Minimum Spanning Trees

Algorithms: Design and Analysis, Part II

Fast Implementation of Prim's Algorithm

Running Time of Prim's Algorithm

- Initialize $X = \{s\}$ [$s \in V$ chosen arbitrarily]
- $T = \emptyset$ [invariant: X = vertices spanned by tree-so-far T]
- While $X \neq V$
  - Let $e = (u, v)$ be the cheapest edge of G with $u \in X$, $v \notin X$.
  - Add e to T, add v to X.

Running time of straightforward implementation:
- $O(n)$ iterations [where $n = \#$ of vertices]
- $O(m)$ time per iteration [where $m = \#$ of edges]

$\Rightarrow O(mn)$ time

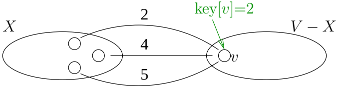
BUT CAN WE DO BETTER?

Prim's Algorithm with Heaps

[Compare to fast implementation of Dijkstra's algorithm]

Invariant #1: Elements in heap = vertices of $V - X$.

Invariant #2: For $v \in V - X$, key[v] = cheapest edge (u, v) with $i \in X$ (or $+\infty$ if no such edges exist).



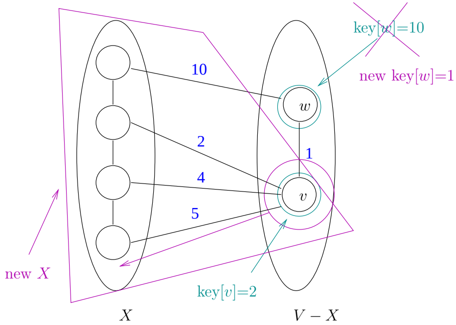
Check: Can initialize heap with $O(m + n \log n) = O(m \log n)$ preprocessing.

To compare keys | $n - 1$ Inserts | $m \geq n - 1$ since G connected

Note: Given invariants, Extract-Min yields next vertex $v \notin X$ and edge (u, v) crossing $(X, V - X)$ to add to X and T, respectively.

Quiz: Issue with Invariant #2

Question: What is: (i) current value of key[v] (ii) current value of key[w] (iii) value of key[w] after one more iteration of Prim's algorithm?

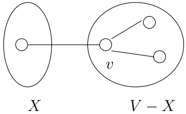


A) 11, 10, 4 C) 2, 10, 1

B) 2, 10, 10 D) 2, 10, 2

Maintaining Invariant #2

Issue: Might need to recompute some keys to maintain Invariant #2 after each Extract-Min.



Pseudocode: When v added to X:

- For each edge $(v, w) \in E$:
  - If $w \in V - X$ $\rightarrow$ The only whose key might have changed (Update key if needed:)
    - Delete w from heap
    - Recompute key[w]:=min{key[w],c_{vw}}
    - Re-Insert into heap

Subtle point/exercise:

Think through book-keeping needed to pull this off

Running Time with Heaps

- Dominated by time required for heap operations

- $(n-1)$ Inserts during preprocessing

- $(n-1)$ Extract-Mins (one per iteration of while loop)

- Each edge (v, w) triggers one Delete/Insert combo
 [When its first endpoint is sucked into X]

$\Rightarrow$ $O(m)$ heap operations [Recall $m \geq n-1$ since G connected]

$\Rightarrow$ $O(m \log n)$ time [As fast as sorting!]