csce350 — Data Structures and Algorithms
Fall 2019 — Lecture Notes: Greedy Algorithms

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with occasional supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

Greedy algorithms solve problems by repeatedly making the most promising local choice.

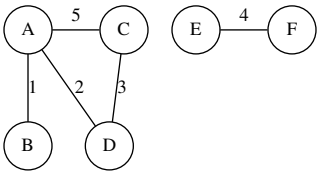
Build a solution step-by step. Each step must be:

- **Feasible** — Satisfy the constraints of the problem.
- **Locally optimal** — Best choice currently available.
- **Irrevocable** — Never 'backtrack' to try something else.

Don't forget: Greedy algorithms are usually not correct.

2 Graphs

A **weighted graph** is a collection of **vertices** and **edges**. Each edge has a number called its **weight**.

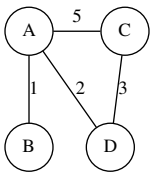


We can represent a graph using an **adjacency matrix** or **adjacency lists**.

3 Problem: Minimum spanning tree

Input: A connected weighted graph with V vertices and E edges.

Output: A minimum-weight list of $V - 1$ of the edges that keep the graph connected.



4 Prim's algorithm

Idea:

- Start with a single node.
- Connect the closest node into the tree.
- Repeat $V - 1$ times.

For each node, keep track of:

- Status: Tree, Fringe, Unseen
- Parent node.
  - Tree nodes: Which other tree node did we connect to?
  - Fringe nodes: Which neighbor tree node is closest?
  - Unseen nodes: nil
- Distance to parent node.
  - Tree nodes: Weight of edge to parent node.
  - Fringe nodes: Weight of edge to parent node.
  - Unseen nodes: ∞

5 Prim's algorithm: Finding the closest node

The main step of Prim's algorithm is to find the node that can be added with a single edge of lowest weight.

To implement this, there are a couple of choices:

- Check every node. (Leads to $\Theta(V^2)$ time.)
- Priority queue of nodes, sorted on distance. (Leads to $\Theta(E \log V)$ time.)

6 Kruskal's algorithm

Idea: Always add the lightest edge that does not create a cycle.

```
KRUSKALMST(G)
  sort edges of G in increasing weight order
  c ← 0
  k ← 0
  T ← empty set of edges
  while c < V − 1 do
    if E_k does not create a cycle in T then
      add E_k to T
      c ← c + 1
    end if
    k ← k + 1
  end while
  return T
```

7 Detecting cycles: Disjoint set forest data structure

Goal: Keep track of a partition of a finite set into subsets.

Operations:

- FIND(a) — return a 'representative' of the set containing a.
- UNION(a, b) — combine the set containing a with the set containing b

Details: Keep track of a **parent** for each node.

```
FIND(a)
  r ← a
  while parent(r) ≠ r do
    r ← parent[r]
  end while
  return r
```

```
UNION(a, b)
  parent[FIND(a)] ← FIND(b)
```

8 Disjoint set forest in Kruskal's

We can use a disjoint set forest to determine whether a new edge would create a cycle.

FIND(a) $\neq$ FIND(b)
$\Downarrow$
a and b are in separate connected components
$\Downarrow$
a — b does not create a cycle

FIND(a) $=$ FIND(b)
$\Downarrow$
a and b are connected already
$\Downarrow$
a — b does create a cycle