Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 12

Wednesday February 1, 2017
based on notes by Denis Pankratov

Graph Algorithms

Definition of a Graph: a graph is a pair(V, E), where V is a set of vertices and E is a set of edges.

$$\text{Undirected Graph: } E \subseteq \big(V//2\big) = \{S \subseteq V \,||S| = 2\}$$

$$\text{Directed Graph: } E \subseteq V \times V$$

Examples:

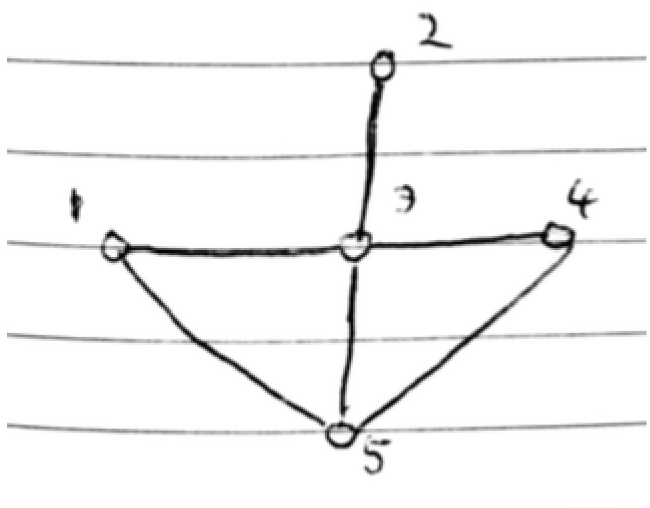

Figure 1: Undirected graph

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{\{1,3\}, \{2,3\}, \{4,3\}, \{5,3\}, \{5,4\}, \{1,5\}\}$$

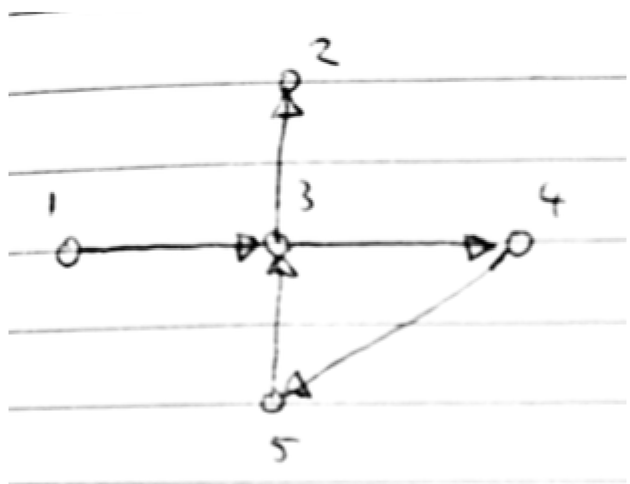

Figure 2: Directed graph

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{\{1, 3\}, \{3, 2\}, \{3, 4\}, \{4, 5\}, \{5, 3\}\}$$

Def: A weighted graph is a pair $(G = (V, E), w), w : E \to \mathbb{R}$

Standard Representations

1. List of edges / adjacencies - linked list of edges uses space $\mathcal{O}(|E|)$, linked list of names of vertives uses space $\mathcal{O}(|V|)$
2. Adjacency lists - Adj - array of size $|V|, \forall v \in V$, Adj$[v]$ - linked list of vertices adjacent to v, size $\mathcal{O}(|V| + |E|)$
3. Adjacency matrix A of $G = (V, E)$ is $|V| \times |V|$:

$$A_{u,v} = \begin{cases} 1, & \text{if } \{u, v\} \in E (\text{or } (u, v) \in E) \\ 0, & \text{otherwise} \end{cases}$$

Assumed Background: BFS, DFS, Union-Find datastructure

Example: Adj - adj. lists representation of $G = ([n], E)$

Construct Adj' - adj. lists representation of G so that $\forall v \in V$, Adj'$[v]$ is sorted in increasing order in time $\mathcal{O}(|V| + |E|)$.

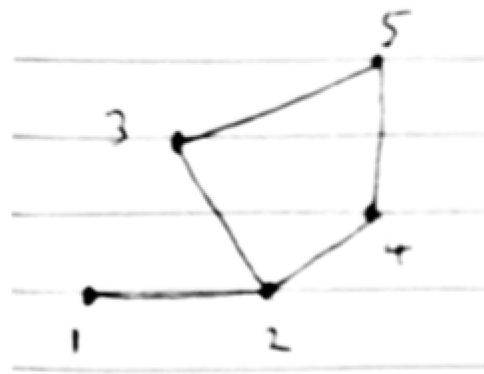

Figure 3: $Adj[2] = (4, 1, 3), Adj'[2] = (1, 3, 4)$

Definition: undirected $G = (V, E)$ is a tree if it is *connected* (each node is reachable from every other node) & *acyclic* (does not have closed walks [cycles])

Definition: $G = (V, E), G' = (V', E')$ is a subgraph of G denoted $G \subseteq G'$, if:

1. $V' \subseteq V$
2. $E' \subseteq E$, only vertices from V' appear in E'
3. $G' = (V', E') \subseteq G = (V, E)$ is called spanning if $V' = V$

Minimum Spanning Tree

Input: Adj - adj. lists of $G = (V, E)$, $w : E \to \mathbb{R}$

Output: $T \subseteq G$ - spanning tree of minimum weight

Kruskal's Algorithm

- Consider edges in increasing order of weights; keep adding the edges, diregarding those that create cycles
- Runtime $\mathcal{O}(|E| \log |E|)$ using Union-Find data structure

Prim's Algorithm

- Start with an arbitrary vvertex $s \in V$
- Keep growing the partial tree by adding a least-weight edge going outside of the tree

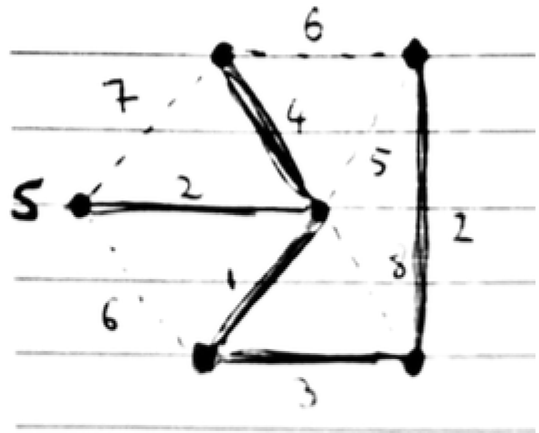

Figure 4: Prim is greedy: it *ignores future grabs*

Correctness: Definition: T_i - the tree constructed by Prim's algorithm after i steps (addition of an edge)

Loop Invariant: T_i can be extended to a *Minimum Spanning Tree* (MST) using edges not between vertices in T_i

Proof by induction on i: $i = 0, T_i = (\{s\}, \varnothing$ clearly extends to an MST using edges from E

Induction Assumption: Assume T_i extends to an MST T_i* for some $i \geq 0$

Induction Step: Let e be the edge chosen by Prim's algorithm in step $i + 1$

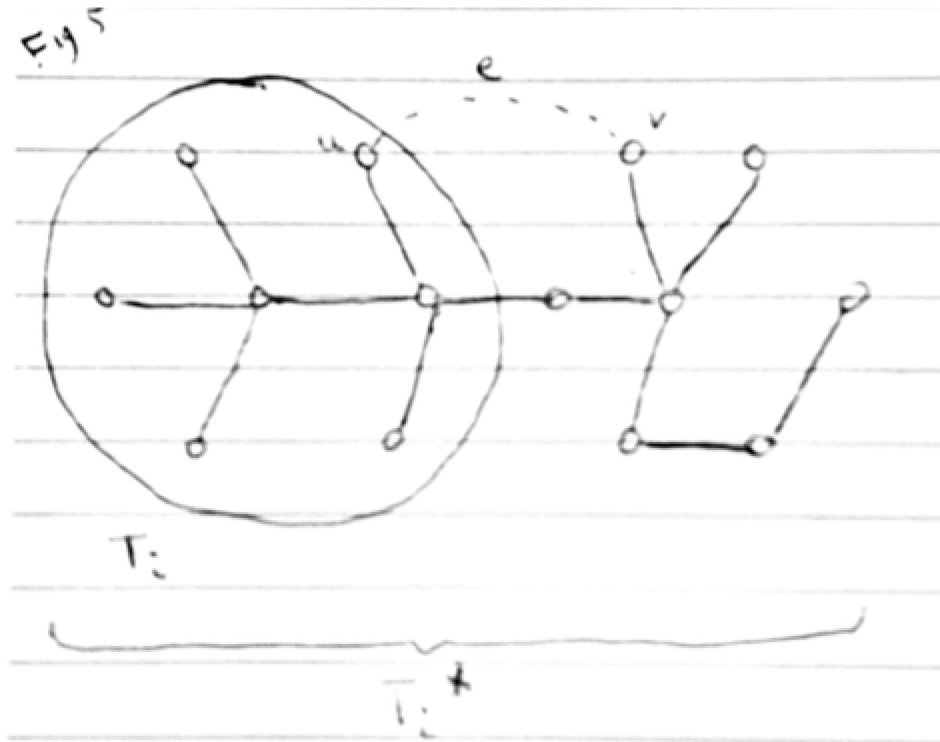

Figure 5: Visualization of problem

<u>Case 1:</u> $e \in T_i*$, then we are done

<u>Case 2:</u> $e \notin T_i*$, so adding e to T_i* creates a cycle C.

C contains an edge $e' \neq e$ that goes across the cut (partition) $(V(T_i), V(G) - V(T_i))$.

(Aside: $V(G)$ - vertices of G, $E(G)$ - edges of G)

By *greedy choice* of Prim's algorithm, $w(e) \leq w(e')$.

Removing e' from T_i* creates 2 connected components. Adding e reconnects them & gives a new tree $T_{i+1}* = (T_i * \ \{e'\}) \cup \{e\}$

$w(T_{i+1}*) = w(T_i*) - w(e') + w(e) \leq w(T_i*) \rightarrow T_{i+1}*$ is an MST and agrees with T_{i+1}.

<u>Algorithm:</u>

```
def Prims(Adj,w):
  Pick arbitrary s in V
  init arrays cost of size |V|, prev of size |V|

  for v in V:
    cost[v] = float('inf')
    prev[v] = None

  cost[s] = 0
```

```

  Q - MinPriorityQueue(V) /* by cost */

  while Q is not empty:
    v = Q.ExtractMin
    for u in Adj[v]:
      if w(v, w) < cost[u]:
        cost[u] = w(v,u) # causes decrease key
        prev[u] = v

  return prev
```

Runtime: using binary heap: $\mathcal{O}((|V| + |E|) \log |V|)$

Facts:

- $G = (V, E)$ is a tree $\rightarrow |E| = |V| - 1$
- $G = (V, E)$ is connected & $|E| = |V| - 1 \rightarrow G$ is a tree
- $G = (V, E)$ is a tree if and only iff there is a unique path between any two nodes