Aalto University

Lecture Notes - Week II

Paths and Trees

Fernando Dias, Philine Schiewe and Piyalee Pattanaik

January 29, 2024

CHAPTER 1

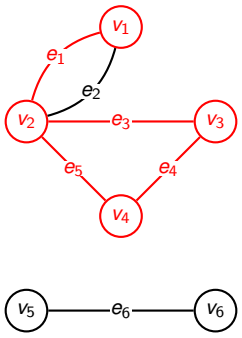
A few more definitions

Recalling the definitions from the previous lecture, we can further improve the definitions of **paths** and **cycles**. For a path P in G from u_1 to u_{k+1} (as an **edge progressio**n):

- Graph $(\{u_1, \ldots, u_{k+1}\}, \{a_1, \ldots, a_k\})$ with $[u_1, a_1, u_2, \ldots, u_k, a_k, u_{k+1}]$ walk and $u_i \neq u_j$, $1 \leq i < j \leq k+1$
- e.g. $[v_1, e_1, v_2, e_3, v_3, e_4, v_4]$

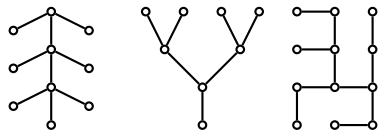
For **cycles** such C in G:

- graph $(\{u_1, \ldots, u_k\}, \{a_1, \ldots, a_k\})$ with $[u_1, a_1, u_2, \ldots, u_k, a_k, u_1]$ (closed) walk, $k \geq 2$ and $u_i \neq u_j$, $1 \leq i < j \leq k$
- e.g. $[v_2, e_3, v_3, e_4, v_4, e_5, v_2]$
- connected if there is a $u - v$ path in G for all $u, v \in V(G)$



1.1 NEWER DEFINITIONS

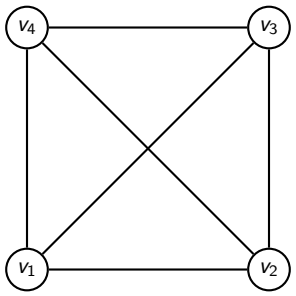
A graph G without a cycle is called *forest*, while a *connected* graph G without a **cycle** is called *tree*.



Now, let $G = (V, E)$ undirected graph with $|V| = n$. Then the following are equivalent:

1. G is a tree, i.e., connected and cycle-free.
2. G is cycle-free and has $n - 1$ edges.
3. G is connected and has $n - 1$ edges.
4. G is minimally connected (removing an edge $\Rightarrow$ not connected anymore).
5. G is maximally cycle-free (adding an edge $\Rightarrow$ cycle).
6. G contains a unique $u - v$ path for any pair of vertices $u, v \in V$.

Let $G = (V, E)$ undirected graph. $T = (V, E')$ with $E' \subseteq E$ is a **spanning tree** of G iff T is a tree. Hence, G is connected if it contains a spanning tree. Let $K_n = (V, E)$ be the complete graph with $|V| = n$ vertices, i.e., for any $u, v \in V$ the edge $\{u, v\} \in E$ exists. Then the number of spanning trees in K_n is n^{n-2}.



CHAPTER 2
Finding Paths

The most useful instance of paths is to **identify the shortest path** in a graph. Finding the **minimum path length** between **two nodes** is trivial, and via **BFS**, it can be easily applied. At the same time, finding the **minimum path length** between **a node and all the others** is also trivial and **BFS** apply to each node individually would suffice.

Challenge: finding the **minimum-cost path** from a node to all the other in a **weighted** graph.

A **weighted graph** is a graph where all the edges have a specific value. It can also named as a **flow network**.

Definition 1 (Flow network) *A tuple* $G = (V, E, f)$ *is said to be a* flow network *if* (V, E) *where for every edge* $(u, v) \in E$ *we have an associated positive integer* flow value f_{uv}.

It also satisfying *conservation of flow* for every $v \in V \setminus \{s, t\}$, where s is an unique source and t is unique sink.

$$\sum_{(u,v)\in E} f_{uv} = \sum_{(v,w)\in E} f_{vw}. \tag{2.1}$$

Therefore, the goal is to calculate the shortest path from a node to each other vertices. Unfortunately, BFS will not suffice (because the shortest path may not have the fewest edges).

Alternative: Dijkstra's algorithm.

Edsger Dijkstra (1930-2002) was a Dutch computer scientist, programmer, software engineer, and science essayist and very influential in Computer Science and Discrete Mathematics. One of this most famous quotes is (which is encapsulated in his most famous algorithm):

"Simplicity is a prerequisite for reliability."

Speaking of algorithm, it is **general ide**a for Dijkstra's approach is as follows:

Figure 2.1: Edsger W. Dijkstra

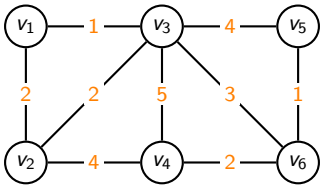
1. Iteratively **increase** the ”set of nodes with known shortest distances”;
2. Any node **outside** this set will have a ”best distance so far”;
3. Update the **”best distance so far”** until add all nodes to set.

The resulting algorithm is:

Algorithm: DIJKSTRA'S ALGORITHM

Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$, nodes V, source s

```
1  d_v distance to reach node v
2  p_v node predecessor to node v
3  Q ← ∅ set of ”unkown distance” nodes.
4  for each node v in V do
5      d_v ← ∞
6      p_v ← FALSE
7      add v in Q
8  d_s ← 0
9  while Q ≠ ∅ do
10     u ← node in Q with min d_u
11     remove u from Q
12     for each neighbor v of u still in Q do
13         d ← d_u + c_uv
14         if alt < d_v then
15             d_v ← alt
16             p_v ← u
17 return d_v,p_v
```



In terms of runtime, this algorithm, when implemented to its best, has a runtime to $O(m + n \cdot \log(n)$, where m is the amount of edges and n is the number of nodes.

Alternatively, there is also an integer linear programming which can be applied (although not recommend):

$$\min \sum_{(u,v)\in E} f_{uv} x_{uv} \tag{2.2a}$$

subject to: (2.2b)

$$\sum_{(s,v)\in E} x_{sv} = 1, \tag{2.2c}$$

$$\sum_{(u,t)\in E} x_{ut} = 1, \tag{2.2d}$$

$$\sum_{(u,v)\in E} x_{uv} - \sum_{(v,w)\in E} x_{vwi} = 0, \tag{2.2e}$$

$$x_{uv} \in \{0,1\}, \qquad \forall (u,v) \in E \tag{2.2f}$$

Constraints (2.2c) and (2.2d) ensures that a path **starts** in the source and **ends** in the sink, while constraint (2.2e) guarantees that **intermediary** nodes have a **single** edge in and a **single** edge out. The objective minimizes the total combined **weight** of the edges in that path.

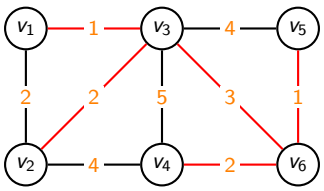
CHAPTER 3

Minimal Spanning Trees

For spanning trees, the goal is to find an algorithm for a minimum spanning tree (MST). First, formally establishing the problem:

Instance: An undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$.

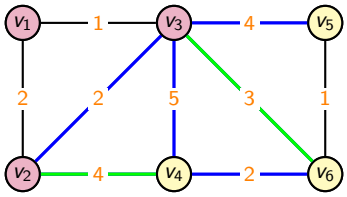
Task: Find a spanning tree T in G of minimum weight.



The **optimality conditions** for such a problem are as follows:

Theorem 1 *Let (G, c) be an instance of the MST problem and T a spanning tree in G. Then the following are equivalent:*

1. *T is optimal.*
2. *For every $e = \{x, y\} \in E(G) \setminus E(T)$, no edge on the $x - y$ path in T has higher cost than e.*
3. *For every $e \in E(T)$, e is a minimum cost edge of $\delta(V(C))$, where C is a connected component of $T - e$.*
4. *We can order $E(T) = \{e_1, \ldots, e_{n-1}\}$ such that for each $i \in \{1, \ldots, n-1\}$ there exists a set $X \subseteq V(G)$ such that e_i is a minimum cost edge of $\delta(X)$ and $e_j \notin \delta(X)$ for all $j \in \{1, \ldots, i-1\}$.*



$\delta(X) = \{\{u, v\} \in E\colon u \in X, v \notin x\}$
edges from X to $V(G) \setminus X$

This problem has been studied to extension, and **two algorithms** have been proposed from the literature. The starting point comes from the following theorem:

Theorem 2 *Let $G = (V, E)$ undirected graph with $|V| = n$. Then the following are equivalent:*

1. *G is a tree, i.e., connected and cycle-free.*
2. *G is cycle-free and has $n - 1$ edges.*
3. *G is connected and has $n - 1$ edges.*
4. *G is minimally connected (removing an edge $\Rightarrow$ not connected anymore).*
5. *G is maximally cycle-free (adding an edge $\Rightarrow$ cycle).*
6. *G contains a unique $u - v$ path for any pair of vertices $u, v \in V$.*

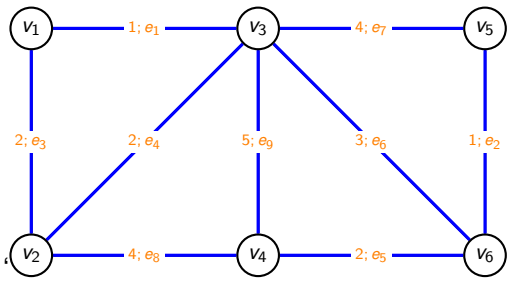
3.1 KRUSKAL'S ALGORITHM

The first option (in no particular order) is **Kruskal's algorithm** (proposed by Joseph Kruskal in 1956).

Algorithm: KRUSKAL'S ALGORITHM
Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$
Output: spanning tree T of minimum weight

```
1 sort edges such that c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m)
2 set T := (V(G), ∅)
3 for i := 1 to m do
4     if T + e_i contains no cycle then
5         set T := T + e_i
6 return T
```

In the following picture, it is shown how each step is calculated:



Execution:
$E(T) = \emptyset$
$E(T) = \{e_1\}$
$E(T) = \{e_1, e_2\}$
$E(T) = \{e_1, e_2, e_3\}$
$E(T) = \{e_1, e_2, e_3, e_5\}$
$E(T) = \{e_1, e_2, e_3, e_5, e_6\}$

Test:
$e_1 = \{v_1, v_3\}$ ✓
$e_2 = \{v_5, v_6\}$ ✓
$e_3 = \{v_1, v_2\}$ ✓
$e_4 = \{v_2, v_3\}$ ✗$\rightsquigarrow$ cycle
$e_5 = \{v_4, v_6\}$ ✓
$e_6 = \{v_3, v_6\}$ ✓
$e_7 = \{v_3, v_5\}$ ✗ $\rightsquigarrow$ cycle
$e_8 = \{v_2, v_4\}$ ✗ $\rightsquigarrow$ cycle
$e_9 = \{v_3, v_5\}$ ✗ $\rightsquigarrow$ cycle

In terms of **correctness**, T is **maximally cycle-free** (no further edge can be added), which is contemplated as a **tree**. For each edge $e_i = \{x, y\} \in E(G) \setminus E(T)$:

- $T + e_i$ contains a **cycle** in line 4;
- there **exists** a $x - y$ path in T at this point;
- all edges in T have **lower** weight than e_i at this point.

Hence, T is **MST**.

In terms of **runtime**:

- sorting edges: $O(m \log m)$
- loop lines 3-5: checking m times for cycles
- checking for cycle containing $e = \{u, v\}$
  - DFS starting from u with at most n edges, check if v is reachable: $O(n)$

$\rightsquigarrow$ total running time: $O(mn)$

To sum up, Kruskal's algorithm is **guaranteed** to be cycle-free and greedily add edges until *maximally cycle-free*.

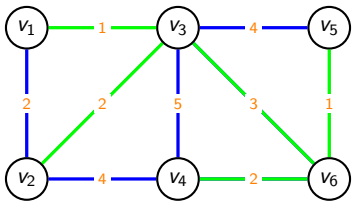
3.2 PRIM'S ALGORITHM

An alternative is Prim's algorithm (developed in 1930 by Czech mathematician **Vojtěch Jarník** and later re-discovered and republished by computer scientists **Robert C. Prim** in 1957 and **Edsger W. Dijkstra** in 1959).

Algorithm: PRIM'S ALGORITHM
Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$
Output: spanning tree T of minimum weight

```
1 choose v ∈ V(G)
2 set T := ({v}, ∅)
3 while V(T) ≠ V(G) do
4 |   choose an edge e ∈ δ_G(V(T)) of minimum weight
5 |   set T := T + e
6 return T
```

Using the following figure as an example:



Execution:
$V(T) = \{v_1\}$
$E(T) = \emptyset$
$V(T) = \{v_1, v_3\}$
$E(T) = \{\{v_1, v_3\}\}$
$V(T) = \{v_1, v_3, v_2\}$
$E(T) = \{\{v_1, v_3\}, \{v_2, v_3\}\}$
$V(T) = \{v_1, v_3, v_2, v_6\}$
$E(T) = \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_6\}\}$
$V(T) = \{v_1, v_3, v_2, v_6, v_5\}$
$E(T) = \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_6\}, \{v_5, v_6\}\}$
$V(T) = \{v_1, v_3, v_2, v_6, v_5, v_4\}$
$E(T) = \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_6\}, \{v_5, v_6\}, \{v_4, v_6\}\}$

Test:
$\delta_G(V(T)) =$
$\{\{v_1, v_2\}, \{v_1, v_3\}\}$
$\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_3, v_6\}\}$
$\{\{v_2, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_3, v_6\}\}$
$\{\{v_2, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_6\}, \{v_5, v_6\}\}$
$\{\{v_2, v_4\}, \{v_3, v_4\}, \{v_4, v_6\}\}$

Regarding **runtime**, the best performance can be achieved as $O(m \log n)$.

Finally, a **ILP formulation** for MST (known as **Martin formulation**):

$$\min \sum_{(u,v)\in E} f_{uv} x_{uv} \tag{3.1a}$$

subject to: (3.1b)

$$\sum_{(u,v)\in E} x_{uv} = n - 1, \tag{3.1c}$$

$$y^k_{uv} + y^k_{vi} = x_{uv}, \qquad (u,v) \in E, k \in V \tag{3.1d}$$

$$\sum_{k\in V\setminus\{(u,v)\}} y^v_{uk} + x_{uv} = 1, \qquad \forall (i,j) \in E \tag{3.1e}$$

$$x_{uv}, y^k_{uv}, y^k_{vu} \in \{0,1\}, \qquad \forall (u,v) \in E, k \in V \tag{3.1f}$$

In the formulation above, y^k_{uv} denotes that edge (u, v) is in the spanning tree and node k is on the side of v.

The **constraint** (3.1d) guarantees that if $(u, v) \in E$ is selected into the tree, any node $k \in V$ must be on **either** side of v (depending if $y^k_{uv} = 1$ or $y^k_{vu} = 1$). If $(u, v) \in E$ is **not** in the tree, any node k **cannot** be on the side of v or u.

The final constraint ensures that if $(u, v) \in E$ is in the tree, edges (u, k) which connects u are on the side of u. If it is **not** in the tree, there **must be and edge** (u, k) such that v is **on the side of** k ($y^v_{uk} = 1$ for some k).