Algorithms and Data Structures

Module 2

Lecture 7

Minimum spanning trees: Prim’s algorithm

Adigeev Mikhail Georgievich

mgadigeev@sfedu.ru

Prim's algorithm

Given a connected graph $G(V, E)$, $|V| = n, |E| = m$.

1. $T(V_T, E_T): V_T = \{s\}, E_T = \emptyset$
2. Array C[1..*n*], P[1..*n*].
   - $C[s] = 0$; P[s]=*s*.
   - For each $v \in V \backslash \mathrm{V}_T: C[v] = w(s, v); P[v] = s$
3. While $V_T \neq V$:
   - Find $v \in V \backslash \mathrm{V}_T: v$ has minimum $C[v]$
   - Add v to V_T; add $(P[v], v)$ to E_T
   - Update_C&P(v).

Prim's algorithm

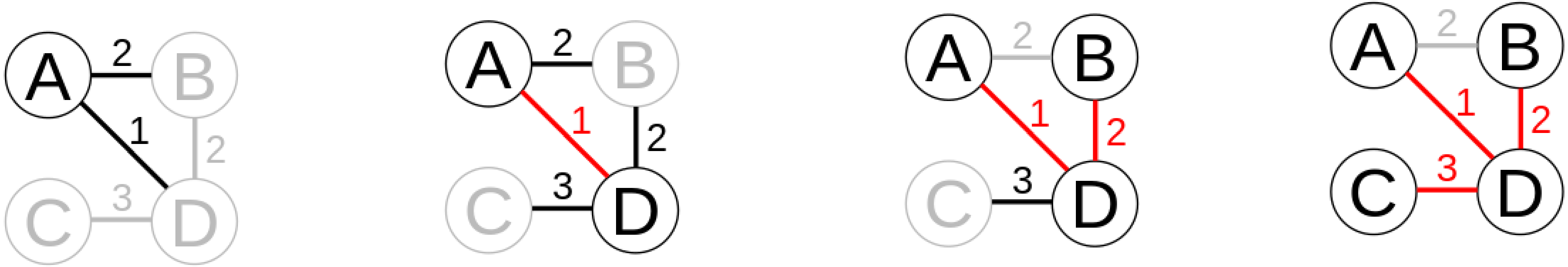
Update_C&P(v)

For each $(v, u) \in E$:

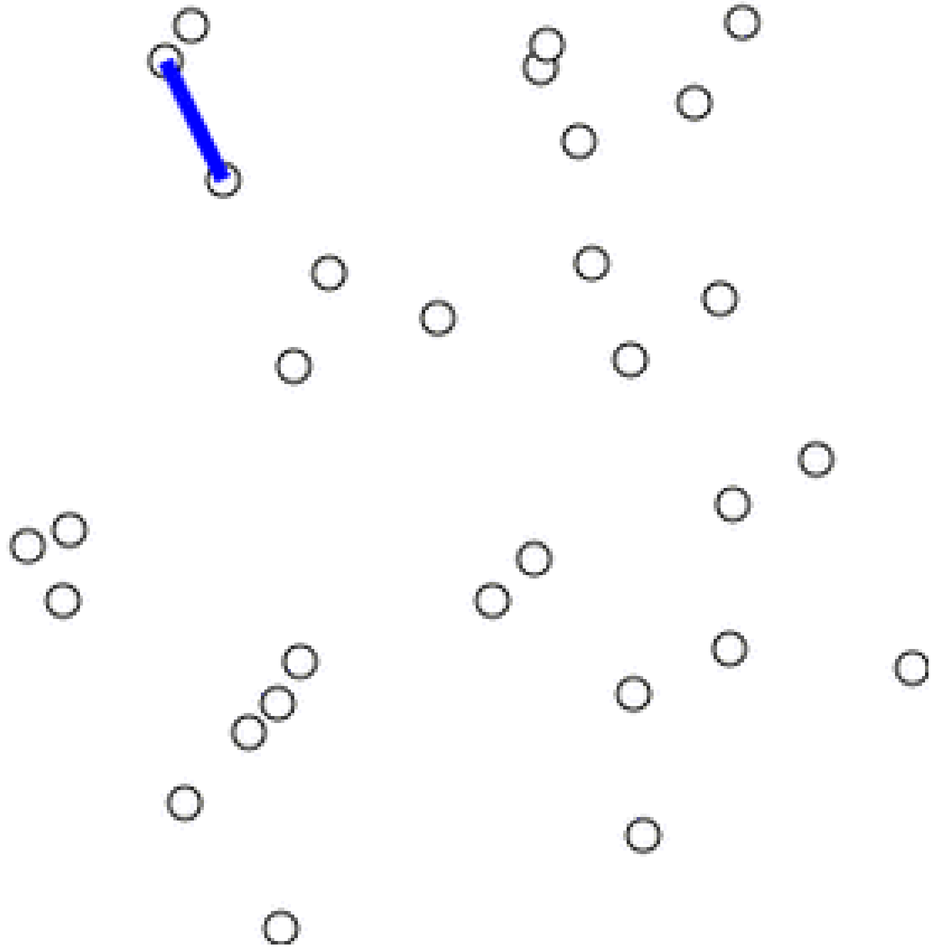
if $u \in V \backslash V_T$ and $C[u] > w(v, u)$:

$C[u] = w(v, u)$

$P[u] = v$



Prim’s algorithm



Prim's algorithm

Given a connected graph $G(V,E)$, $|V| = n, |E| = m$.

1. $T(V_T, E_T)$: $V_T = \{s\}, E_T = \emptyset$
2. Array C[1..*n*], P[1..*n*].
   - $C[s] = 0$; P[1..*n*]=*s*.
   - For each $v \in V \backslash V_T$: $C[v] = w(s,v)$; $P[v] = s$
3. While $V_T \neq V$: *n*-1 iterations
   - Find $v \in V \backslash V_T$: v has minimum $C[v]$???
   - Add v to V_T; add $(P[v], v)$ to E_T O(1)
   - Update_C&P(v). ???

Prim’s algorithm

Let us evaluate the total complexity of Update_C&P calls. Actually, we update C[] and P[] at most one time for each edge => the total complexity is $O(m)$.

The complexity of searching for the closest $v \in V \backslash V_T$ depends on the implementation.

Prim's algorithm

1) Naïve implementation: scan $V \backslash \mathrm{V}_T$ and search for the minimum value of $C[v]$. Each scan needs $O(n)$ time => the total time complexity is $O(m + n^2) = O(n^2)$.
2) Use a *priority queue* for keeping $C[v]$ and getting the minimum value at each iteration. The total complexity depends on the priority queue implementation:
   a) Binary heap: $O(m \ \log n)$
   b) Fibonacci heap: $O(m + n \log n)$

Priority queue: definition

- *Priority queue* is an abstract data structure which allows to efficiently append new items and select an item with the highest priority.
- ‘*Priority*’ means numeric values attached to items.
- ‘The highest’ means either ‘the maximum’ or ‘the minimum’ value of priority. Priority queue must be build as either ‘max’ or ‘min’ priority queue; for a max-priority queue one can select an item with the maximum priority and cannot select the minimum priority item, and vice versa.
- Priority queue is not a queue…

Priority queue: definition

Priority queue is an abstract data structure which efficiently implements operations:

- `Init(n)` – initialize an empty priority queue with *n* possible items.
- `Build(S)` – build priority queue containing items of S.
- `Add(x, prior)` – add item *x* with priority *prior* to the priority queue.
- `GetMin()` / `GetMax()` – get the item with the highest priority.
- `DelMin()` / `DelMax()` – delete the item with the highest priority.
- `ChangePriority(x, new_prior)` – change the priority of *x* to *new_prior*.

Priority queue: definition

For Prim’s algorithm we apply:

- At the initialization phase:
  - ✓ Add(x,prior) – n times
- At the main phase:
  - ✓ GetMin() – n times
  - ✓ ChangePriority(x,new_priority) – $O(m)$ times.

Prim's algorithm

Given a connected graph $G(V, E)$, $|V| = n, |E| = m$.

1. $T(V_T, E_T): V_T = \{s\}, E_T = \emptyset$
2. Array C[1..*n*], P[1..*n*].
   - $C[s] = 0$; P[s]=*s*.
   - For each $v \in V \setminus \mathrm{V}_T: C[v] = w(s, v); P[v] = s$
3. While $V_T \neq V$:
   - Find $v \in V \setminus \mathrm{V}_T$: v has minimum $C[v]$
   - Add v to V_T; add $(P[v], v)$ to E_T
   - Update_C&P(v).

Prim's algorithm

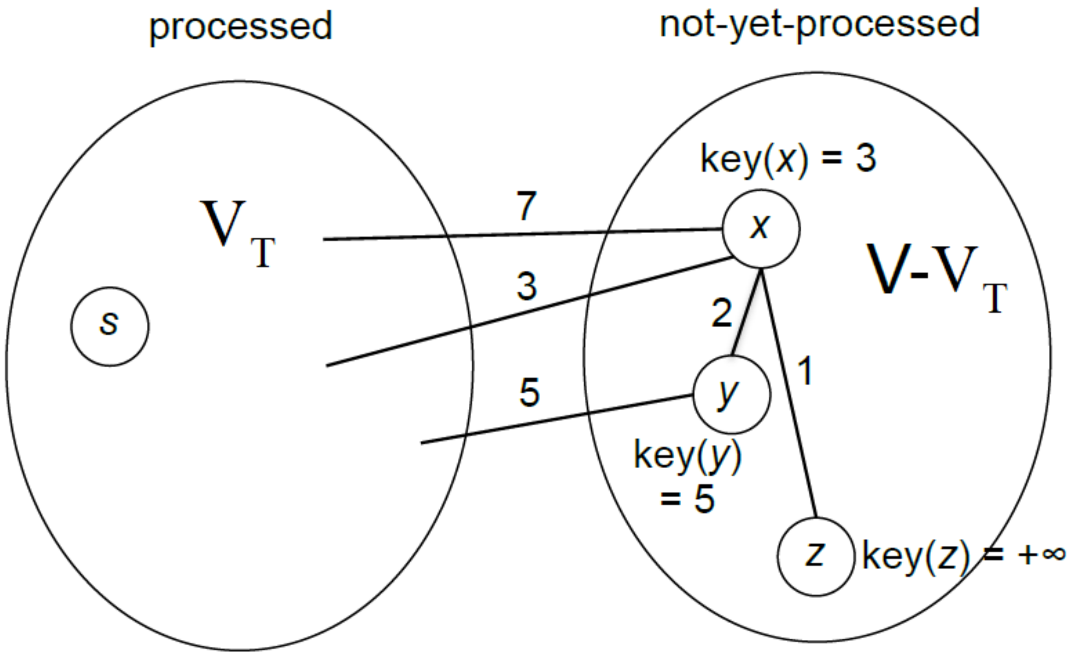
Update_C&P(v)

For each $(v, u) \in E$:

if $u \in V \backslash V_T$ and $C[u] > w(v, u)$:

$C[u] = w(v, u)$

$P[u] = v$

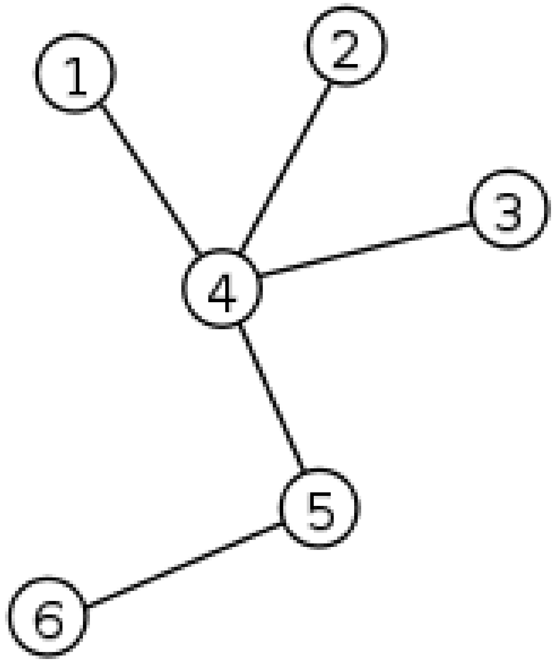


If we use a heap for storing $C[u]$,
the time complexity is $O(m \cdot \log n)$.

Trees

<u>Theorem</u> (properties of trees).

A graph $G(V, E)$ is a tree iff any of the following equivalent conditions hold:



1) G is connected and acyclic (contains no cycles).
2) G is acyclic, and a simple cycle is formed if any edge is added to G.
3) G is connected, but would become disconnected if any single edge is removed from G.
4) Any two vertices in G can be connected by a unique simple path.
5) G is connected and has $n - 1$ edges ($n = |V|$).
6) G has no simple cycles and has $n - 1$ edges.

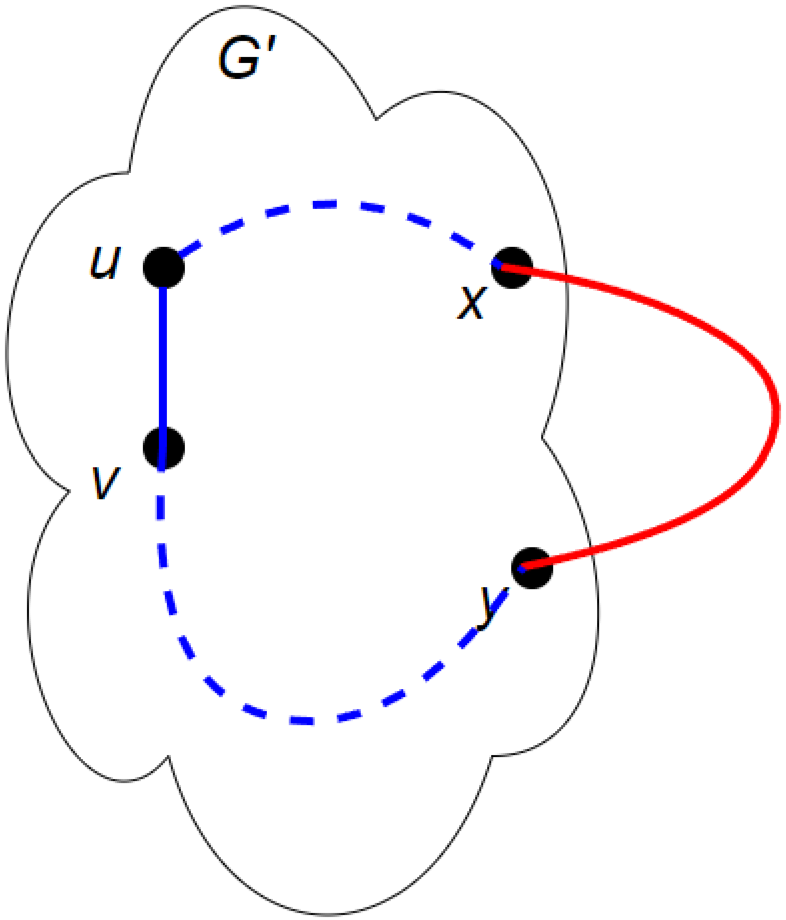
Prim’s algorithm

Definition. Let G' be a spanning tree of G and edge (u, v) belongs to G'. If we delete (u, v) from G', the tree G' is split into 2 components (p.3 from the theorem). Let $\delta(G', (u, v))$ denote the ***cut***, i.e. the set of edges whose endpoints belong to different components of the forest.

Prim’s algorithm

<u>Theorem</u> (Cut Criterion).

A spanning tree $G'(V', E')$ is minimal iff for each tree edge $(u, v) \in E'$ and any non-tree edge $(x, y) \in \delta(G', (u, v))$, the following condition holds: $w(u, v) \leq w(x, y)$.



Prim’s algorithm

Theorem. Prim’s algorithm builds a minimum spanning tree.

Proof.

Let $G'(V, E')$ be the result of Prim’s algorithm. The structure of the algorithm guarantees that G' is a spanning tree. Let us prove that G' is a minimum spanning tree.

Let us demonstrate that G' satisfies the cut criterion of optimality. Let (u, v) be an arbitrary tree edge. Suppose the cut criterion is violated for G'. It means that a non-tree edge $(x, y) \in \delta\left(G', (u, v)\right)$ exists such that $w(u, v) \geq w(x, y)$. But it means that (x, y) should be added to G' instead of (u, v) which makes a contradiction. QED.