Lecture II - Paths and Trees

[1] Department of Mathematics and Systems Analysis,
Systems Analysis Laboratory, Aalto University, Finland

January 8, 2024

Aalto University

Previously on..

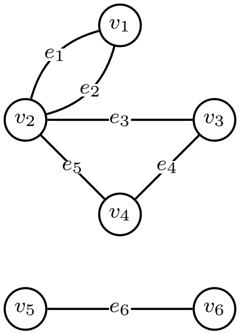
PREVIOUSLY ON...

- Graphs
- Paths, Walks, Trials,
- BFS and DFS.

Useful Definitions

Cycles

- Path P in G from u_1 to u_{k+1}:
  - Graph $(\{u_1, \dots, u_{k+1}\}, \{a_1, \dots, a_k\})$ with $[u_1, a_1, u_2, \dots, u_k, a_k, u_{k+1}]$ walk and $u_i \neq u_j$, $1 \leq i < j \leq k+1$
  - e.g. $[v_1, e_1, v_2, e_3, v_3, e_4, v_4]$
- Cycle C in G:
  - graph $(\{u_1, \dots, u_k\}, \{a_1, \dots, a_k\})$ with $[u_1, a_1, u_2, \dots, u_k, a_k, u_1]$ (closed) walk, $k \geq 2$ and $u_i \neq u_j$, $1 \leq i < j \leq k$
  - e.g. $[v_2, e_3, v_3, e_4, v_4, e_5, v_2]$



Trees and forests

Aalto University

Combinatorial Optimization

Previously on..

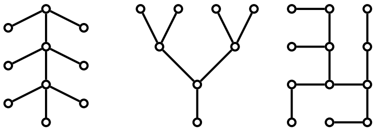
Useful Definitions

Shortest Path

Spanning Tree

Definition

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



Characterization of trees

Theorem

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

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

Spanning trees

Definition

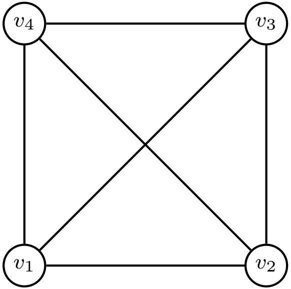
Let $G = (V, E)$ undirected graph. $T = (V, E')$ with $E' \subseteq E$ is a *spanning tree* of G iff T is a tree.

Lemma

G is connected iff it contains a spanning tree.

Theorem

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}.



Shortest Path

Finding Paths

Finding the **minimum path length** between **two nodes** is trivial.

$\rightarrow$ **BFS** can be easily applied;

Finding the **minimum path length** between **a node and all the others** is also trivial.

$\rightarrow$ **BFS** apply to each node individually;

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

Flow Network

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

Definition (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.

$$\sum_{(u,v)\in E} f_{uv} = \sum_{(v,w)\in E} f_{vw}. \tag{1}$$

General Idea

Goal: from a given node, what are the shortest path to each of the other vertices.
Unfortunately, BFS will not suffice.

Shortest path may not have the fewest edges.

Alternative: Dijkstra's algolrithm.

Dijkstra

Edsger Dijkstra (1930-2002)

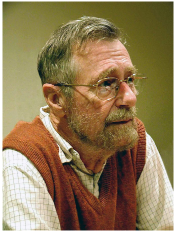


Figure: Edsger W. Dijkstra

"Simplicity is prerequisite for reliability."

General Idea

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.

Dijkstra's Algorithm

Algorithm: DIJKSTRA'S ALGORITHM - Preparation

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

d_v distance to reach node v

p_v node predecessor to node v

$Q \leftarrow \emptyset$ set of "unkown distance" nodes.

for *each node* v *in* V **do**

    $d_v \leftarrow \infty$

    $p_v \leftarrow FALSE$

    add v in Q

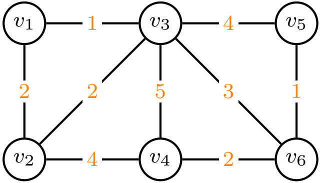
$d_s \leftarrow 0$

Dijkstra's Algorithm

Algorithm: DIJKSTRA'S ALGORITHM - Calculation

Output: d_v,p_v

```
while Q ≠ ∅ do
    u ← node in Q with min d_u
    remove u from Q
    for each neighbor v of u still in Q do
        d ← d_u + c_uv
        if alt < d_v then
            d_v ← alt
            p_v ← u
return d_v,p_v
```


v_1
v_3
v_5
v_2
v_4
v_6
1
4
2
2
5
3
1
4
2

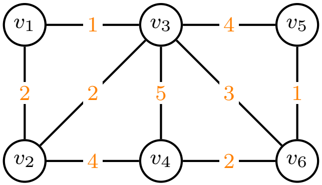
Spanning Tree

Minimal spanning trees

Minimum Spanning Tree 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.



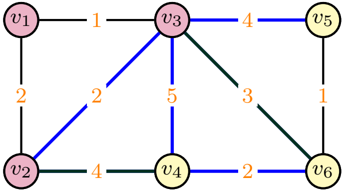
Optimality conditions

Theorem

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

a) *T is optimal.*
b) *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.*
c) *For every $e \in E(T)$, e is a minimum cost edge of $\delta(V(C))$, where C is a connected component of $T - e$.*
d) *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 $j \in \{1, \ldots, i-1\}$.*

Optimality conditions



$$\delta(X) = \{\{u, v\} \in E \colon u \in X, v \notin x\}$$

edges from X to $V(G) \setminus X$

Two algorithms

Theorem

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

a) *G is a tree, i.e., connected and cycle-free.*
d) *G is minimally connected (removing an edge $\Rightarrow$ not connected anymore).*
e) *G is maximally cycle-free (adding an edge $\Rightarrow$ cycle).*

Kruskal

- guaranteed to be cycle-free
- greedily add edges until *maximally cycle-free*

Prim

- grow one connected component
- greedily add edges until *minimally connected*

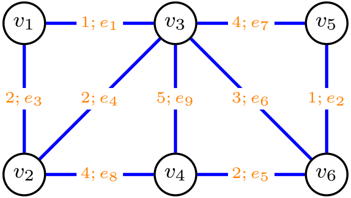
Kruskal's algorithm

Algorithm: KRUSKAL'S ALGORITHM

Input: undirected, connected graph G, weights $c\colon E(G) \to \mathbb{R}$
Output: spanning tree T of minimum weight

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

Kruskal's algorithm



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

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

Kruskal's algorithm – Correctness

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 $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$
2 set $T := (V(G), \emptyset)$
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

- T is maximally cycle-free (no further edge can be added) $\Rightarrow T$ is a tree
- for $e_i = \{x, y\} \in E(G) \setminus E(T)$:
  - $T + e_i$ 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

$\Rightarrow$ T is MST

Kruskal's algorithm – Running time

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
```

- 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: $O(mn)$

Prim's algorithm

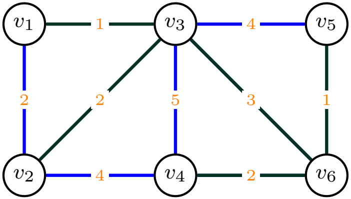
Algorithm: Prim's Algorithm

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

Output: spanning tree T of minimum weight

choose $v \in V(G)$
set $T := (\{v\}, \emptyset)$
while $V(T) \neq V(G)$ **do**
  choose an edge $e \in \delta_G(V(T))$ of minimum weight
  set $T := T + e$
return T

Prim's algorithm



Prim's algorithm

Aalto University

Combinatorial Optimization

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

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

Summary running times MST

Kruskal	
naive implementation	$O(mn)$
most optimal	$O(m \log n)$

Prim	
naive implementation	$O(m + n^2)$
most optimal	$O(m \log n)$

Aalto University