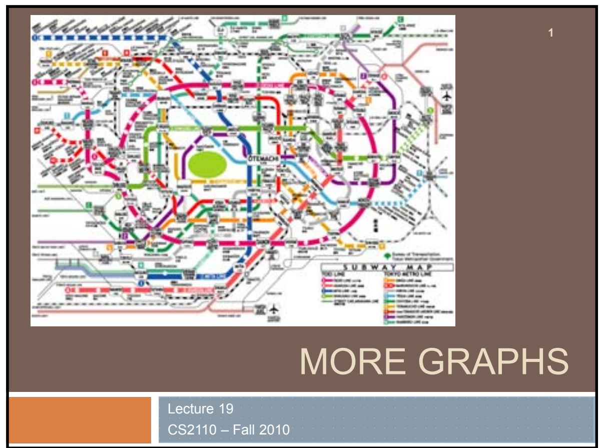
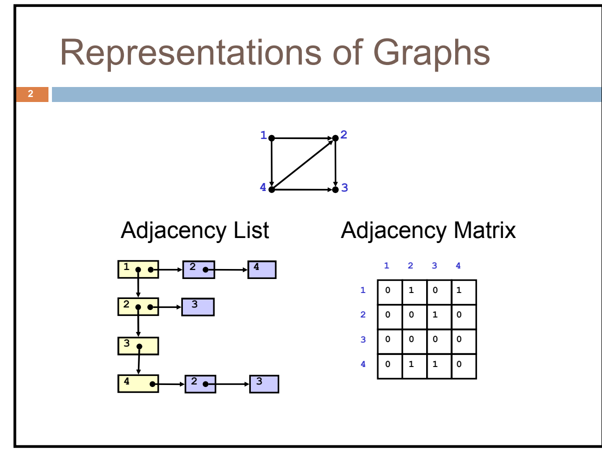




Adjacency Matrix or Adjacency List?

3

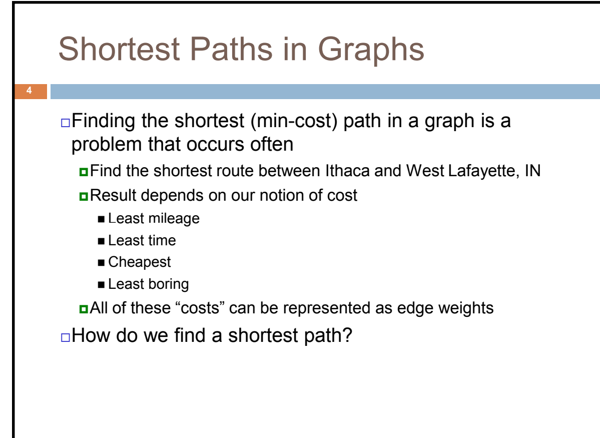
n = number of vertices

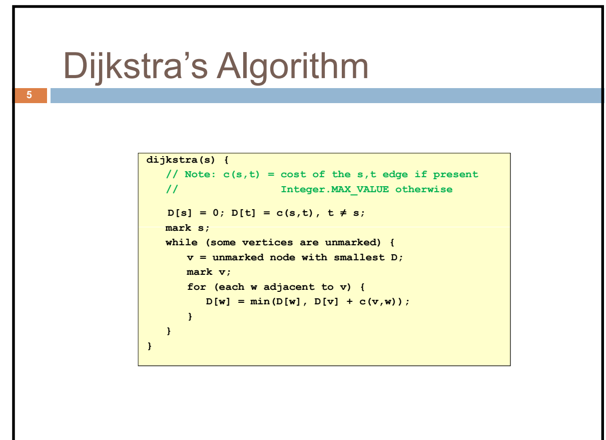
m = number of edges

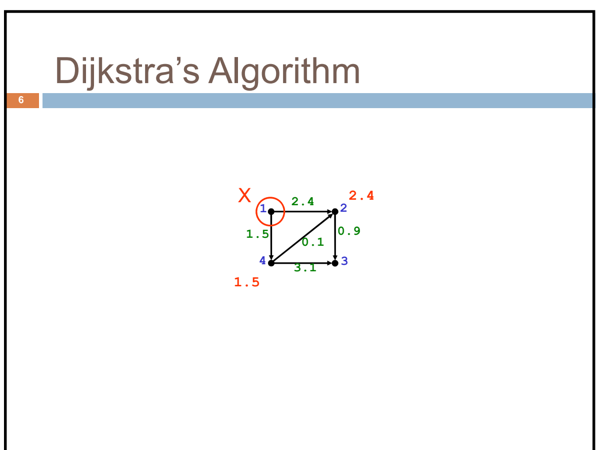
d(u) = outdegree of u

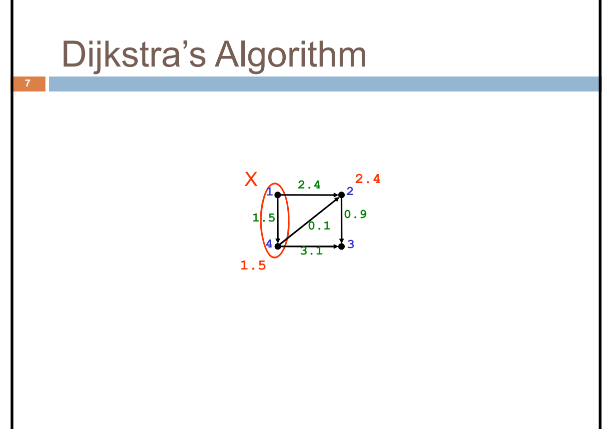
- Adjacency Matrix
  - Uses space $O(n^2)$
  - Can iterate over all edges in time $O(n^2)$
  - Can answer "Is there an edge from u to v?" in O(1) time
  - Better for dense graphs (lots of edges)

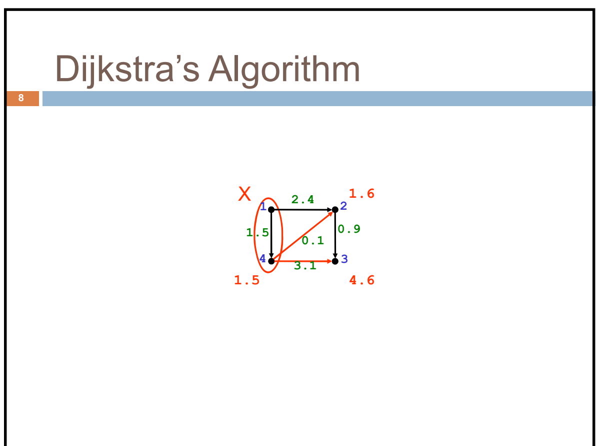
- Adjacency List
  - Uses space O(m+n)
  - Can iterate over all edges in time O(m+n)
  - Can answer "Is there an edge from u to v?" in O(d(u)) time
  - Better for sparse graphs (fewer edges)

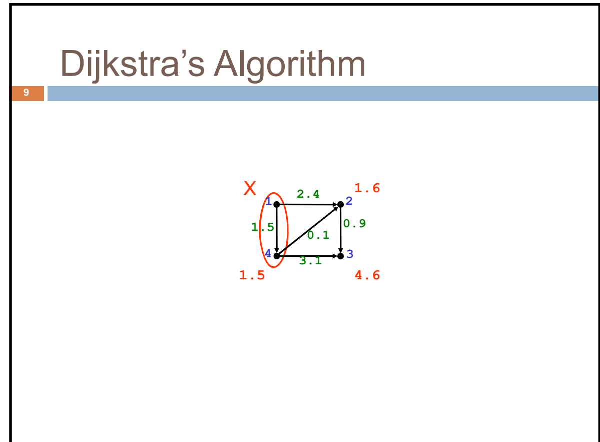


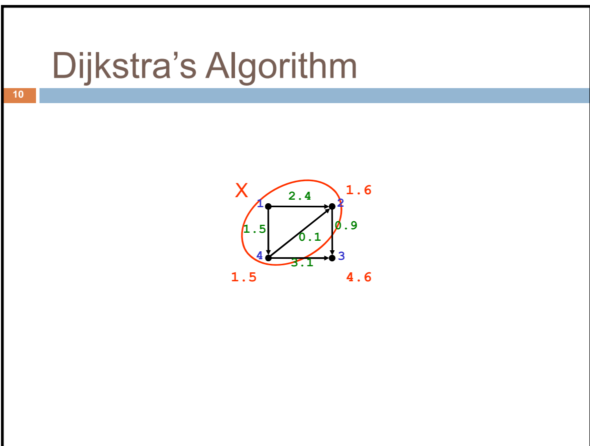


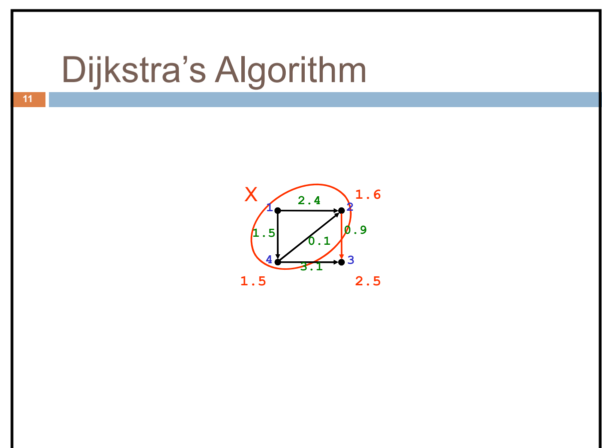


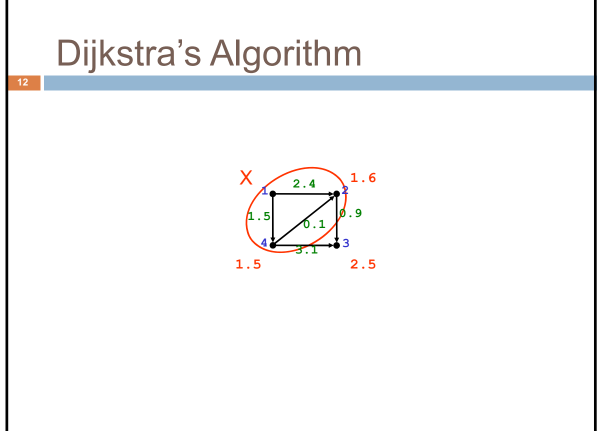

Dijkstra’s Algorithm
7
X
2.4
2.4
1
2
1.5
0.9
0.1
4
3.1
3
1.5


Dijkstra’s Algorithm
8
X
2.4
1.6
1
2
1.5
0.9
0.1
4
3.1
3
1.5
4.6


Dijkstra’s Algorithm
9
X
2.4
1.6
1
2
1.5
0.9
0.1
4
3.1
3
1.5
4.6

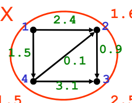

Dijkstra’s Algorithm
10
X
2.4
1.6
1
2
1.5
0.9
0.1
4
3.1
3
1.5
4.6


Dijkstra’s Algorithm
11
X
2.4
1.6
1
2
1.5
0.9
0.1
4
3.1
3
1.5
2.5


Dijkstra’s Algorithm
12
X
2.4
1.6
1
2
1.5
0.9
0.1
4
3.1
3
1.5
2.5

Dijkstra's Algorithm

13



Proof of Correctness

14

The following are invariants of the loop:

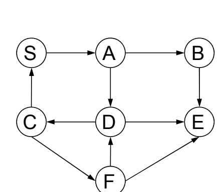
- X is the set of marked nodes
- For u ∈ X, D(u) = d(s,u)
- For u ∈ X and v ∉ X, d(s,u) ≤ d(s,v)
- For all u, D(u) is the length of the shortest path from s to u such that all nodes on the path (except possibly u) are in X

Implementation:

- Use a priority queue for the nodes not yet taken – priority is D(u)

Shortest Paths for Unweighted Graphs – A Special Case

15

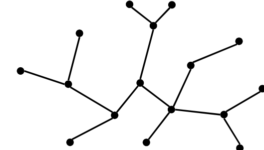


- Use breadth-first search
- Time is O(n + m) in adj list representation, O(n^2) in adj matrix representation

Undirected Trees

16

- An undirected graph is a *tree* if there is exactly one simple path between any pair of vertices

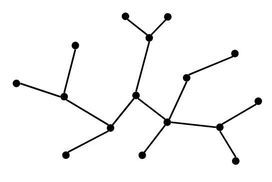


Facts About Trees

17

- |E| = |V| – 1
- connected
- no cycles

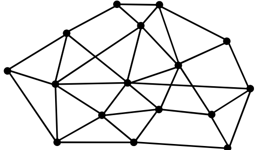
In fact, any two of these properties imply the third, and imply that the graph is a tree



Spanning Trees

18

A *spanning tree* of a connected undirected graph (V,E) is a subgraph (V,E') that is a tree

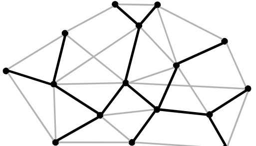


Spanning Trees

19

A *spanning tree* of a connected undirected graph (V,E) is a subgraph (V,E') that is a tree

- Same set of vertices V
- E' ⊆ E
- (V,E') is a tree

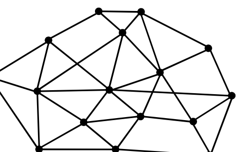


Finding a Spanning Tree

20

A subtractive method

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles

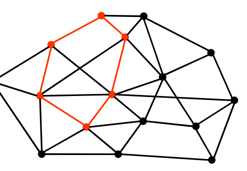


Finding a Spanning Tree

21

A subtractive method

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles

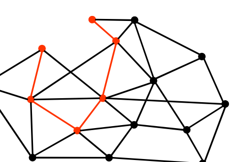


Finding a Spanning Tree

22

A subtractive method

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles

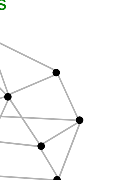


Finding a Spanning Tree

23

An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component

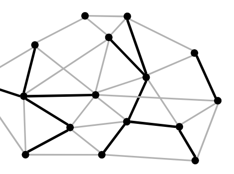


Finding a Spanning Tree

24

An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component



Finding a Spanning Tree

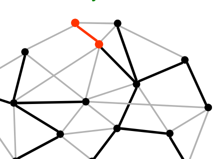
An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component

Finding a Spanning Tree

An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component



Finding a Spanning Tree

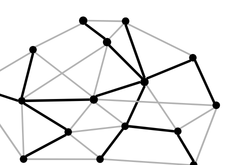
An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component

Finding a Spanning Tree

An additive method

- Start with no edges – there are no cycles
- If more than one connected component, insert an edge between them – still no cycles (why?)
- Repeat until only one component

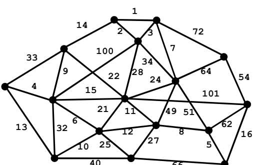


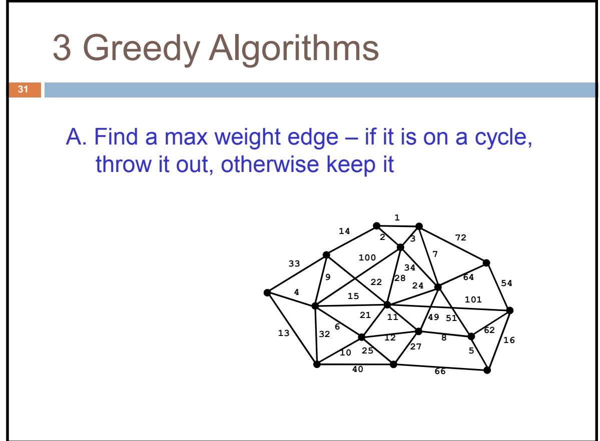
Minimum Spanning Trees

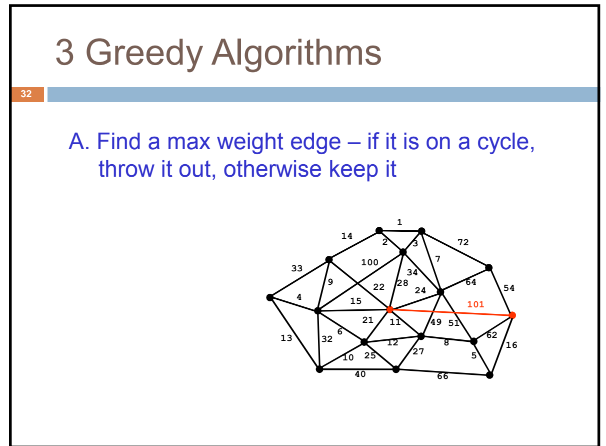
- Suppose edges are weighted, and we want a spanning tree of *minimum cost* (sum of edge weights)
- Some graphs have exactly one minimum spanning tree. Others have multiple trees with the same cost, any of which is a minimum spanning tree

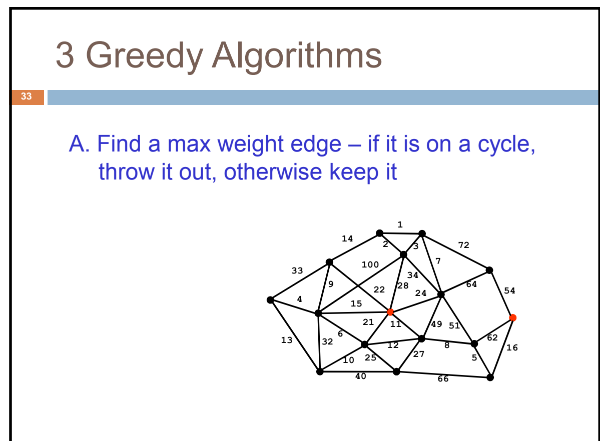
Minimum Spanning Trees

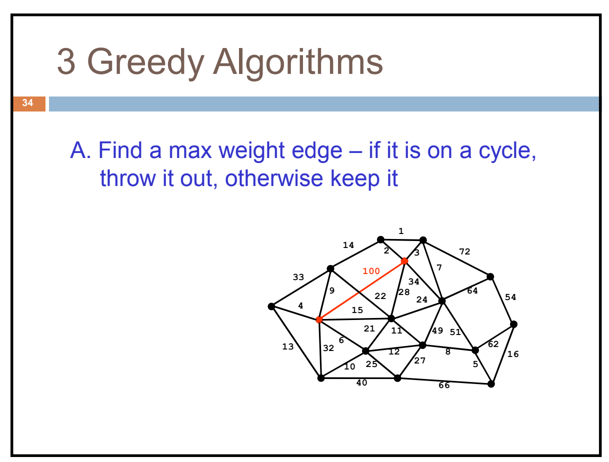
- Suppose edges are weighted, and we want a spanning tree of *minimum cost* (sum of edge weights)
- Useful in network routing & other applications
- For example, to stream a video

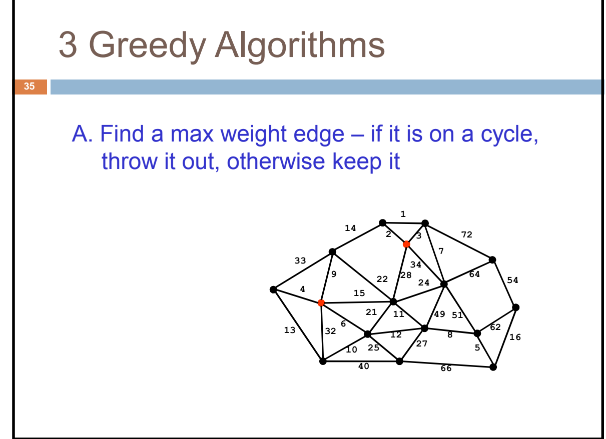


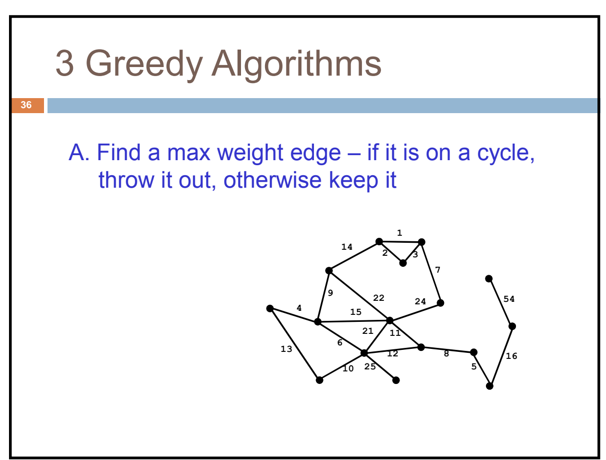

3 Greedy Algorithms
31
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it


3 Greedy Algorithms
32
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it


3 Greedy Algorithms
33
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it


3 Greedy Algorithms
34
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it

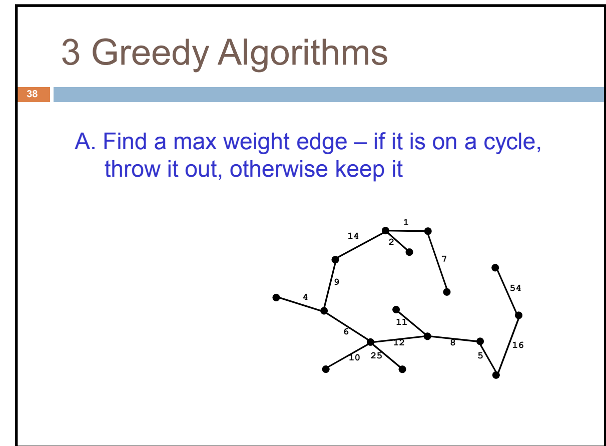

3 Greedy Algorithms
35
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it

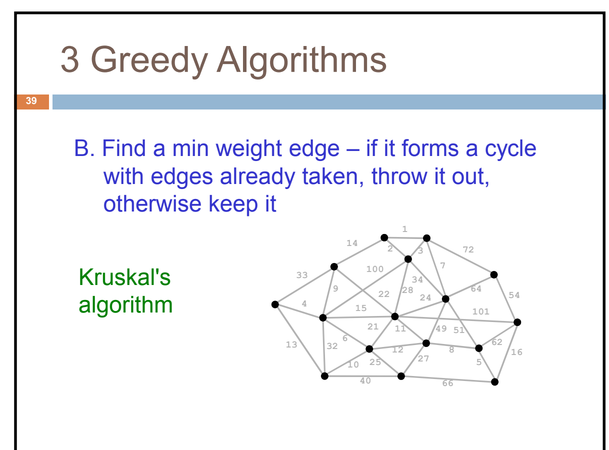

3 Greedy Algorithms
36
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it

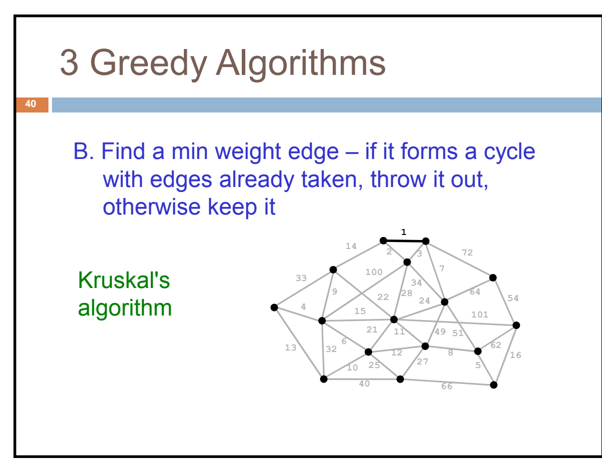
3 Greedy Algorithms

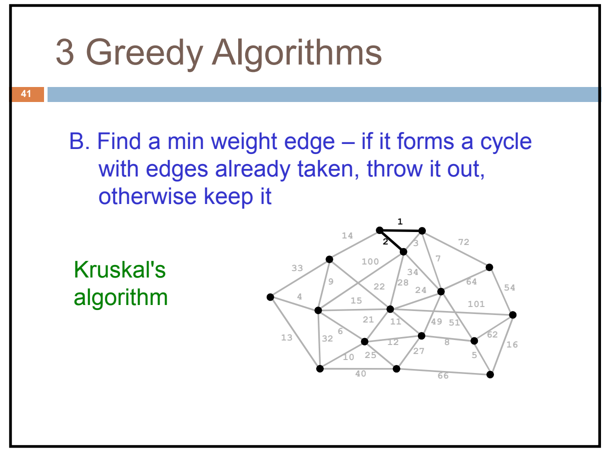
37

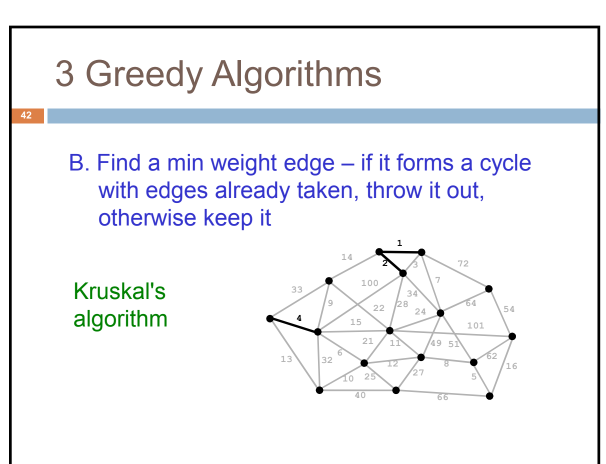
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it

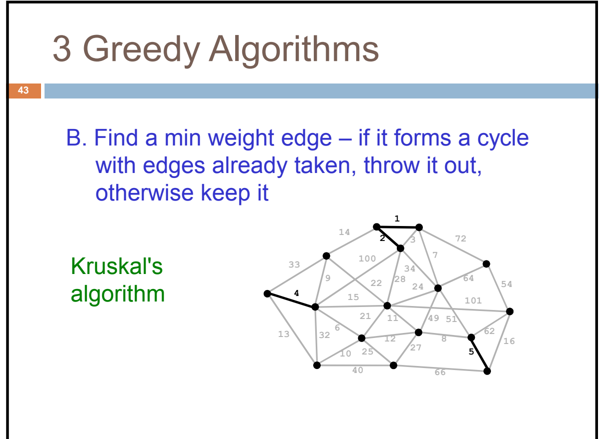


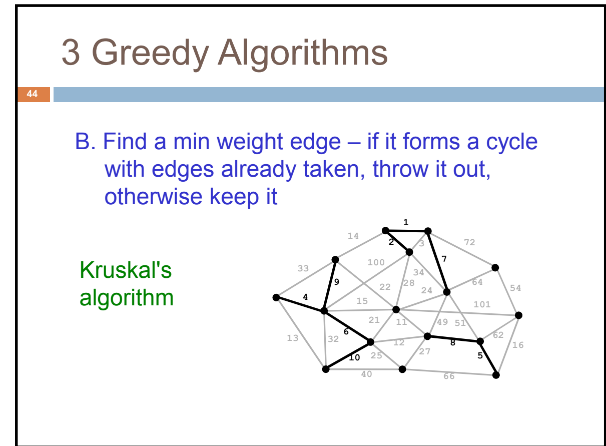


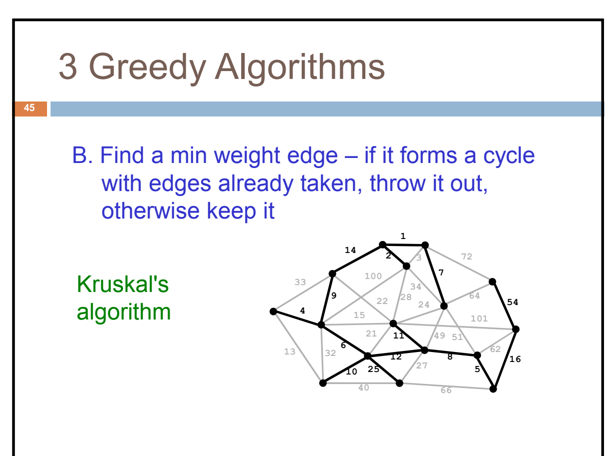










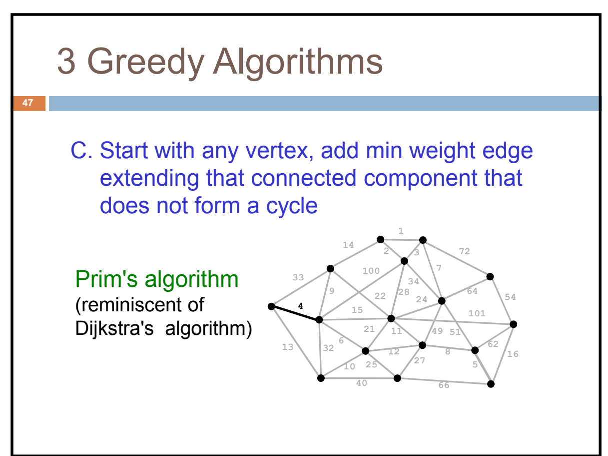


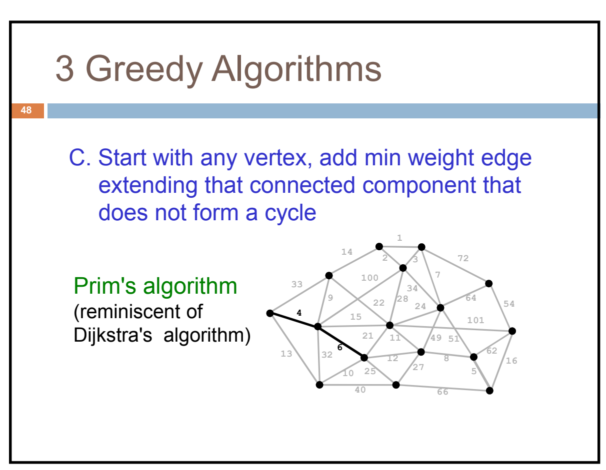
3 Greedy Algorithms

46

C. Start with any vertex, add min weight edge extending that connected component that does not form a cycle

Prim's algorithm
(reminiscent of Dijkstra's algorithm)



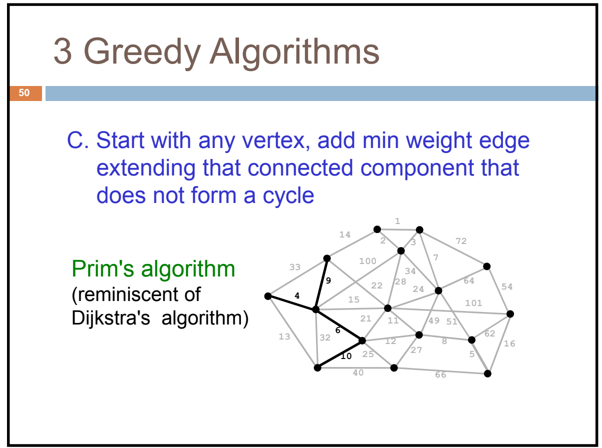


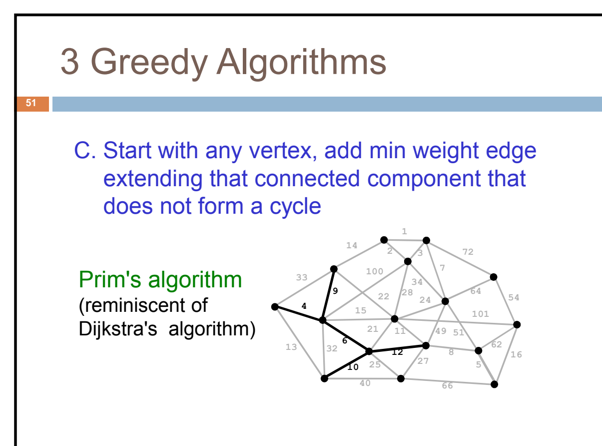
3 Greedy Algorithms

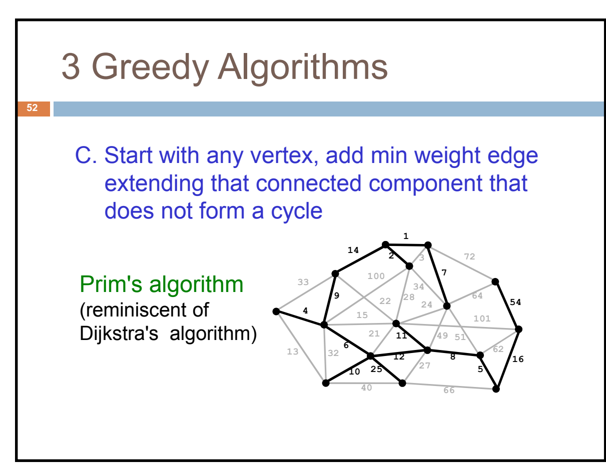
49

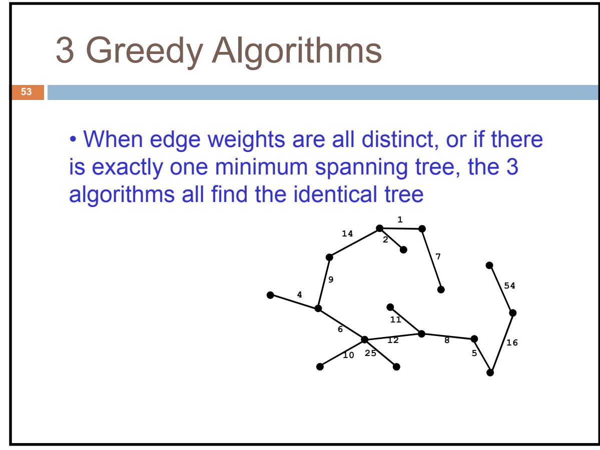
C. Start with any vertex, add min weight edge extending that connected component that does not form a cycle

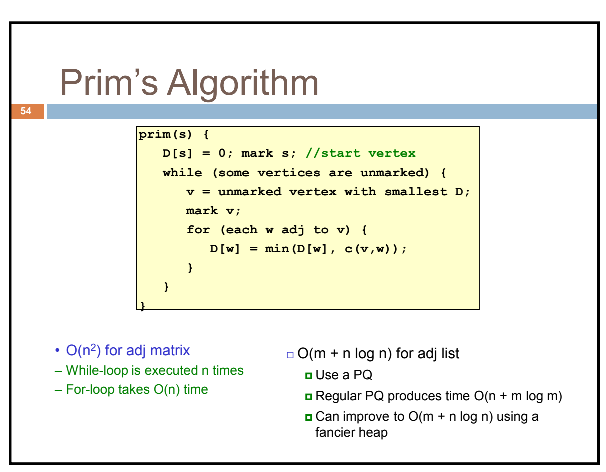
Prim's algorithm
(reminiscent of
Dijkstra's algorithm)











Greedy Algorithms

55

- These are examples of Greedy Algorithms
- The Greedy Strategy is an algorithm design technique
  - Like Divide & Conquer
- Greedy algorithms are used to solve optimization problems
  - The goal is to find the *best* solution
- Works when the problem has the greedy-choice property
  - A global optimum can be reached by making locally optimum choices

- Example: the Change Making Problem: Given an amount of money, find the smallest number of coins to make that amount
- Solution: Use a Greedy Algorithm
  - Give as many large coins as you can
- This greedy strategy produces the optimum number of coins for the US coin system
- Different money system ® greedy strategy may fail
  - Example: old UK system

Similar Code Structures

56

```
while (some vertices are
        unmarked) {
   v = best of unmarked
        vertices;
   mark v;
   for (each w adj to v)
        update w;
}
```

- Breadth-first-search (bfs)
  - best: next in queue
  - update: D[w] = D[v]+1
- Dijkstra's algorithm
  - best: next in PQ
  - update: D[w] = min(D[w], D[v]+c(v,w))
- Prim's algorithm
  - best: next in PQ
  - update: D[w] = min(D[w], c(v,w))