ALGORITHM DESIGN GREEDY ALGORITHMS

by

David Toman

University of Waterloo

List of Slides

1
2 Greedy Approaches
3 Example: Making Change
4 Making Change (cont.)
5 Minimum Spanning Tree
6 Example
7 Approaches that Don’t Work
8 Prim’s Algorithm
9 Example
10 Example
11 ADTs needed for Prim’s Algorithm
12 Kruskal’s Algorithm
13 Example
14 Which one to use?
15 Finding Path(s) in a Graph
16 Dijkstra’s Algorithm
17 Example
18 Why does all of this Work?
19 Summary

Greedy Approaches

IDEA: iteratively make the best *local* choice

$\Rightarrow$ no "undoing" of decisions made

$\Rightarrow$ locality guarantees *reasonable performance*

$\Rightarrow$ in general may lead to non-optimal solutions

proofs of correctness

Examples:

- making change
- finding a *minimum spanning tree*

  $\Rightarrow$ Prim's and Kurskal's algorithms
- finding a shortest path

  $\Rightarrow$ Dijkstra's algorithm

Example: Making Change

Input: A natural number k.

Output: A minimum-sized collection of coins whose values sum to k.

The allowed coin values s are 1, 5, 10, 25, and 100.

Greedy solution:

while $k > 0$

output the maximum coin value v s. t. $v \leq k$

$k := k - v$

Making Change (cont.)

Example 1: Change for 48:

$$25, 10, 10, 1, 1, 1$$

Example 2: Change for 30
$\Rightarrow$ but assume that there are no "5-coins":

$$25, 1, 1, 1, 1, 1$$

$\Rightarrow$ NOT OPTIMAL ($10, 10, 10$ is optimal)

Questions:

1. what sets of coins guarantee optimal solution?
2. how hard is it really to make change optimally?

Minimum Spanning Tree

Def: A tree is a connected acyclic undirected graph.

Def: Let $G = (V, E)$ be a connected undirected graph. A spanning tree for G is a subset $E' \subseteq E$ such that (V, E') is a tree.

We are looking for an algorithm:

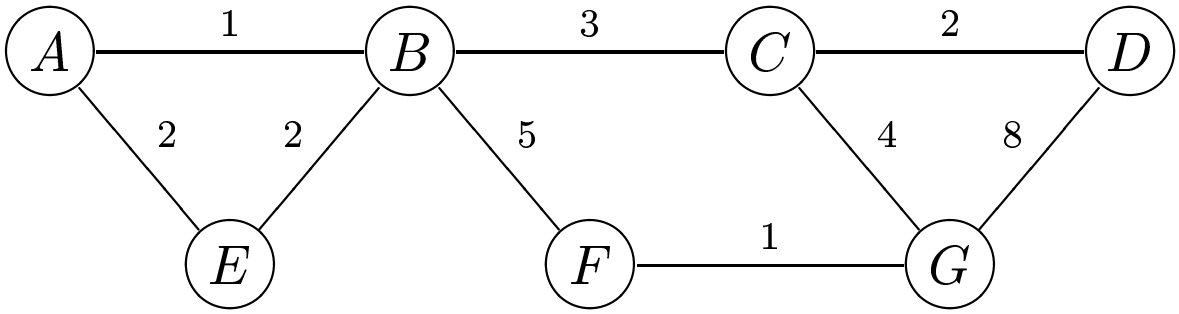
Input: A connected undirected graph G, each of whose edges has an associated positive integer weight.

Output: A spanning tree for G whose total weight is minimized.

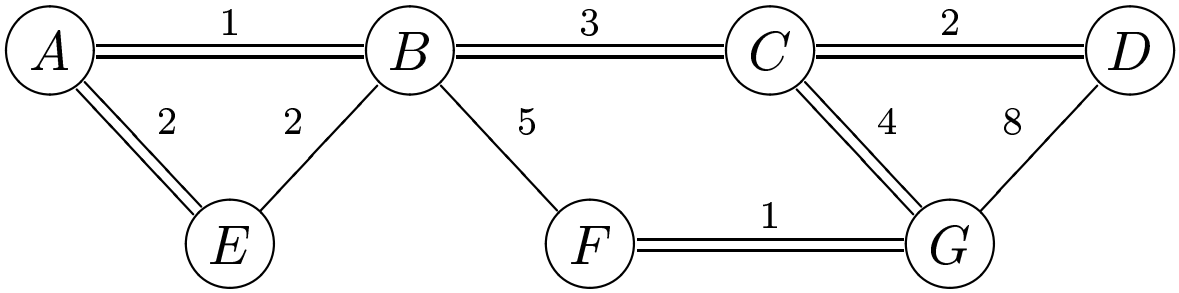
$\Rightarrow$ solution is not necessarily unique (example?)

Example

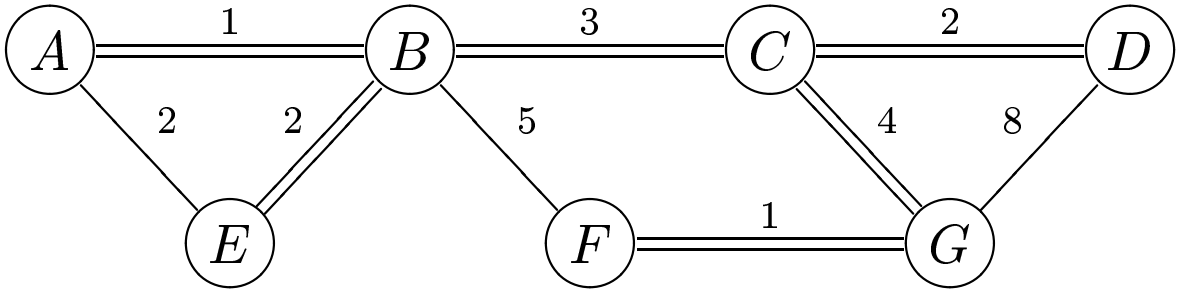
Given Graph:



Minimal Spanning Tree:



Another Minimal Spanning Tree:



Approaches that Don't Work

- pick a node, follow the "shortest" edge greedily

  $\Rightarrow$ similar to BFS/DFS
  $\Rightarrow$ guarantees to produce a minimal spanning tree

  $\Rightarrow$ NOT optimal (example?)

- do an exhaustive search

  $\Rightarrow$ modify DFS/BFS to generate all trees (how?)
  $\Rightarrow$ pick the one with minimal total weight
  $\Rightarrow$ guaranteed to find a *correct* solution

  $\Rightarrow$ too inefficient (exponential; why?)

Prim's Algorithm

IDEA: Grow the tree greedily in successive stages

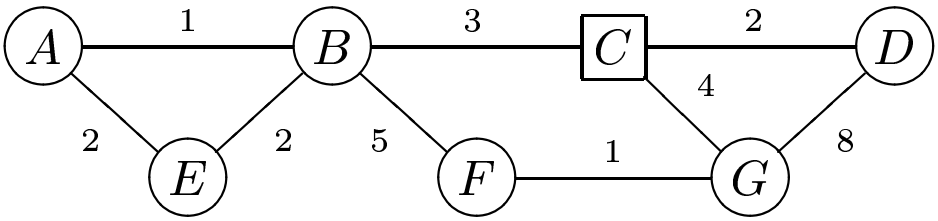
$\Rightarrow$ pick a node and mark it visited

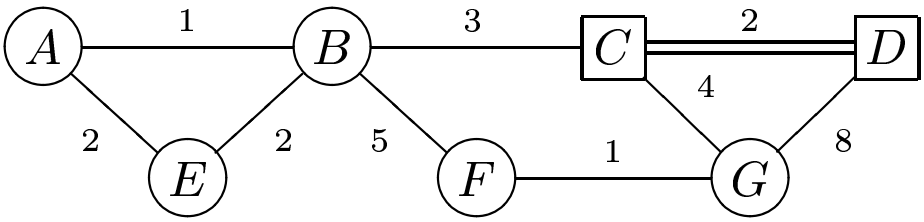
$\Rightarrow$ select a minimum-weight edge e from edges between a visited and an unvisited nodes and add it to E'

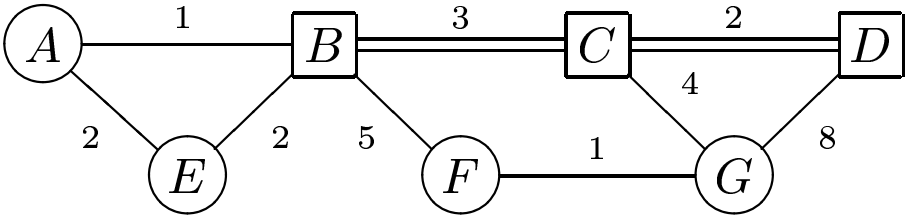
$\Rightarrow$ mark both endpoints of e visited and repeat.

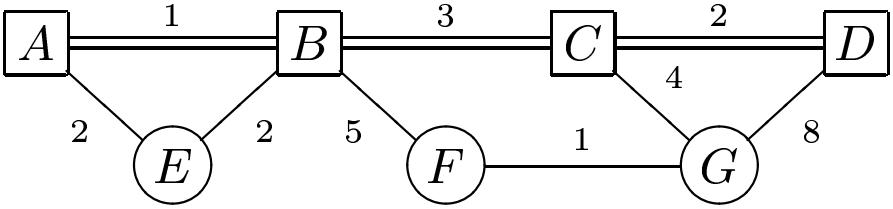
Example

Starting from C:

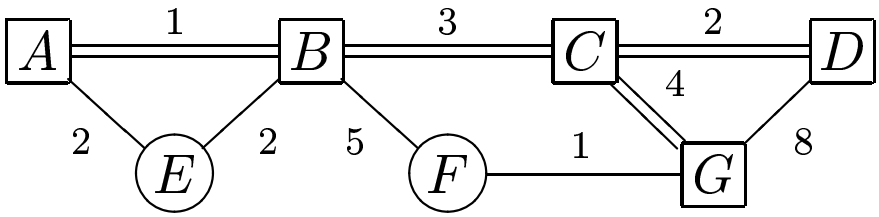


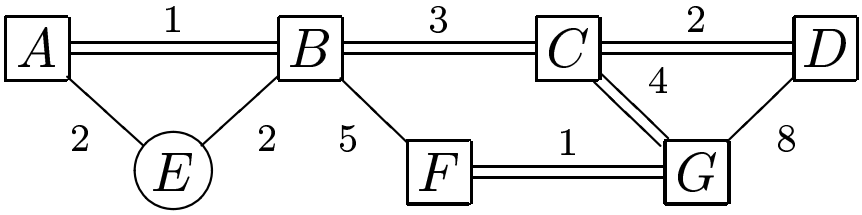


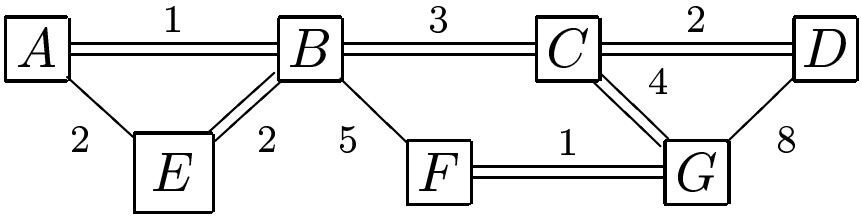




Example







ADTs needed for Prim's Algorithm

- how do we store visited nodes?

  $\Rightarrow$ insert, retrieve [dictionary?]

- how to store edges?

  $\Rightarrow$ get in increasing order of cost [ordered list?]

this gives us $O(|V|^2)$ algorithm

$\Rightarrow$ optimal for dense graphs ($|E| \sim |V|^2$)
$\Rightarrow$ for sparse graphs ($|E| << |V|^2$) we can use heaps

Kruskal's Algorithm

IDEA: Always select a minimum-weight edge from all of the edges in the graph that does not close a cycle.

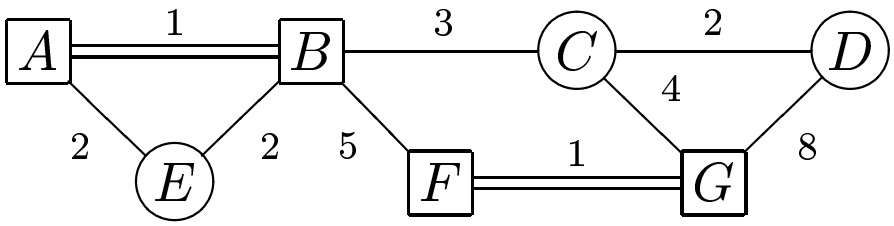
$\Rightarrow$ use a sorted list of edges

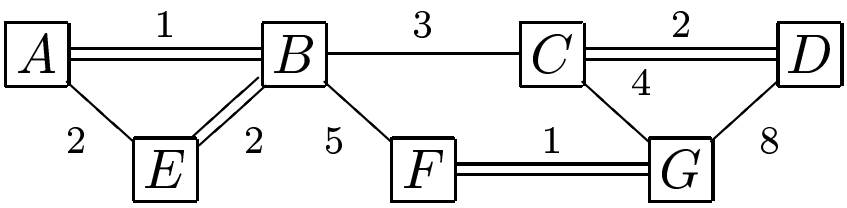
- needs to keep a **forest** of trees

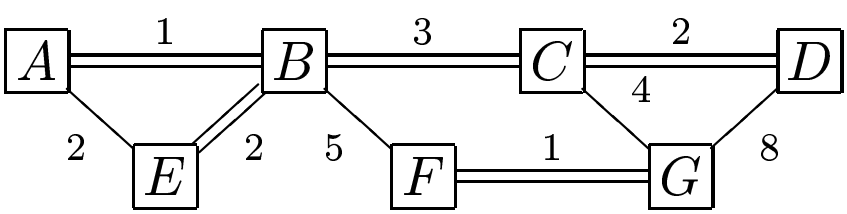
  $\Rightarrow$ initially all nodes are separate trees

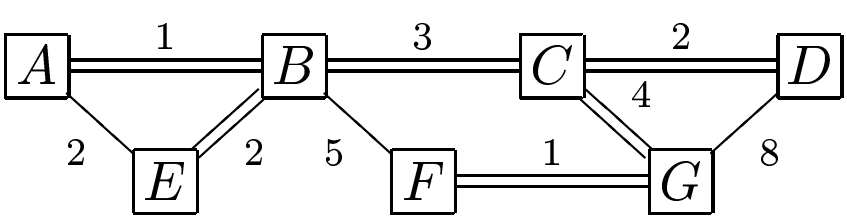
- use *union-find* data structure/algorithm (chapter 8)

Example


A
B
C
D
E
F
G
1
3
2
4
2
2
5
1
8


A
B
C
D
E
F
G
1
3
2
4
2
2
5
1
8


A
B
C
D
E
F
G
1
3
2
4
2
2
5
1
8


A
B
C
D
E
F
G
1
3
2
4
2
2
5
1
8

Which one to use?

- Kruskal's algorithm ($O(a \log n)$) is better if the graph is sparse.
- Prim's algorithm ($O(n^2)$) is better if the graph is dense.

Finding Path(s) in a Graph

Def: A directed graph is an ordered pair (V, E) where V is a finite set of nodes, and E is a set of ordered pairs of distinct nodes from V.

$\Rightarrow$ how to find a shortest path between nodes u and v?
$\Rightarrow$ is there a path at all?

Single-Source Shortest Paths:

Input: A directed graph with positive integer edge weights and a start node $u \in V$.

Output: For each $v \in V$, the length of a minimum-weight path from u to v.

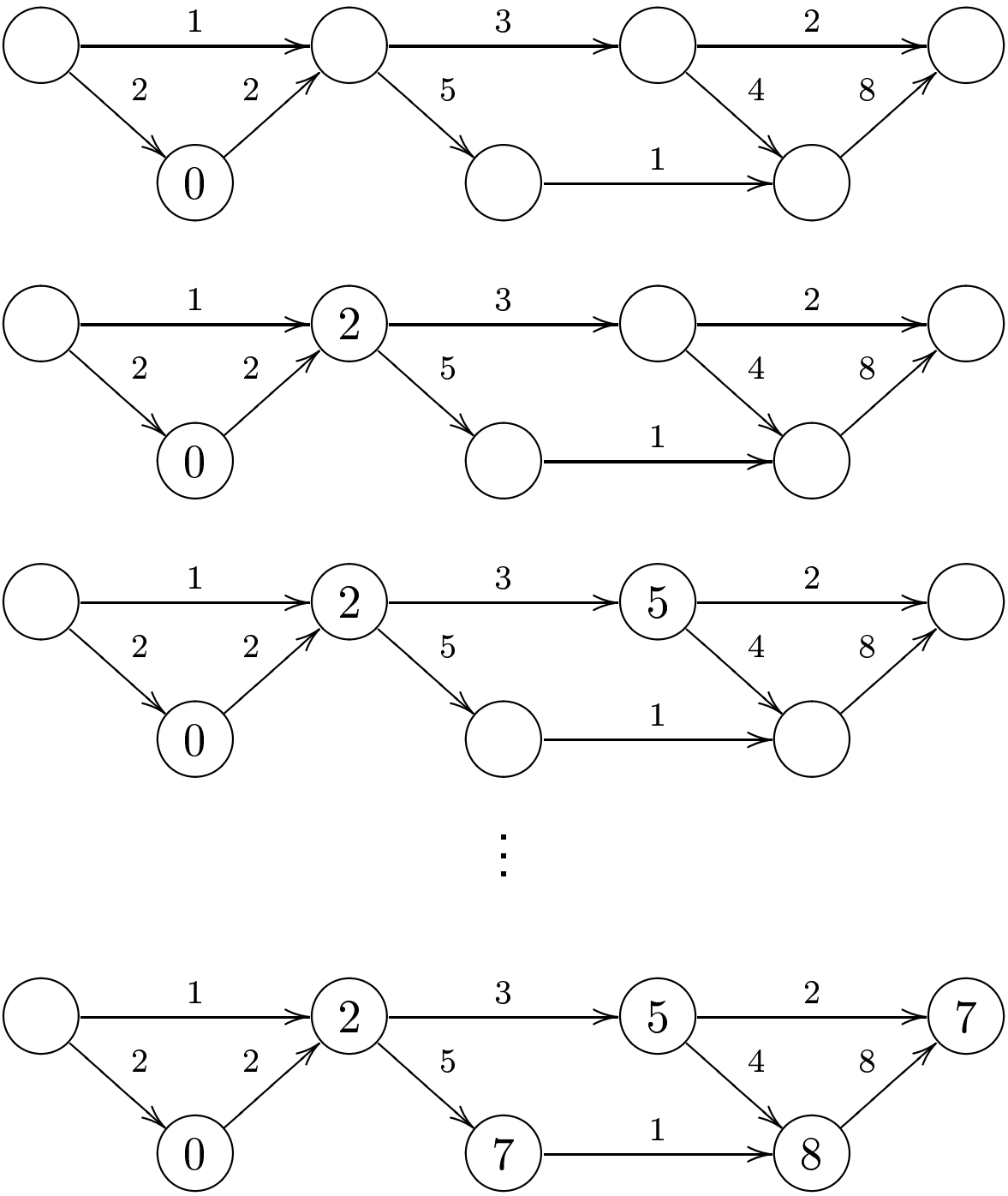
negative cycles?

Dijkstra's Algorithm

IDEA: compute the path length by adding the next closest vertex (to vertices seen so far) until all vertices are added.

$S := \{u\}$, $D(u) = 0$
while there is an edge from S to $V - S$
    let $v \in V - S$ be a node that
        minimizes $D(w) + d_{wv}$ for all $w \in S$
    $D(v) := \min\{D(w) + d_{wv} : w \in S\}$
    for all $w \in S$ do
        if $D(w) > D(v) + d_{vw}$ then
            $D(w) := D(v) + d_{vw}$
    $S := S \cup \{v\}$

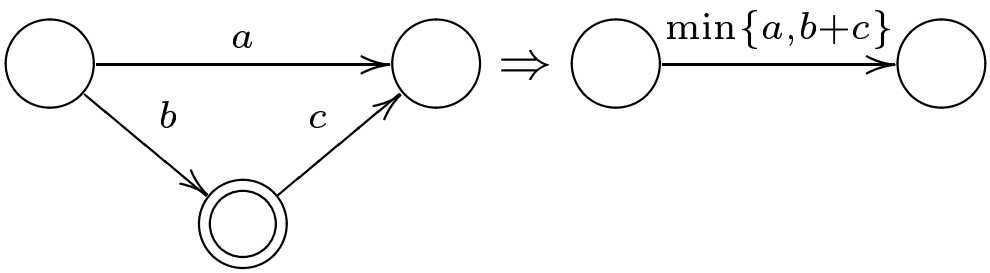
Example



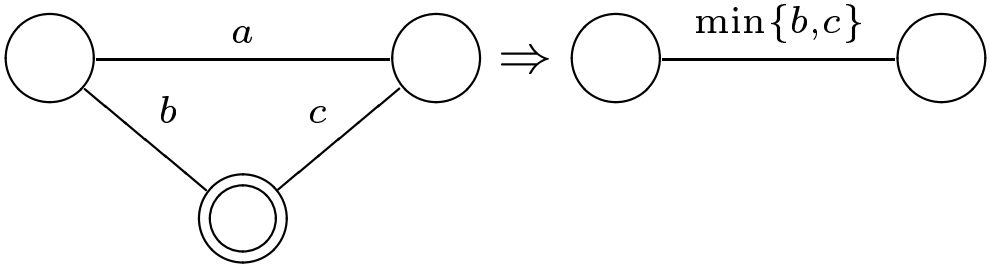
Why does all of this Work?

OBSERVATION:

- Dijkstra's Algorithm:



- Prim's Algorithm:



+ induction

Summary

- greedy approach gives *optimal* solutions

  $\Rightarrow$ but only if set up correctly

- other typical problems with greedy solutions:
  1. simple (non-preemptive) scheduling
  2. Huffman codes
  3. approximate bin packing

     $\Rightarrow$ not optimal! (e.g., at least extra $4/3$ bins)
  4. ...