CSE 417: Algorithms and Computational Complexity

Greedy Graph Algorithms

Autumn 2002
Paul Beame

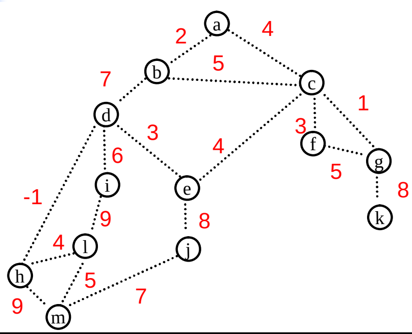
1

Minimum Spanning Trees (Forests)

- Given an undirected graph G=(V,E) with each edge e having a weight w(e)
- Find a subgraph T of G of minimum total weight s.t. every pair of vertices connected in G are also connected in T
  - if **G** is connected then **T** is a tree otherwise it is a forest

2

Weighted Undirected Graph



3

First Greedy Algorithm

- Prim's Algorithm:
  - start at a vertex v
  - add the cheapest edge adjacent to v
  - repeatedly add the cheapest edge that joins the vertices explored so far to the rest of the graph.

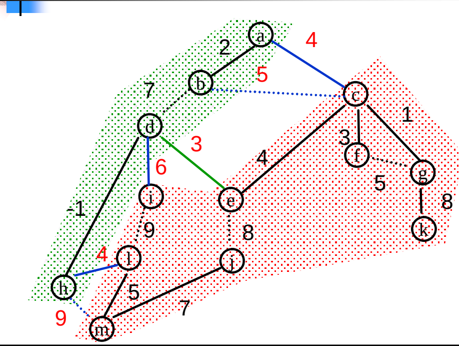
4

Why a greedy algorithm works here

- **Definition:** Given a graph **G=(V,E)**, a **cut** of **G** is a partition of **V** into two non-empty pieces, **S** and **V-S**
- **Lemma:** For every cut **(S,V-S)** of **G**, there is a minimum spanning tree (or forest) containing any **cheapest edge crossing the cut**, i.e. connecting some node in **S** with some node in **V-S**.
  - call such an edge **safe**

5

Cuts and Spanning Trees

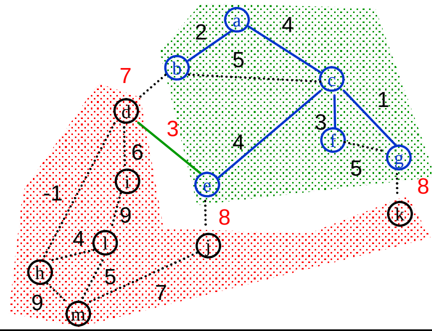


6

The greedy algorithm always chooses a safe edges

- Prim's Algorithm
  - Always chooses cheapest edge from current tree to rest of the graph
  - This is cheapest edge across a cut which has the vertices of that tree on one side.

Prim's Algorithm



Naive Prim's Algorithm Implementation & Analysis

- Computing the minimum weight edge at each stage.
  - **O(m)** per step (new vertex)
- n vertices in total
- O(nm) overall

Data Structure Review

- **Priority Queue:**
  - Elements each with an associated **key**
  - Operations
    - **Insert**
    - **Find-min**
      - Return the element with the smallest key
    - **Delete-min**
      - Return the element with the smallest key and delete it from the data structure
    - **Decrease-key**
      - Decrease the key value of some element
- Implementations
  - Arrays: **O(n)** time find/delete-min, **O(1)** time insert/decrease-key
  - Heaps: **O(**log **n)** time insert/find/delete-min, **O(1)** time decrease-key

Prim's Algorithm with Priority Queues

- For each vertex **u** not in tree maintain current cheapest edge from tree to **u**
  - Store **u** in priority queue with key = weight of this edge
- Operations:
  - n-1 insertions (each vertex added once)
  - n-1 delete-mins (each vertex deleted once)
    - pick the vertex of smallest key, remove it from the priority queue and add its edge to the graph
  - <m decrease-keys (each edge updates one vertex)

Prim's Algorithm with Priority Queues

- Priority queue implementations
  - Array
    - insert O(1), delete-min O(n), decrease-key O(1)
    - total $O(n+n^2+m)=O(n^2)$
  - Heap
    - insert, delete-min, decrease-key all O(log n)
    - total O(m log n)
  - d-Heap (d=m/n)
    - insert, delete-min, decrease-key all $O(\log_{m/n} n)$
    - total $O(m \log_{m/n} n)$

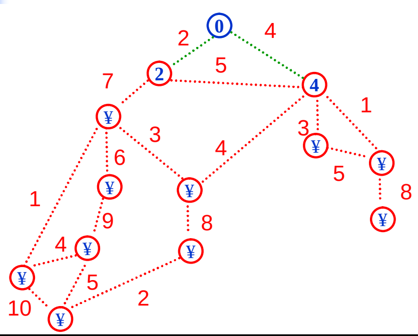
Single-source shortest paths

- Given an (un)directed graph G=(V,E) with each edge e having a non-negative weight w(e) and a vertex v
- Find length of shortest paths from v to each vertex in G

A greedy algorithm

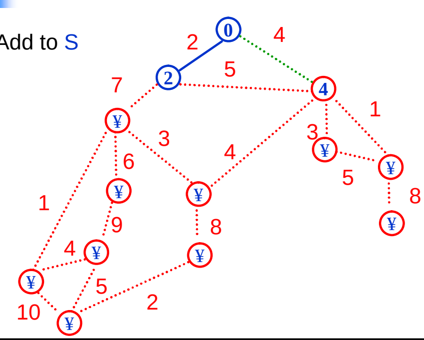
- Dijkstra's Algorithm:
  - Maintain a set **S** of vertices whose shortest paths are known
    - initially **S={v}**
  - Maintaining current best lengths of paths that only go through **S** to each of the vertices in **G**
    - path-lengths to elements of **S** will be right, to **V-S** they might not be right
  - Repeatedly add vertex **u** to **S** that has the shortest path-length of any vertex in **V-S**
    - update path lengths based on new paths through u

Dijkstra's Algorithm



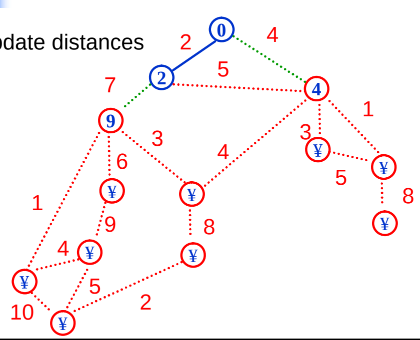
Dijkstra's Algorithm

Add to S



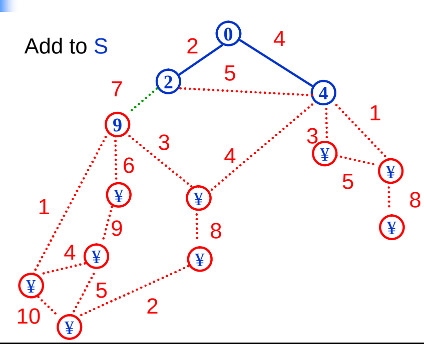
Dijkstra's Algorithm

Update distances



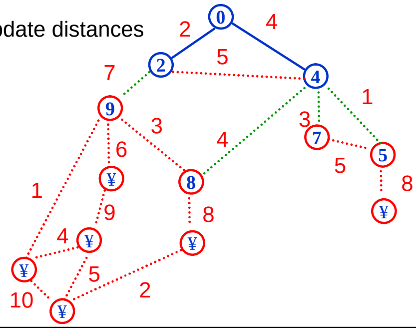
Dijkstra's Algorithm

Add to S



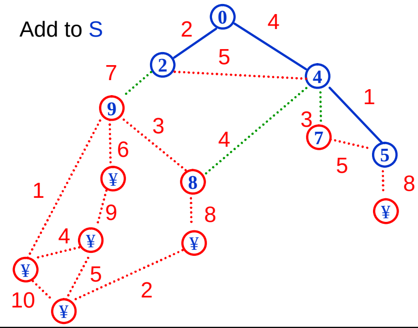
Dijkstra's Algorithm

Update distances



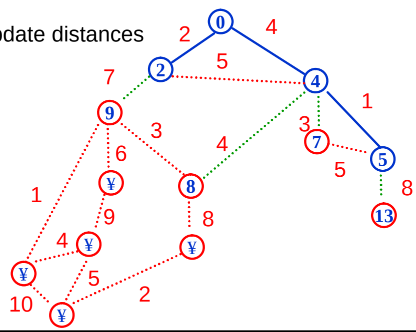
Dijkstra's Algorithm

Add to S



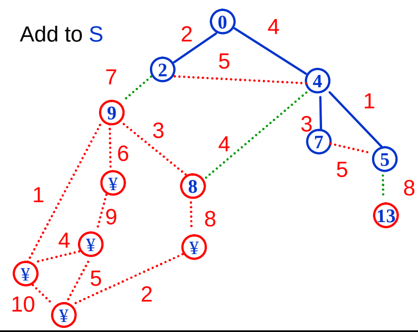
Dijkstra's Algorithm

Update distances



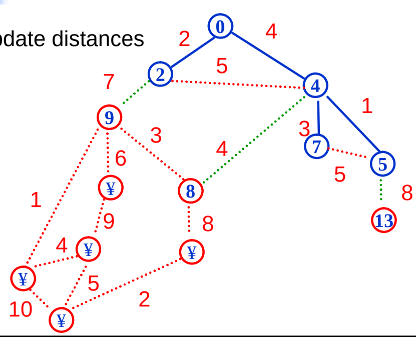
Dijkstra's Algorithm

Add to S



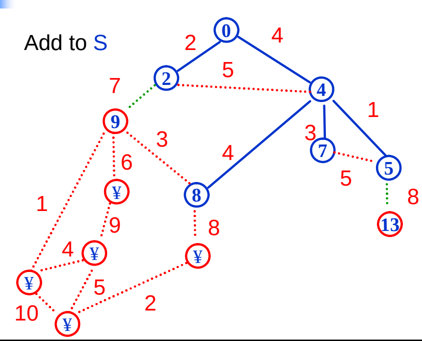
Dijkstra's Algorithm

Update distances



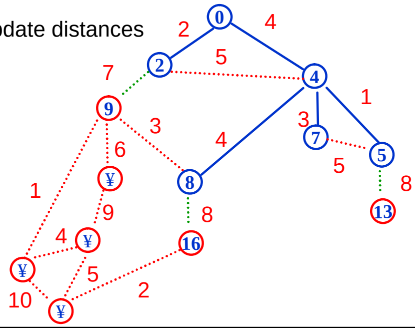
Dijkstra's Algorithm

Add to S



Dijkstra's Algorithm

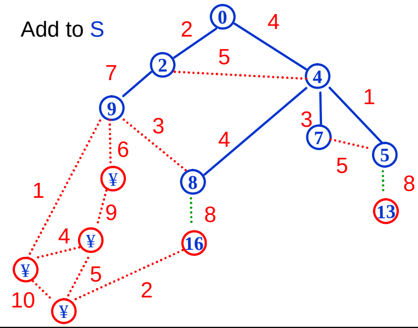
Update distances



25

Dijkstra's Algorithm

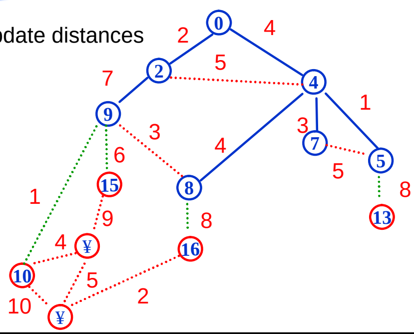
Add to S



26

Dijkstra's Algorithm

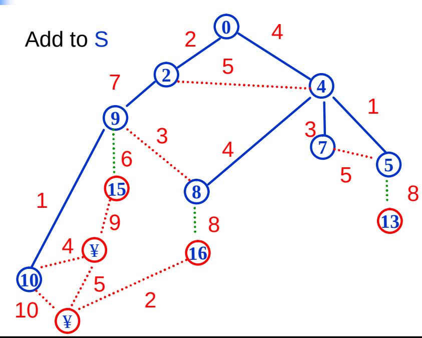
Update distances



27

Dijkstra's Algorithm

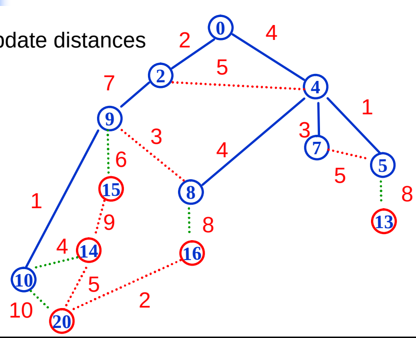
Add to S



28

Dijkstra's Algorithm

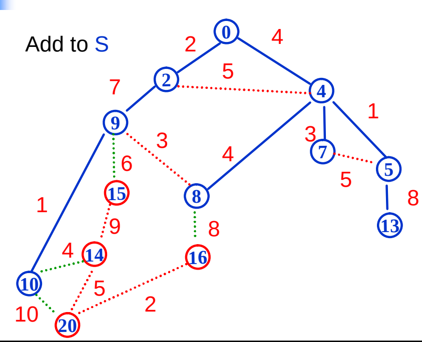
Update distances



29

Dijkstra's Algorithm

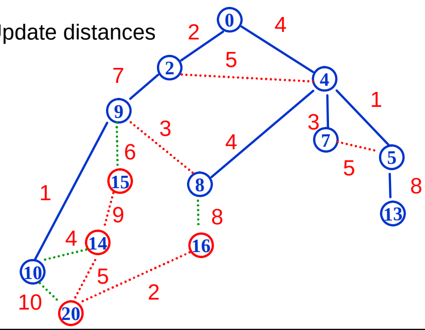
Add to S



30

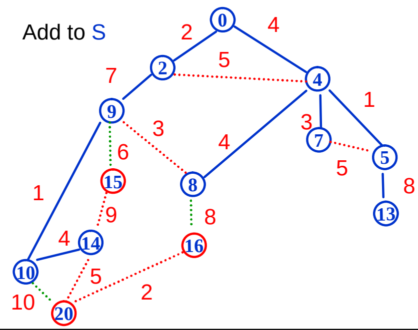
Dijkstra's Algorithm

Update distances



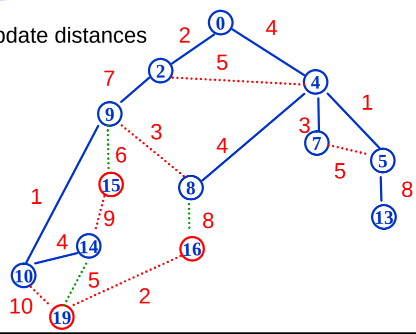
Dijkstra's Algorithm

Add to S



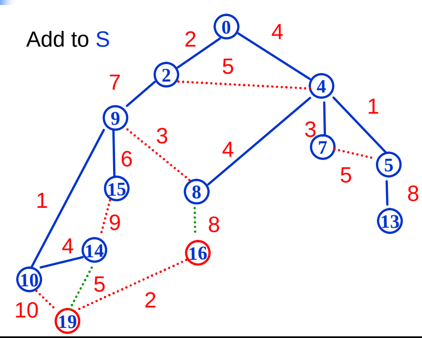
Dijkstra's Algorithm

Update distances



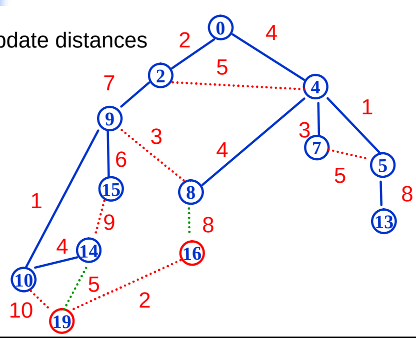
Dijkstra's Algorithm

Add to S



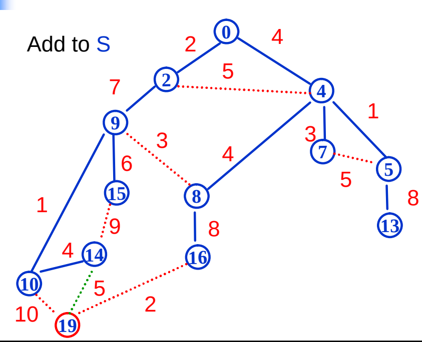
Dijkstra's Algorithm

Update distances



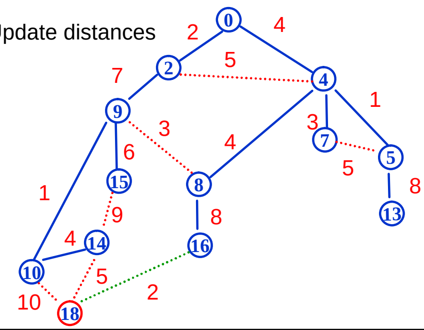
Dijkstra's Algorithm

Add to S



Dijkstra’s Algorithm

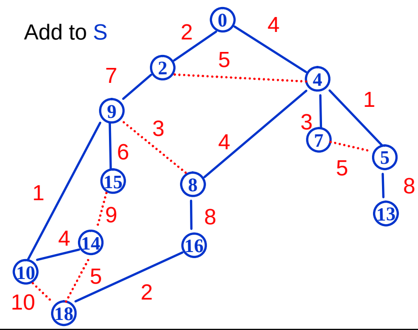
Update distances



37

Dijkstra’s Algorithm

Add to S

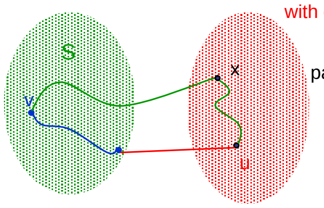


38

Dijkstra’s Algorithm Correctness

Suppose all distances to vertices in S are correct and u has smallest current value in V-S

∴distance value of vertex in V-S=length of shortest path from v with only last edge leaving S



Suppose some other path to u and x= first vertex on this path not in S

d(u)≤ d(x)

x-u path length ≥ 0

∴ other path is longer

Therefore adding u to S keeps correct distances

39

Dijkstra’s Algorithm

- Algorithm also produces a tree of shortest paths to v
  - From **w** follow its ancestors in the tree back to **v**
- If all you care about is the shortest path from v to w simply stop the algorithm when w is added to S

40

Implementing Dijkstra’s Algorithm

- Need to
  - keep current distance values for nodes in **V-S**
  - find minimum current distance value
  - reduce distances when vertex moved to **S**
- Same operations as priority queue version of Prim’s Algorithm
  - only difference is rule for updating values
    - node value + edge-weight vs edge-weight alone
  - same run-times as Prim’s Algorithm **O(m log n)**

41