 \text{ to } n \text{ do} \\
\quad \text{makeset}(v) \quad // \text{node } v \text{ is a singleton component} \\
E' = \emptyset \\
i = 1 \\
\text{while } |E'| < n - 1 \text{ do} \\
\quad (u, v) \leftarrow E[i++] \\
\quad \text{if } \text{findset}(u) \neq \text{findset}(v) \text{ then} \\
\quad \quad E' \leftarrow E' \cup \{(u, v)\} \\
\quad \quad \text{union}(u, v) \quad // \text{join the components} \\
\text{return } E' 
\]
Running time of Kruskal’s algorithm

1. At most \( \max(n, m) = O(m) \) calls of \text{makeset}, \text{findset}, \) and \text{union}. 
2. Each takes \( O(\log n) \) steps \( \Rightarrow O(m \log n) \subseteq O(m \log m) \) in total.
3. Even \( \text{sort}(E) \) runs in time \( O(m \log m) \).

**Running Time of Kruskal’s Algorithm**

Kruskal’s Algorithm runs in time \( O(m \log m) \), where \( m \) is the number of edges in the input graph.
Prim’s algorithm, rough description

For Prim’s algorithm, \( G \) is represented by adjacency lists.

1. Select any starting node \( v \), and put \( G' = (V', E') = (\{v\}, \emptyset) \).

2. **Greedy rule:** While \( |V'| < |V| \), choose \( e = (u, v) \in E \) such that
   1. \( u \in V' \) and \( v \in V \setminus V' \) and
   2. \( w(u, v) \) is minimal among all edges with this property.

3. Add \( v \) to \( V' \) and \( e \) to \( E' \) and repeat.

4. Return \((V', E')\).

**Correctness** is similar to the case of Kruskal’s algorithm.
Implementing Prim’s algorithm efficiently

Efficient implementation of Prim’s algorithm is slightly more complex.

- Represent the output by an array $parent[1, \ldots, n]$.
- Maintain a minheap $h$ to organize the nodes in $V \setminus V'$.
- The key of each node $v$ is $\min\{w(u, v) \mid v \in V'\}$.
- Iteration:
  1. Retrieve and remove the node $v$ with the minimal key from $h$.
  2. Iterate over $adj[v]$ to update the keys of all adjacent $v' \notin V'$ such that $w(v, v') < key(v')$.
  3. For each such $v'$, set $parent[v'] = v$.

$\Rightarrow$ each entry in the adjacency lists is considered at most once, and the time spent at it is $O(\log n)$
$\Rightarrow$ the total running time is $O(m \log n)$. 

Frank Drewes (Umeå University) Efficient Algorithms and Problem Complexity Lecture 4 13 / 16
A slightly more formal description

**Prim(adj)**

initialize minheap $h$ with $\text{key}(1) = 0$, $\text{key}(u) = \infty$ for $u > 1$

$\text{parent}[1] \leftarrow \bot$

**while** $h$ not empty **do**

$u \leftarrow h.\text{del}()$  // retrieve and delete node $u$ with smallest key

**for** $(v, \text{weight})$ in $\text{adj}[u]$ **do**

**if** $v$ is in $h$ && $\text{weight} < h.\text{keyval}(v)$ **then**

$h.\text{decrease}(v, \text{weight})$

$\text{parent}(v) \leftarrow u$

**return** $\text{parent}$
Running time of Prim’s algorithm

Running Time of Prim’s Algorithm

Prim’s Algorithm runs in time $O(m \log n)$.

(Again, $m$ is the number of edges in the input graph and $n$ is the number of nodes.)

Compare with Kruskal’s algorithm – is there a difference?
Example: Prim’s Minimal Spanning Tree Algorithm

Reading . . .

Please read the sections on minimal spanning trees in the textbook.