CSE 417: Algorithms and Computational Complexity

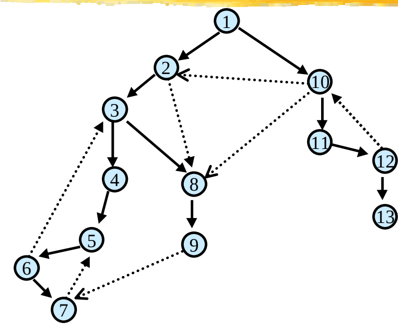
Winter 2001

Lecture 10

Instructor: Paul Beame

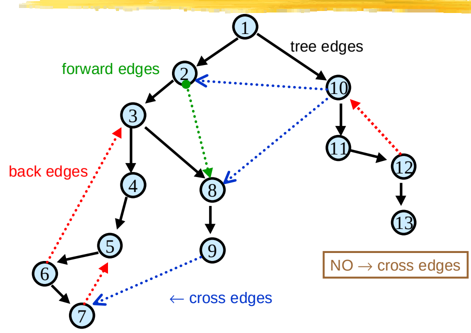
1

DFS(v) for a directed graph



2

DFS(v)



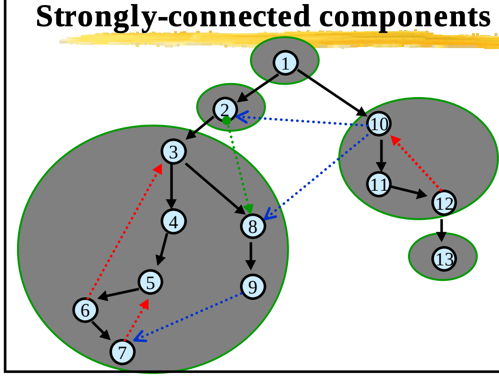
3

Strongly-connected components

- In directed graph if there is a path from a to b there might not be one from b to a
- a and b are **strongly connected** iff there is a path in both directions (i.e. a directed cycle containing both a and b
- Breaks graph into components

4

Strongly-connected components



5

Uses for SCC's

- Optimizing compilers need to find loops, which are SCC's in the program flow graph.
- If (u,v) means process u is waiting for process v, SCC's show deadlocks.

6

Directed Acyclic Graphs

- If we collapse each SCC to a single vertex we get a directed graph with no cycles
  - a **directed acyclic graph** or **DAG**
- Many problems on directed graphs can be solved as follows:
  - Compute SCC's and resulting DAG
  - Do one computation on each SCC
  - Do another computation on the overall DAG

7

Simple SCC Algorithm

- u,v in same SCC iff there are paths $u \to v$ & $v \to u$
- DFS from every u, v: $O(nm) = O(n^3)$

8

Better method

- Can compute all the SCC's while doing a single DFS! $O(n+m)$ time
- We won't do the full algorithm but will give some ideas

9

Definition

The **root** of an SCC is the first vertex in it visited by DFS.

Equivalently, the root is the vertex in the SCC with the smallest number in DFS ordering.

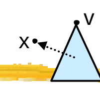
10

Subgoal

- All members of an SCC are descendants of its root.
- Can we identify some root?
- How about the root of the first SCC completely explored by DFS?
- Key idea: no exit from first SCC
  (first SCC is leftmost "leaf" in collapsed DAG)

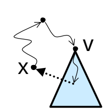
11

Definition



x is an *exit* from v (from v's subtree) if
- x is not a descendant of v, but
- x is the head of a (cross- or back-) edge from a descendant of v (including v itself)

- Any non-root vertex v has an exit

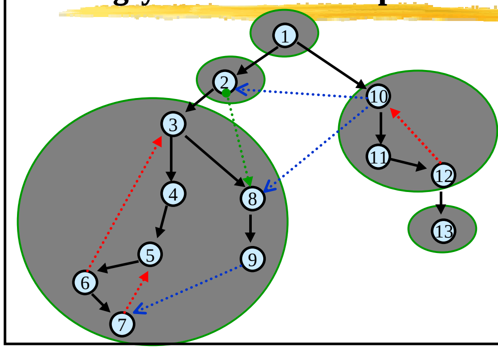


12

Finding SCC's

- A root nodes v sometimes have exits
  - only via a cross-edge to a node x that is not in a component with a root above v, e.g. vertex 10 in the example.

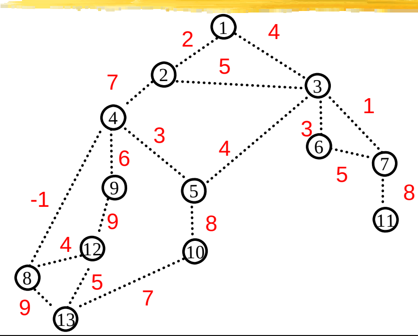
Strongly-connected components



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

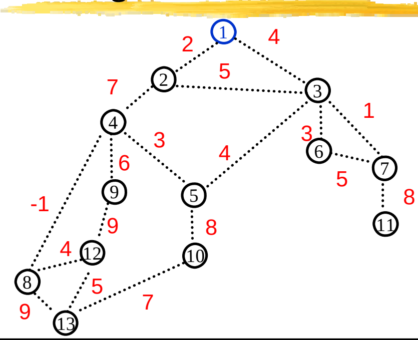
Weighted Undirected Graph



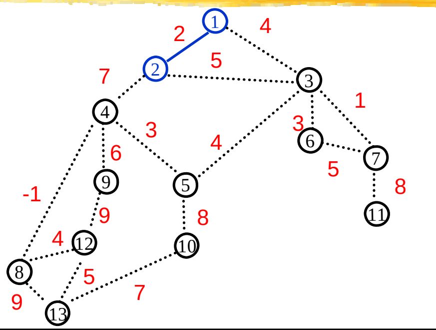
First Greedy Algorithm

- 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.
- We'll show it works later

Prim's Algorithm

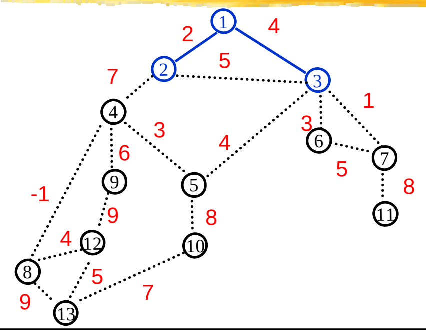


Prim’s Algorithm



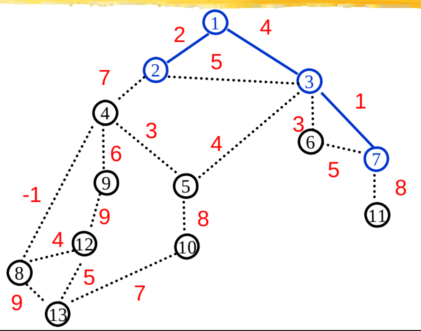
19

Prim’s Algorithm



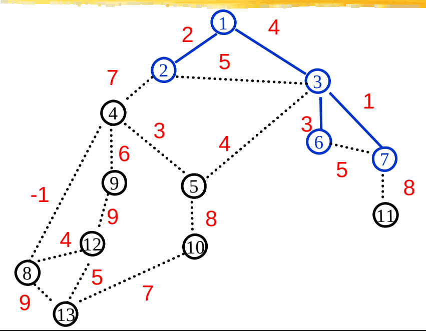
20

Prim’s Algorithm



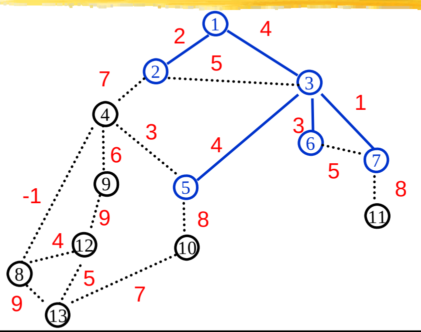
21

Prim’s Algorithm



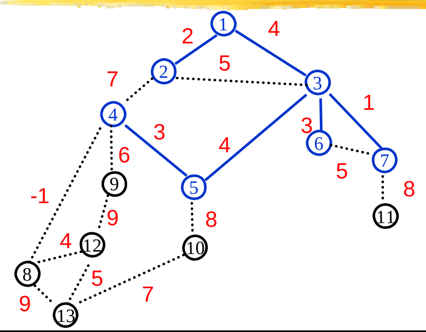
22

Prim’s Algorithm



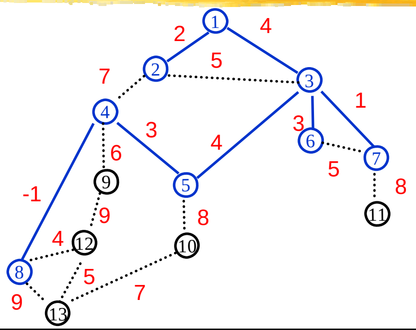
23

Prim’s Algorithm



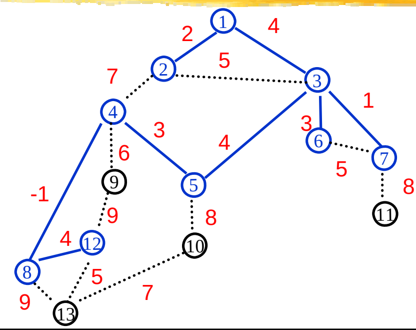
24

Prim’s Algorithm



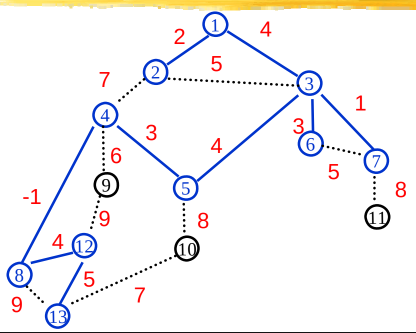
25

Prim’s Algorithm



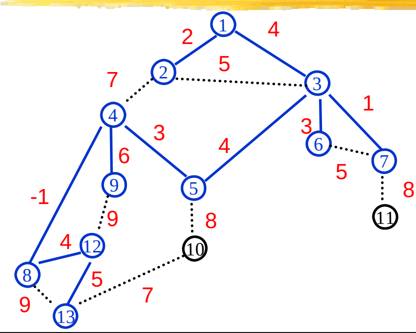
26

Prim’s Algorithm



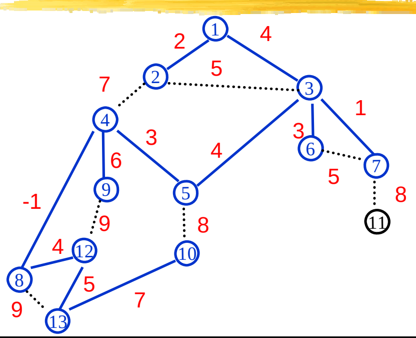
27

Prim’s Algorithm



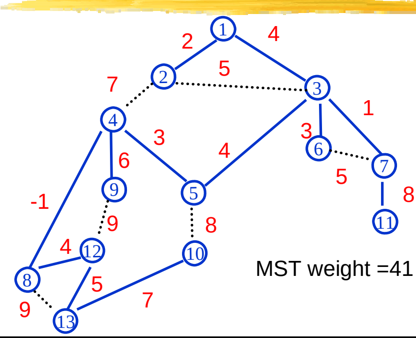
28

Prim’s Algorithm



29

Prim’s Algorithm



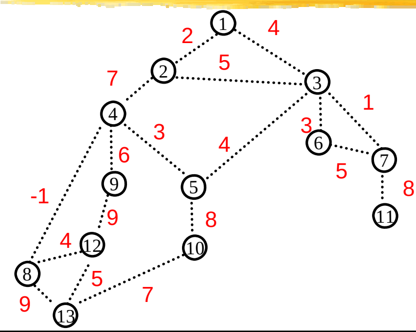
30

Second Greedy Algorithm

- Kruskal's Algorithm
  - Start with the vertices and no edges
  - Repeatedly add the cheapest edge that joins two different components. i.e. that doesn't create a cycle
- Again we save the proof of correctness for later

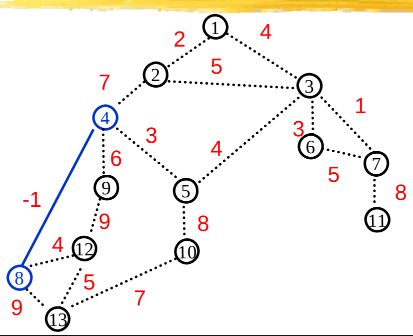
31

Kruskal's Algorithm



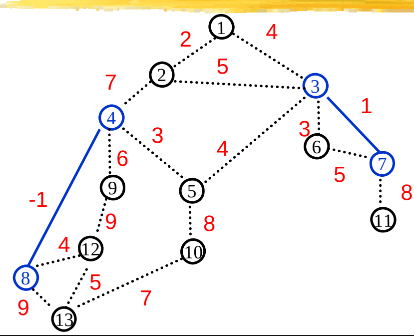
32

Kruskal's Algorithm



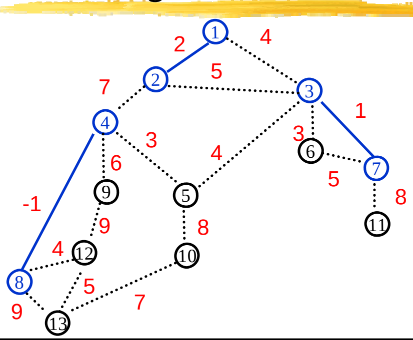
33

Kruskal's Algorithm



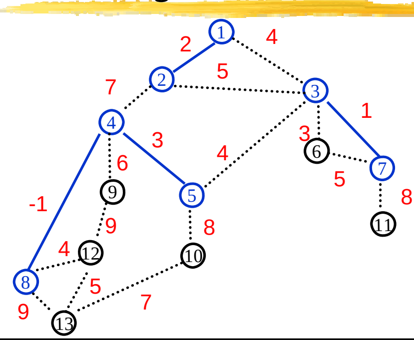
34

Kruskal's Algorithm



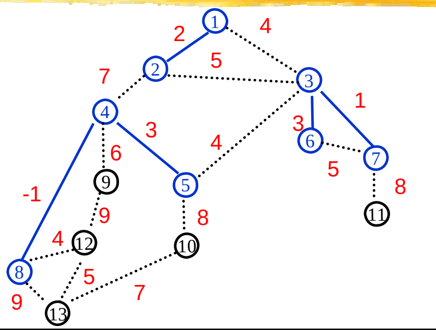
35

Kruskal's Algorithm



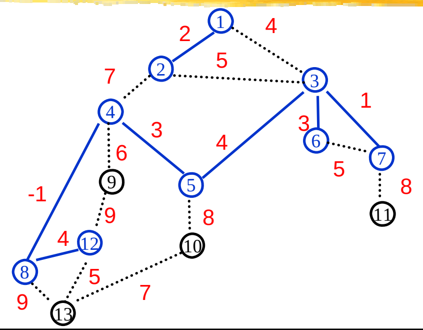
36

Kruskal's Algorithm



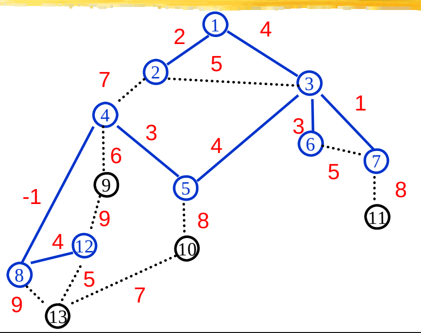
37

Kruskal's Algorithm



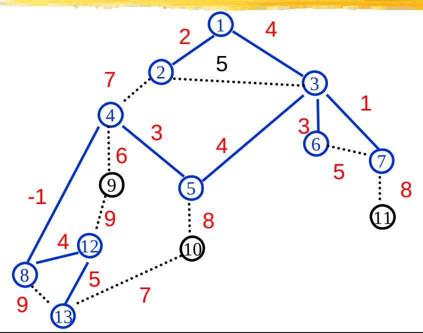
38

Kruskal's Algorithm



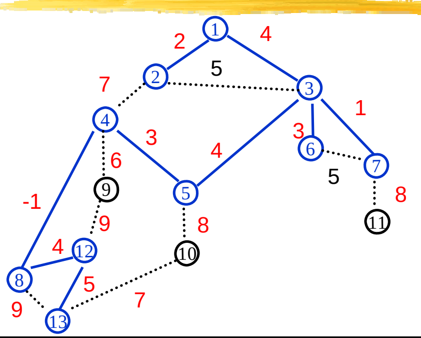
39

Kruskal's Algorithm



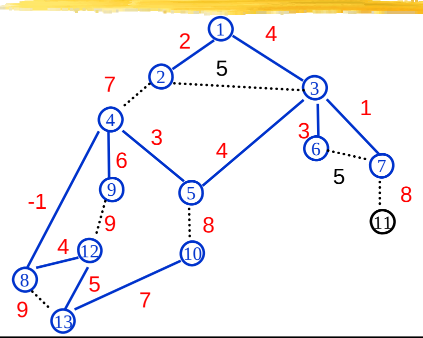
40

Kruskal's Algorithm



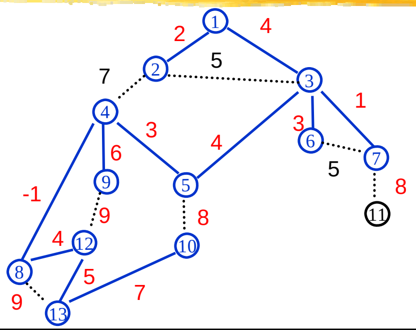
41

Kruskal's Algorithm



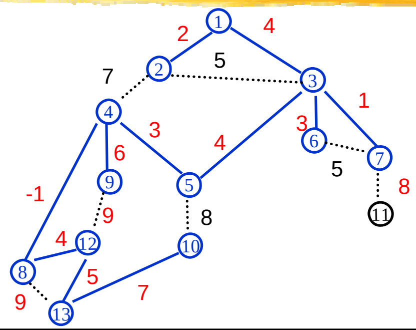
42

Kruskal's Algorithm



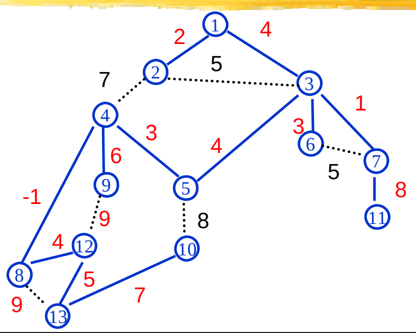
43

Kruskal's Algorithm



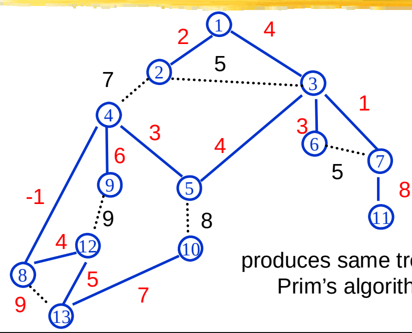
44

Kruskal's Algorithm



45

Kruskal's Algorithm



produces same tree as
Prim's algorithm

46