Graph Algorithms - II

CSIE 2136 Algorithm Design and Analysis, Fall 2018

https://cool.ntu.edu.tw/courses/61

Hsu-Chun Hsiao

Announcement

HW3 due in two weeks

Minhw#8 due next week

Midterm grade released

下週課程會事先錄影

Application of DFS: Topological Sort

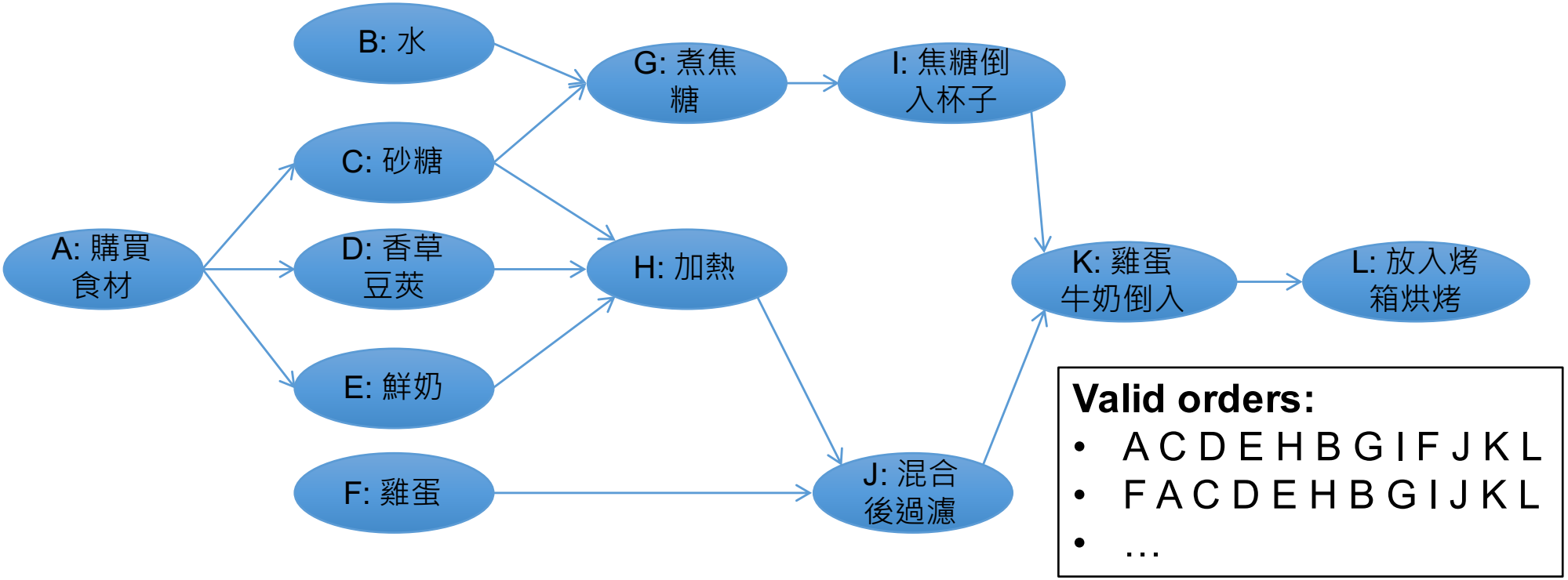
Textbook chapter 22.4

MasterChef: 布丁篇

A->B: 要先處理完A才能處理B

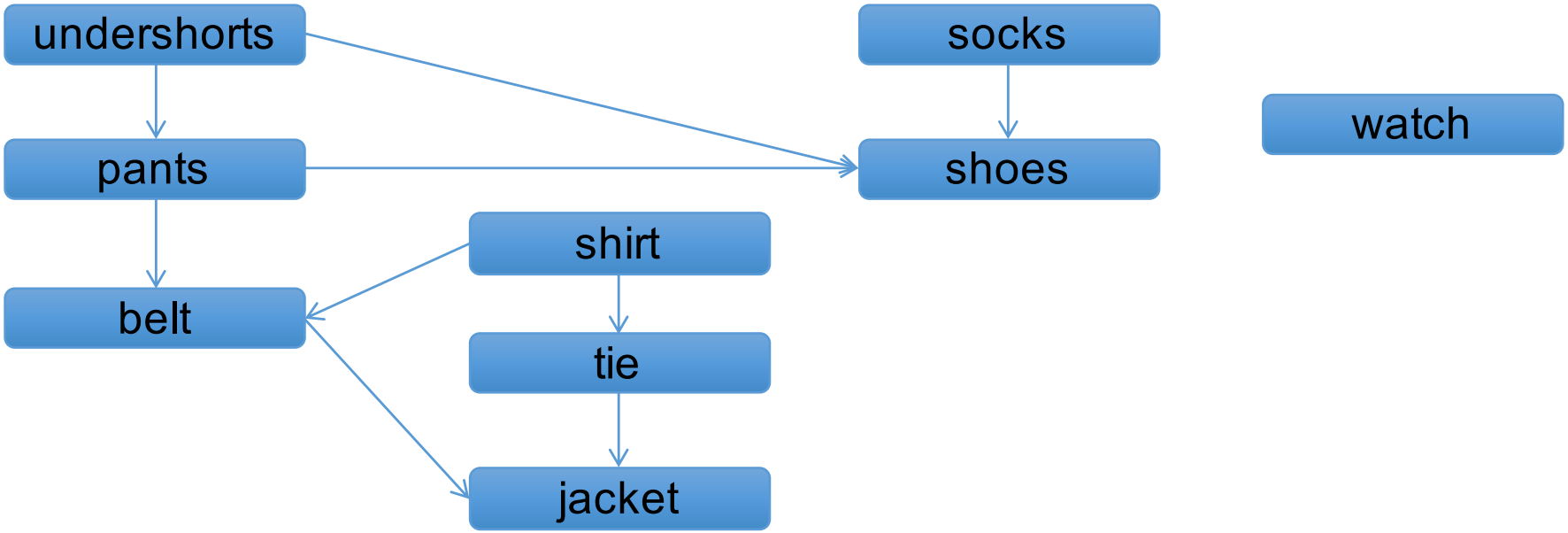
新手一次只能做一件事，用什麼順序才能順利做出布丁？

Intuition: 前置作業要先完成，才能做後面的步驟



Directed Acyclic Graphs (DAGs)

- A DAG is a directed graph with no cycles
- Often used to indicate precedence among events (X must happen before Y)
  - E.g., cooking, taking courses, clothing…



Topological Sort

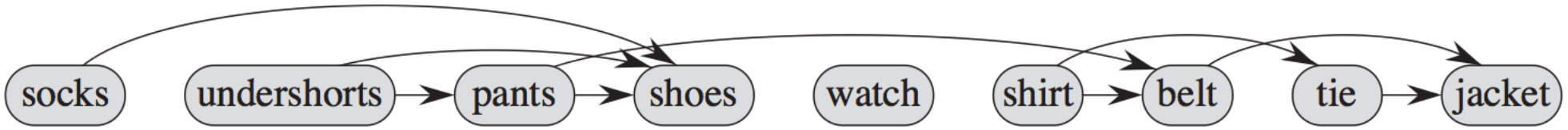
Input: a DAG G = (V, E)

Output: a linear ordering of all its vertices such that for all edges (u, v) in E, u precedes v in the ordering

Alternative view: a vertex ordering along a horizontal line so that all directed edges go from left to right

A DAG can have multiple valid topological orders

- E.g., watch can be placed anywhere in the following example



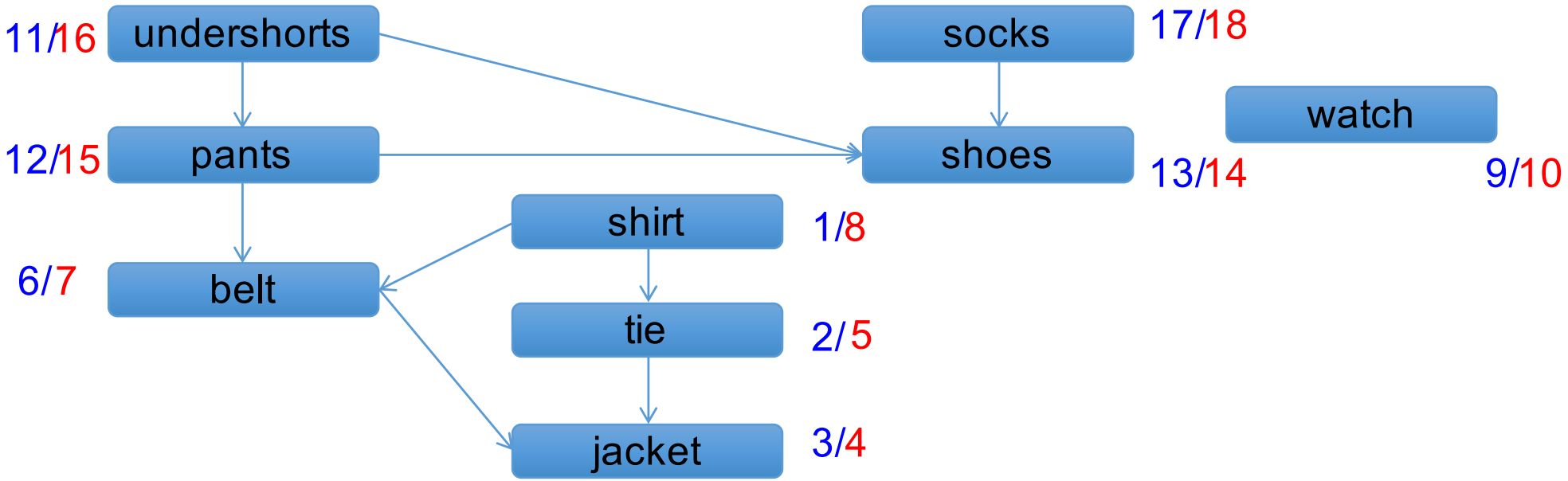
Topological sort algorithm

```
TOPOLOGICAL-SORT(G) //G is a DAG
Call DFS(G) to compute finishing times v.f for each vertex v
As each vertex is finished, insert it onto the front of a linked list
Return the linked list of vertices
```

Vertex u is in front of v if u.f > v.f

We will prove this linked list comprises a topological ordering

Topological sort using DFS



Running time analysis

```
TOPOLOGICAL-SORT(G) //G is a DAG
Call DFS(G) to compute finishing times v.f for each vertex v
As each vertex is finished, insert it onto the front of a linked list
Return the linked list of vertices
```

DFS with adjacency lists: $\Theta(V + E)$ time

Insert each vertex to the linked list: $\Theta(V)$ time

=> total running time is $\Theta(V+ E)$

Characterizing directed acyclic graphs

Lemma 22.11 A directed graph is acyclic ⇔ a DFS yields no back edges

Proof

The => direction: suppose there is a back edge (u,v)

- v is an ancestor of u in DFS forest
- There is a path from v to u in G and (u, v) completes the cycle

The <= direction: suppose there is a cycle c

- Let v be the first vertex in c to be discovered and u is a predecessor of v in c
- Upon discovering v the whole cycle from v to u is WHITE
- At time v.d, the vertices of c form a path of white vertices from v to u
- By the white-path theorem, vertex u becomes a descendant of v in the depth-first forest
- Therefore (u, v) is a back edge

Correctness of the topological sort algorithm

Theorem 22.12 The algorithm produces a topological sort of the input DAG

對所有的edge (u, v), 證明在此list中u一定在v前面（也就是 v.f < u.f 成立）

Proof

When (u,v) explored, u is gray. We can distinguish three cases

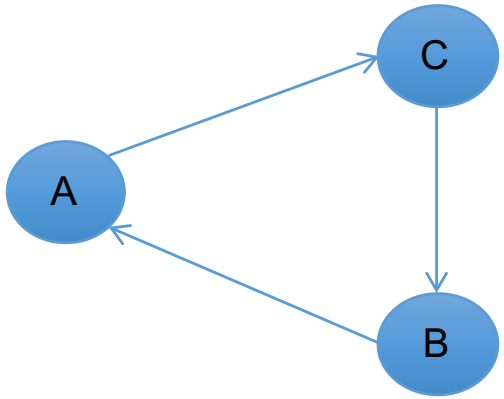
- v = gray
  - (u,v) = back edge (contradicting Lemma 22.11), so v cannot be gray
- v = white
  - v becomes descendant of u
  - v will be finished before u
  - v.f < u.f
- v = black
  - v is already finished
  - v.f < u.f

Discussion

Since cycle detection becomes back edge detection (Lemma 22.11), DFS can be used to test whether a graph is a DAG

Is there a topological order for cyclic graphs?

Given a topological order, is there always a DFS traversal that produces such an order?



Minimum Spanning Trees

Textbook Chapter 23

Spanning tree

Spanning tree of a graph G = a subgraph that is a tree and connects all the vertices

- Exactly n-1 edges
- Acyclic

There can be many spanning trees of a graph

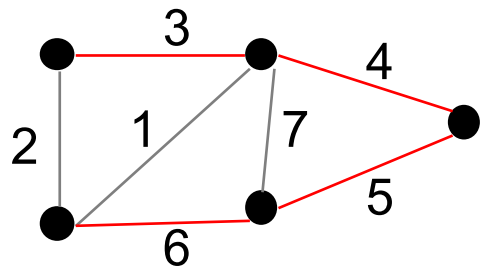
BFS and DFS also generate spanning trees

- BFS tree is typically “short and bushy”
- DFS tree is typically “long and stringy”

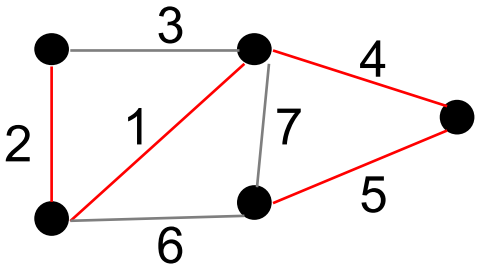
Minimum spanning tree (MST)

A minimum spanning tree of a graph G = a spanning tree with minimal weight

Weight of a spanning tree = the sum of weights of all edges in G



Weight = 18



Weight = 12, MST

Minimum spanning tree (MST)

Q: What if the graph is unweighted?

The problem becomes trivial

Q: What if the graph contains edges with negative weights?

Add a large constant to every edge; a MST remains the same

Q: Given G, can there be more than one MST?

Yes, consider an unweighted graph: every spanning tree is an MST.
But we will show that MST is unique if all edge weights are distinct.

Minimum spanning tree (MST)

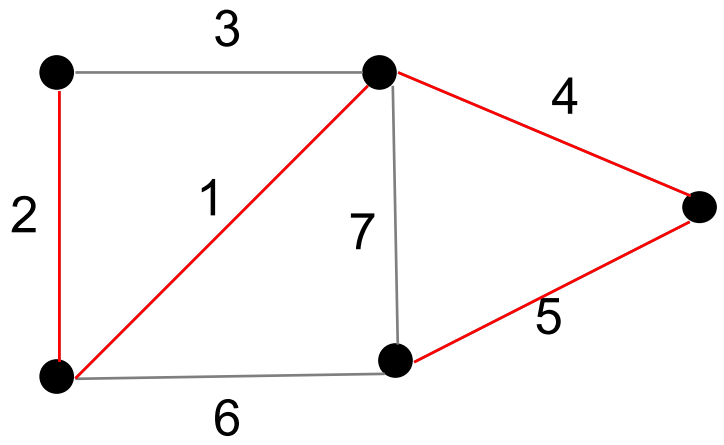
Finding an MST is an optimization problem.

Two greedy algorithms compute an MST:

- **Kruskal's algorithm**: consider edges in ascending order of weight. At each step, select the next edge as long as it does not create cycle.
- **Prim's algorithm**: start with any vertex *s* and greedily grow a tree from *s*. At each step, add the edge of the least weight to connect an isolated vertex.

Kruskal's algorithm

```
Kruskal(G)
Start with T = V (no edges)
For each edge in increasing order by weight
    If adding edge to T does not create a cycle
        Then add edge to T
```

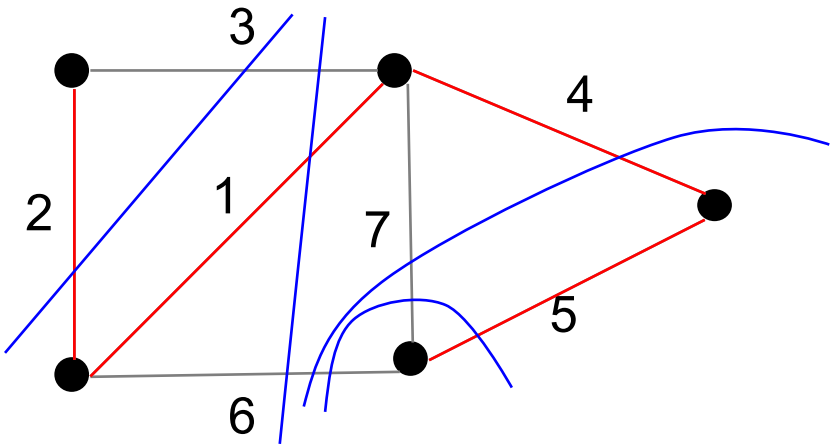


Weight = 12
MST

Running time depends on how the cycle test is implemented.
Using a disjoint-set data structure, running time = O(E log V)
(will show the details later)

Prim's Algorithm

```
Prim(G)
Start with a tree T with one vertex (any vertex)
While T is not a spanning tree
    Find least-weight edge that connects T to a new vertex
    Add this edge to T
```



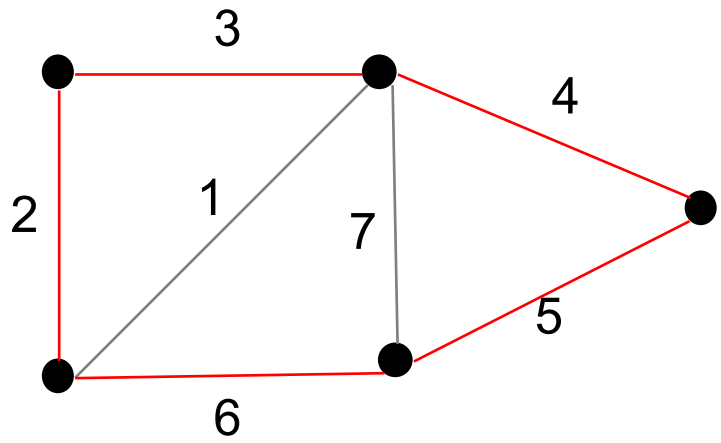
Weight = 12
MST

Running time depends on how finding least-weight edge implemented.
Using a binary min-heap, running time = O(E log V)
(will show the details later)

Cycle property

Let *C* be any cycle in the graph G, and let *e* be an edge with the maximum weight on *C*. Then the MST does not contain e.

- For simplicity, assume all edge weights are distinct (thus unique MST)



No MST contains the edge of cost 6

Cycle property

Let *C* be any cycle in the graph *G*, and let e be an edge with the maximum weight on *C*. Then the MST does not contain *e*.

Proof by contradiction

Suppose *e* is in the MST

Then removing *e* disconnects the MST into two components *T1* and *T2*

There exists another edge *e'* in *C* that can reconnect T1 & *T2*

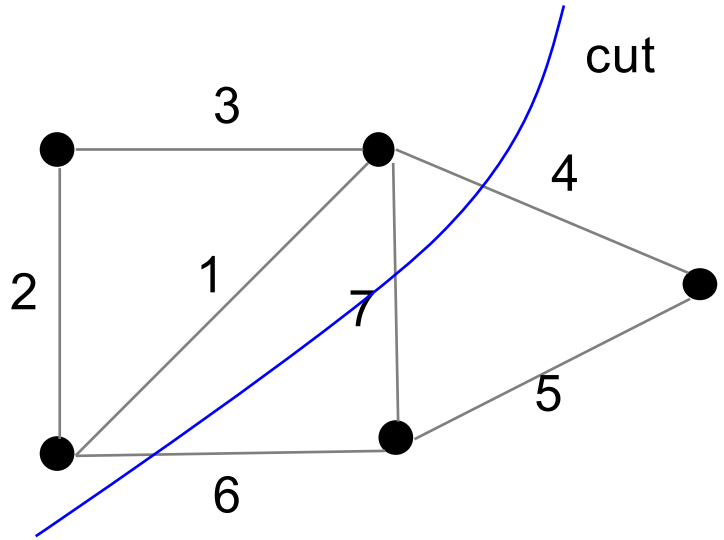
Since weight(e') < weight(e), the new tree has a lower weight

Contradiction!

Cut property

Let *C* be a cut in the graph, and let *e* be the edge with the minimum cost in *C*. Then the MST contains *e*.

- Cut = a partition of the vertices
- For simplicity, assume all edge weights are distinct (thus unique MST)



There is an MST containing the edge of cost 4

Cut property

Let *C* be a *cut* in the graph, and let *e* be the edge with the minimum cost in *C*. Then the MST contains *e*.

Proof by contradiction

Suppose *e* is not in the current MST

Adding *e* creates a cycle in the MST

There exists another edge *e'* in *C* that can break the cycle

Since weight(*e'*) > weight(*e*), the new tree has a lower weight

Contradiction!

Uniqueness of MST

Given G, can there can be more than one MST?

MST is unique if all edge weights are distinct (Why?)

Proof by contradiction

- Suppose there are two MSTs A and B
- Let e be the least-weight edge in A $\cup$ B and e is not in both
- WLOG, assume e is in A
- Add e to B; {e} $\cup$ B contains a cycle C
- C includes at least one edge e' that is not in A
- In B, replacing e' with e yields a MST with less cost
- Contradiction!

When edge costs are not distinct

For proof purpose, we can break tie and ensure a unique MST by applying a lexicographical order of edges

Define a new weight function w' over edges such that

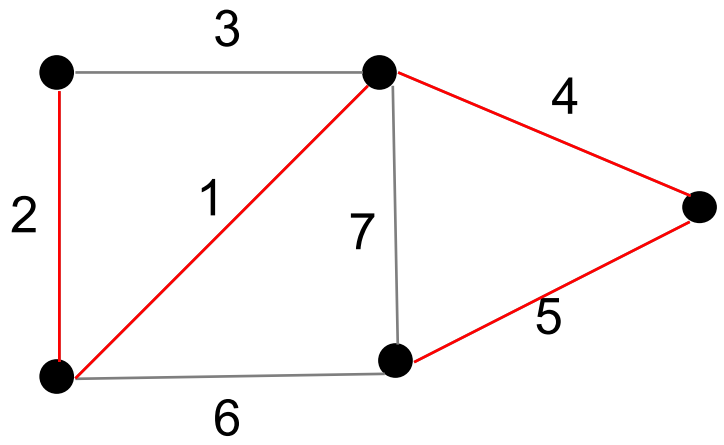
- $w'(e_i) < w'(e_j)$ if $w(e_i) < w(e_j)$ or $(w(e_i) = w(e_j)$ and $i < j)$
- $w'(S_i) < w'(S_j)$ if $w(S_i) < w(S_j)$ or $(w(S_i) = w(S_j)$ and $S_i \backslash S_j$ has a lower indexed edge than $S_j \backslash S_i)$

Hence, there is a unique MST w.r.t. to this new weight function w'

Note: Prim and Kruskal algorithms don't require the weights to be distinct. The above is needed for the proof purpose only.

Kruskal's algorithm

```
Kruskal(G)
Start with T = V (no edges)
For each edge e in increasing order by weight
    If adding edge e to T does not create a cycle
        Then add edge e to T
```



Weight = 12
MST

Implementation of Kruskal's algorithm

```
MST-KRUSKAL(G,w)  // w = weights
A = empty // edge set of MST
for v in G.V
    MAKE-SET(v)
sort the edges of G.E into non-decreasing order by weight w
for (u,v) in G.E, taken in non-decreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {u, v}
         UNION(u,v)
return A
```

Disjoint-set data structure

- MAKE-SET, FIND-SET, UNION

Each set contains the vertices in one tree of the current forest

Running time analysis

The ***amortized cost*** *of the* disjoint-set-forest implementation with *union-by-rank* only (textbook Chapter 21):

- MAKE-SET = O(1)
- FIND-SET = O(log V)
- UNION = O(log V)
- The amortized cost of *m* operations on *n* elements is O(*m log n*) (Exercise 21.4-4)

Sort edge = O(E log E) = O(E log V)

- log E = O(log V) because E is at most V^2

Running time of Kruskal = O(E log V)

Correctness of Kruskal's algorithm

Theorem Kruskal's algorithm computes the MST

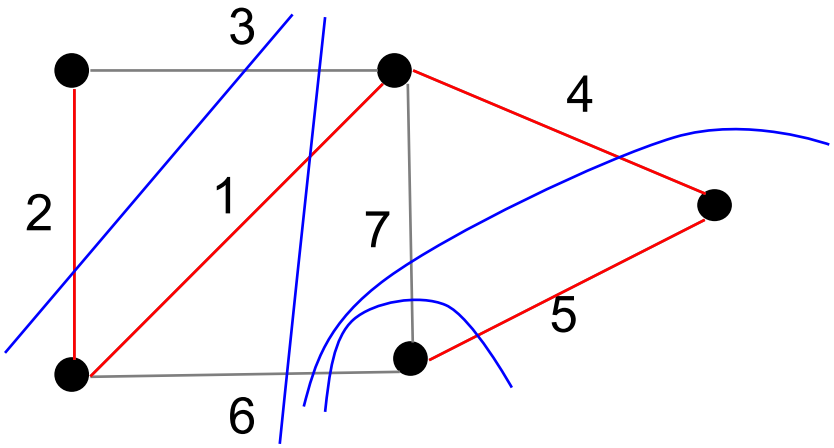
Proof

Consider whether adding e creates a cycle:

1. If adding e to T creates a cycle C
   - Then e is the max weight edge in C
   - The **cycle property** ensures that e is not in the MST
2. If adding e = (u, v) to T does not create a cycle
   - Before adding e, the current set contains at least two trees T1 and T2 such that u in T1 and v in T2
   - e is the minimum cost edge on the cut of T1 and T2
   - The **cut property** ensures that e is in the MST

Prim's Algorithm

```
Prim(G)
Start with a tree T with one vertex (any vertex)
While T is not a spanning tree
    Find least-weight edge that connects T to a new vertex
    Add this edge to T
```



Weight = 12
MST

Implementation of Prim's algorithm

```
MST-PRIM(G, w, r) //w = weights, r = root
for u in G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ empty
    u = EXTRACT-MIN(Q)
    for v in G.adj[u]
        if v ∈ Q and w(u,v) < v.key
            v.π = u
            v.key = w(u,v) //DECREASE-KEY
```

Q = min-priority queue, containing vertices not yet in the tree

v.key = minimum weight of any edge connecting v to the tree

v.π = the parent of v in the tree

Running time analysis

Binary min-heap (textbook Chapter 6)

- BUILD-MIN-HEAP = O(V)
- EXTRACT-MIN = O(log V)
- DECREASE-KEY = O(log V)

Running time of Prim = O(V log V + E log V)
= O(E log V), because V = O(E) in a connected graph

Can be improved to O(E + V log V) using Fibonacci heaps (textbook Chapter 19)

Correctness of Prim’s algorithm

Theorem Prim's algorithm computes the MST

Proof

1. All edges found by Prim’s are in the MST:
   - Let S be the subset of vertices in current tree T
   - Prim’s algorithm adds the cheapest edge e with exactly one endpoint in S
   - The **cut property** ensures that e is in the MST
2. All edges in the MST are found by Prim’s:
   - Why?