Algorithms and Data Structures (CSci 115)

California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning outcomes

- **Minimum Spanning Trees**
  - Definitions
  - How to solve it with:
    1. Kruskal's algorithm
    2. Prim's algorithm
  - What data structures to consider to solve Kruskal and Prim's algorithms

Rationale

- Example
  - Interconnection of a set of n pins in an electronic circuit
    - Arrangement of n-1 wires (edges), each connecting 2 pins (vertices)
    - Solution with the least amount of wire = the most desirable
- Input: Undirected Graph G=(V,E)
  - V: set of pins
  - E: set of wires
  - w(u,v): amount of wire needed to connect u and v
  - What we want:
    - an **acyclic** subset T $\subseteq$ E connecting all the vertices , acyclic → must form a tree !
    - Total weight $\mathrm{w}(\mathrm{T}) = \sum_{(u,v)\epsilon T} w(u,v)$
- Finding T: solving the minimum spanning tree problem

Solutions

- G=(V,E) , w: E → R
- Greedy approaches
  - Kruskal's algorithm
  - Prim's algorithm
- Principle
  - The minimum spanning tree grows 1 edge at a time
    - The generic method manages a set of edges A with the loop invariant:
      - **Prior to each iteration, A is a subset of some minimum spanning tree**
- At each step, we determine (u,v) that we can add to A
  - without violating this invariant,
    - A U {(u,v)} is **also** a subset of a minimum spanning tree
    - **(u,v) = a safe edge for A**

Solutions

- Pseudo-code
  - Very generic
  - Tough part: line 3 ☺
    - How to find safe edges?
    - A is always acyclic,
      - otherwise, an MST including A would contain a cycle, which is a contradiction
    - $G_A=(V,A)$ it is a forest
  - Loop invariant
    - **Initialization:**
      - After line 1, A satisfies the loop invariant.
    - **Maintenance:**
      - Loop (lines 2–4) keeps the invariant **by adding only safe edges**.
    - **Termination:**
      - All edges added to A are in an MST and
      - A that is returned (line 5) must be an MST

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

Safe edges

- Definitions
  - A **cut** (S,V-S) of an undirected graph G=(V,E) is a partition of V.
  - An edge (u,v) **crosses** the cut (S,V-S) if
    - one of its endpoints is in S and the other is in V-S
  - A cut **respects** a set A of edges if **no** edge in A crosses the cut.
  - An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut.
    - Remark: there can be more than one light edge crossing a cut in the case of ties !!
    - In general, an edge is a **light edge** satisfying a given property if its weight is the minimum of **any** edge satisfying the property.

Safe edges

- **Theorem**
  - G=(V,E) is a connected undirected graph
    - With real-valued weight function w (w : E → R)
  - A is a subset of E that is **included** ($\subseteq$) in some MST for G
  - (S,V-S) any cut of G **that respects** A
  - (u,v) a light edge crossing (S,V-S)

➢ **Then** (u,v) is a safe edge for A

Safe edges

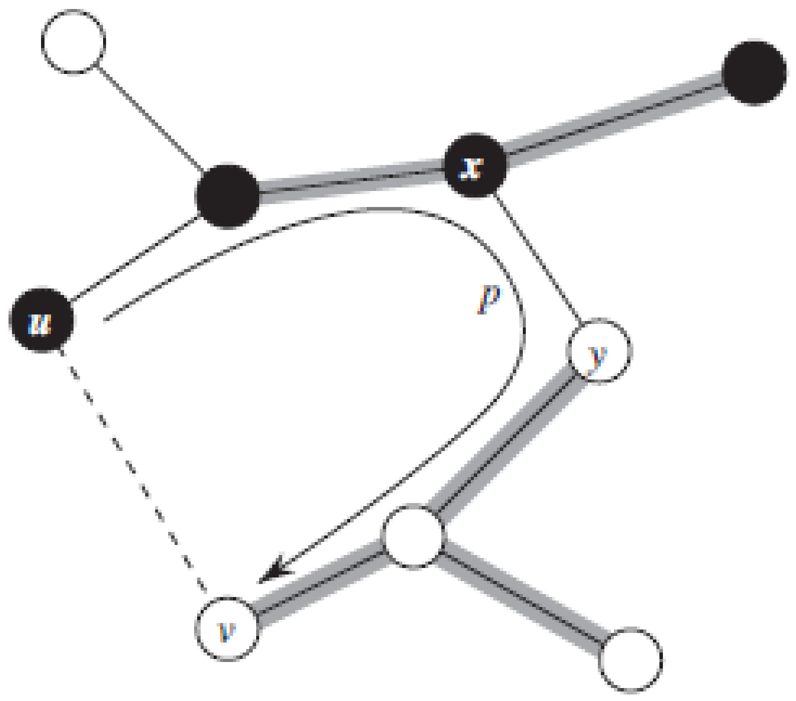
- **Proof** (part 1)
  - Let T be an MST that includes A, and assume that T **does not** contain the light edge (u,v)
    - because if it is the case, we are done.
  - We shall construct another minimum spanning tree T' including A U {(u,v)}
    - by using a **cut-and-paste** technique, to show that (u,v) is a safe edge for A.
  - The edge (u,v) forms a cycle with the edges on the simple path p from u to v in T
  - As u and v are on opposite sides of the cut (S,V-S)
    - Then at least one edge in T lies on the simple path p and also crosses the cut.
  - Let (x,y) be any such edge.
    - Remark: the edge (x,y) is not in A, because the cut respects A.
  - Since (x,y) is on the unique simple path from u to v in T , removing (x,y) breaks T into 2 parts.
  - Adding (u,v) reconnects them to form a new spanning tree
    - T'=T-{(x,y)} U {(u,v)}.

Safe edges

- **Proof** (part 2)
  - We show that T' is an MST
    - As (u,v) is a light edge crossing (S,V-S) and (x,y) also crosses this cut:
      - $w(u,v) \leq w(x,y)$
      - → $w(T') = w(T)-w(x,y)+w(u,v)$
      - $\leq w(T)$
  - T is an MST (first hypothesis) → $w(T) \leq w(T')$ → T' is also an MST !☺
  - A $\subseteq$ T' because A $\subseteq$ T and (x,y) is not in A → A U {(u,v)} $\subseteq$ T'
  - As T' is an MST → (u,v) is **safe for A**

Safe edges

- Example
  - **Vertices**
    - Black in S
    - White in V-S
  - **Edges**
    - Shown in the MST T are shown,
    - (Not shown in the graph G)
    - Edges in A are shaded and (u,v) is a light edge crossing the cut (S,V-S)
    - The edge (x,y) is an edge on the **unique** simple path p from u to v in T
  - To form an MST T' that contains (u,v)
    - Remove the edge (x,y) from T and add the edge (u,v)



Kruskal's algorithm

- **Principle**
  - The set A is a forest whose vertices are all those of the given graph.
  - The safe edge added to A is *always* a **least-weight edge** in the graph that **connects 2 distinct components**
- **What we need**
  - A disjoint-set data structure to maintain several disjoint sets of elements
  - Each set contains the vertices in one tree of the current forest
  - Operations
    - Find-Set(u) returns a representative element from the set that contains u.
      - → determine whether 2 vertices u and v belong to the same tree
      - by testing whether Find-Set(u)==Find-Set(v)
    - Union(u,v) : to combine trees

Kruskal’s algorithm

- Pseudo-code
  - Remark: It is better to use Adjacency lists for the implementation

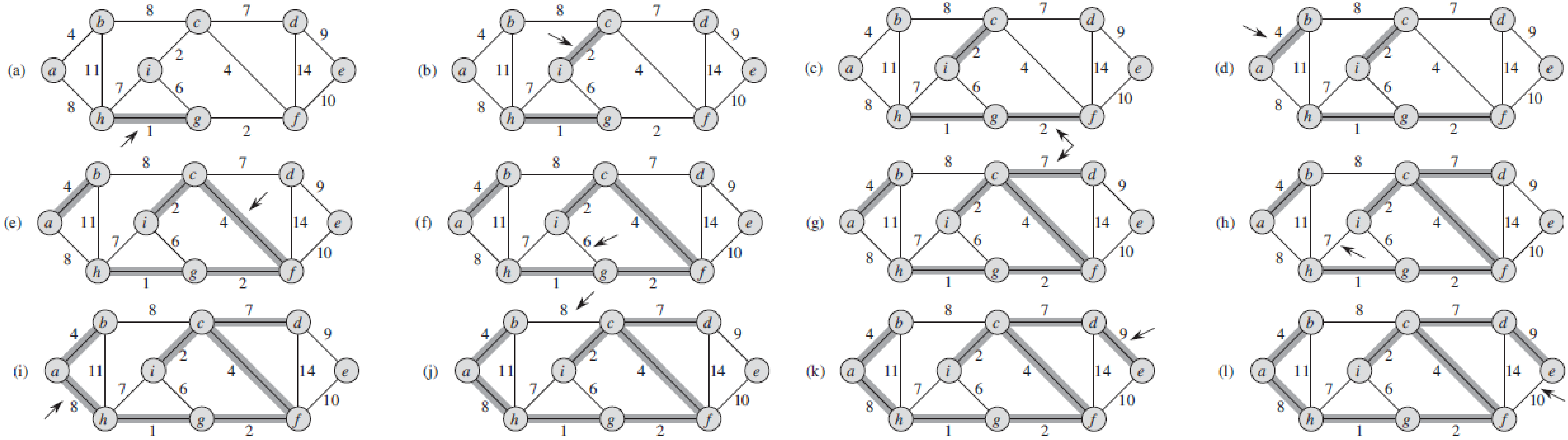
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

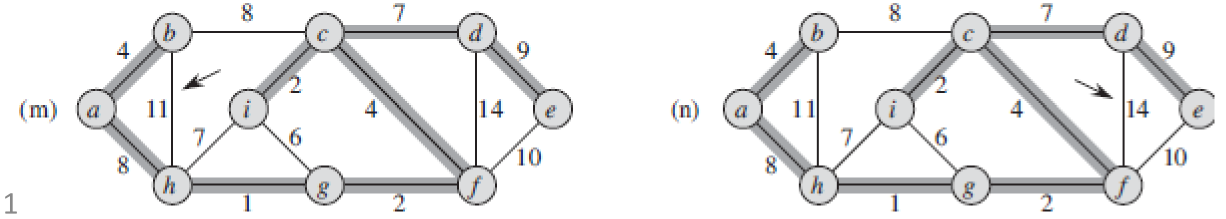
Remark: To avoid ambiguity: Increasing vs. Non-decreasing

- Example
  - Increasing: 1,2,3,4 → Increasing sequence, for x(n) and x(n+1), x(n+1) > x(n)
  - Nondecreasing: 1,1,2,3 → Nondecreasing sequence, for x(n) and x(n+1), x(n+1) >= x(n)

Kruskal’s algorithm

- Example





- Shaded edges in the forest A being grown.
- Algo: it considers each edge in **sorted order by weight**.
- An arrow points to the edge under consideration at each step of the algorithm.
- If the edge joins 2 distinct trees in the forest, it is added to the forest, thereby merging the 2 trees.

Prim's algorithm

- **Principle**
  - The set A forms a single tree.
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex **not** in the tree.
- Property
  - The edges in the set A always form a single tree.
- The tree starts from an **arbitrary** root vertex r and grows until the tree spans all the vertices in V
    - Each step adds to A a light edge that connects A to an **isolated** vertex
      - one on which no edge of A is incident. (edges that are safe for A)
  - → at the end, edges form by A = MST
- Greedy because:
  - At each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight

Prim's algorithm

- Implementation strategy
  - Need of a fast way to select a new edge to add to the tree formed by the edges in A.
  - **Inputs**:
    1. G
    2. the root r of the minimum spanning tree to be grown
  - During execution of the algorithm:
    - All vertices that are **not** in the tree reside in a **min-priority queue** Q based on a key attribute.
    - For each vertex , the attribute v.key is the minimum weight of any edge connecting to a vertex in the tree
    - If there is no such edge, we have $v.key=\infty$
    - The attribute $v.\pi$ names the **parent** of v in the tree.
  - The algorithm implicitly maintains the set A from GENERIC-MST as:
    - $A=\{(v,v.\pi)\} : v \in V- \{r\} - Q\}$
  - When the algorithm ends
    - The min-priority queue $Q=\emptyset$
    - The MST A for G is
    - $A=\{(v,v.\pi)\} : v \in V- \{r\} \}$

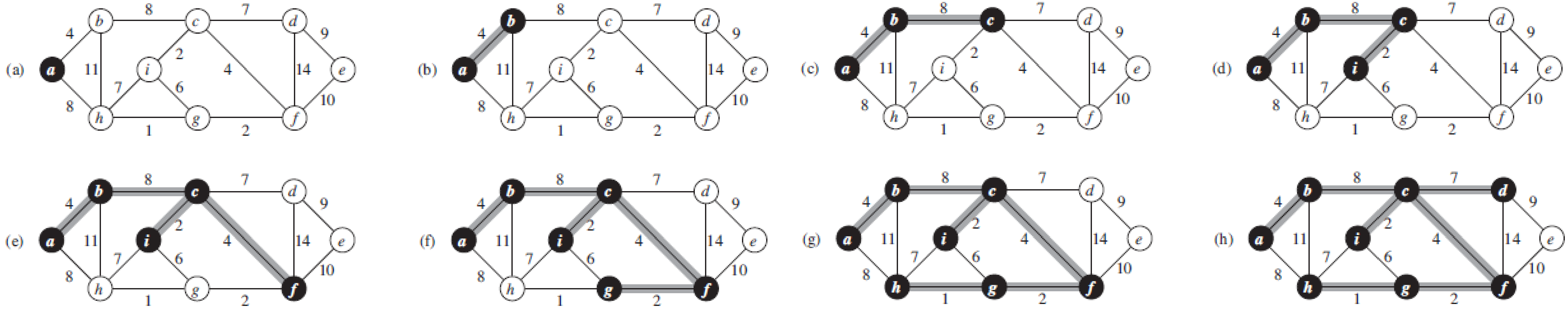
Prim’s algorithm

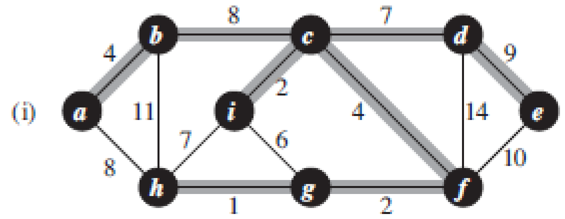
- Pseudo-code

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Prim’s algorithm

- Example





The root vertex is a (start).
Shaded edges are in the tree being grown, and black vertices are in the tree.
At each step of the algo: the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree.
Special case: In the 2nd step, the algorithm has a choice of adding either (b,c) or (a,h) to the tree because both are light edges crossing the cut.

Conclusion

- Minimum Spanning Trees (MST)
  - Definitions:
    1. Cut
    2. Light edges
    3. Safe edge
  - You should know the principles and how to implement (pseudo-code & C++):
    - Kruskal's algorithm (**greedy**)
      - Binary heap: O(E log V)
      - *Start with an edge…*
    - Prim's algorithm (**greedy**)
      - Binary heap: O(E log V)
      - Fibonacci heap: O(E + V log V)
      - *Start with a node…*

Questions ?

- Reading
  - ➢Csci 115 book – Section 9.7
  - ➢Introduction to Algorithms, 3rd edition, **Chapter 23**