Generic MST algorithm

\[
\text{GENERIC-MST}(G,w) \\
A = 0 \\
\textbf{while} A \text{ is not a spanning tree} \hspace{1em} \textbf{do} \\
\hspace{1em} \text{find an edge } (u, v) \text{ that is safe for } A \\
\hspace{2em} A = A \cup \{(u, v)\} \\
\textbf{return} A
\]
Kruskal’s Algorithm for MST

- $G = (V, E)$ is a connected, undirected, weighted graph. $w : E \rightarrow \mathbb{R}$
  - Starts with each vertex being its own component
  - Repeatedly merges two components into one by choosing the light edge that connects them (i.e., the light edge crossing the cut between them)
  - Scans the set of edges in monotonically increasing order by weight
  - Uses a disjoint-set data structure to determine whether an edge connects vertices in different components
The Algorithm

KRUSKAL(V,E,w)
A=0
for each vertex v ∈ V do MAKE-SET(v)
sort E into non-decreasing order by weight w
for each (u,v) taken from the sorted list do
  if FIND-SET(u) ≠ FIND-SET(v)
  then A=A∪{(u, v)}
      UNION(u,v)
return A
Prim’s Algorithm for MST

- Builds one tree, so A is always a tree
- Starts from an arbitrary “root” r
- At each step, find a light edge crossing cut \((V_A, V \setminus V_A)\), where \(V_A\) = vertices that the tree A is incident on
- Add this edge to A
Selecting Edges Efficiently

- Use a min-priority queue $Q$ based on a key field
  - For each $v$, $\text{key}[v]$ is the minimum weight of any edge $(u,v)$, where $u \in V_A$
- The vertex returned by $\text{EXTRACT-MIN}$ is $v$ such that there exists $u \in V_A$ and $(u,v)$ is a light edge crossing $(V_A, V \setminus V_A)$
- Key of $v$ is $\infty$ if $v$ is not adjacent to any vertices in $V_A$
Prim’s MST

- The edges of $A$ will form a rooted tree with root $r$
  - $r$ is given as an input to the algorithm, but it can be any vertex
- $\pi[v] = \text{parent of } v$. $\pi[v] = \text{NIL}$ if $v = r$ or $v$ has no parent
- As algorithm progresses:
  \[ A = \{(v, \pi[v]) : v \in V \setminus \{r\} \setminus Q\} \]
- At termination $V_A = V \Rightarrow Q = 0$, so MST is:
  \[ A = \{(v, \pi[v]) : v \in V \setminus \{r\}\} \]
Prim, the Algorithm

PRIM(G,w,r)

for each u ∈ V do key[u]=∞; π[u]=NIL
key[r]=0; Q=V
while Q≠0 do
    u=EXTRACT-MIN(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v]
            then π[v]=u
            key[v]=w(u, v)
A Three-Part Loop Invariant

Prior to each iteration of the white loop:

1. $A = \{(v, \pi[v]) : v \in V \setminus \{r\} \setminus Q\}$
2. The vertices in $A$ (MST) are those in $V \setminus Q$
3. For all $v \in Q$, if $\pi[u] \neq \text{NIL}$ then $\text{key}[v] < \infty$ and $\text{key}[v] = w(v, \pi[v])$, with $\pi[v] \in A$
Prim’s Analysis

- Depends on the way Q is implemented
- Binary min-heap:
  - Init in $O(V)$
  - Total time for EXTRACT-MIN is $O(V \log V)$
  - For loop is executed $O(E)$ times, and for each time we decrease the key and modify the heap: $O(\log V)$
  - Total time: $O(V \log V + E \log V) = O(E \log V)$
  - Same as Kruskal’s
Improved Prim’s

- Use of a Fibonacci heap for Q
  - EXTRACT-MIN in $O(\log V)$
  - Decreasing the key in $O(1)$
    (amortized times)
  - Total time: $O(E + V \log V)$
Amortized Analysis

- Time required to perform a data structure operation is averaged over all the operation performed.
- Can be used to show that the average cost of an operation is small even though a single operation might be expensive.
- Different from average-case analysis → no probability here.
- Guarantees the average performance of each operation in the worst case.
Binary Heaps

- A binary heap is an (array) object that can be seen as a nearly complete binary tree.
- The tree is completely filled on all levels except, possibly, the lowest, which is partially filled from the left.
- Two kind of binary heaps:
  - Max-heaps, and
  - Min-heaps
Priority Queues

- A priority queue is a data structure for maintaining a set $S$ of elements, each with a key.

A min-priority queue supports the operations:

- $\text{Insert}(S,x)$, insertion
- $\text{Minimum}(S)$, returns the element with the largest key
- $\text{Extract-Min}(S)$, remove and returns the min
- $\text{Decrease-Key}(S,x,k)$ decrease the key of $x$ to the new value $k$ (assumed smaller than $\text{key}[x]$)
Heaps for Priority Queues

Given the operations on binary heaps, the operations on a priority queue cost:

- Insert: $O(\log n)$
- Minimum: $O(1)$
- Extract-Min: $O(\log n)$
- Decrease-Key: $O(\log n)$

A heap can support any priority queue operations on a set of size $n$ in $O(\log n)$ time (worst case)
Fibonacci Heaps

- Heap operations that do not involve deletion are implemented in $O(1)$ amortized time.
- Desirable when Extract-Min and Delete are small compared to other operations.
- A Fibonacci heap is a collection of trees.
- Not of practical use sometimes ...
Assignments

- Textbook, Chapter 23, pages 561—574
- Updated information on the class web page:
  www.ece.neu.edu/courses/eceg205/2003fa