

# Notes for Operations Research & More

Lan Peng, PhD Student

Department of Industrial and Systems Engineering  
University at Buffalo, SUNY  
lanpeng@buffalo.edu

September 8, 2019

September 8, 2019

# Contents

<b>I</b>	<b>Graph and Network Theory</b>	<b>5</b>
<b>1</b>	<b>Basic concepts</b>	<b>7</b>
1.1	Graph . . . . .	7
1.2	Subgraph . . . . .	7
<b>2</b>	<b>Paths, Trees, and Cycles</b>	<b>9</b>
2.1	Walk . . . . .	9
2.2	Path and Cycle . . . . .	9
2.3	Tree and forest . . . . .	9
2.4	Spanning tree . . . . .	10
2.5	Special Graphs . . . . .	10
2.6	Complexity . . . . .	11
<b>3</b>	<b>Shortest-Path Problem</b>	<b>13</b>
<b>4</b>	<b>Minimum Spanning Tree Problem</b>	<b>15</b>
<b>5</b>	<b>Maximum Flow Problem</b>	<b>17</b>
<b>6</b>	<b>Minimum Cost Flow Problem</b>	<b>19</b>
<b>7</b>	<b>Assignment and Matching Problem</b>	<b>21</b>
<b>8</b>	<b>Graph Algorithms</b>	<b>23</b>
<b>9</b>	<b>Polygon Triangulation</b>	<b>25</b>
9.1	Types of Polygons . . . . .	25
9.2	Triangulation . . . . .	25
9.3	Art Gallery Theorem . . . . .	25
9.4	Triangulation Algorithms . . . . .	25
9.5	Shortest Path . . . . .	25



**Part I**

**Graph and Network Theory**



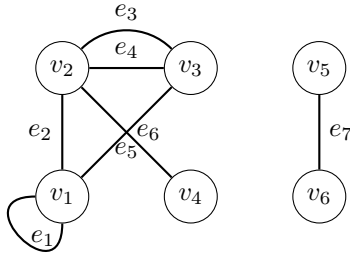
# Chapter 1

## Basic concepts

### 1.1 Graph

**Definition 1.1.1** (Graph). A **graph**  $G$  consists of a finite set  $V(G)$  on vertices, a finite set  $E(G)$  on edges and an **incident relation** than associates with any edge  $e \in E(G)$  an unordered pair of vertices not necessarily distinct called **ends**.

For example, the following graph



can be represented as

$$V = V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\} \quad (1.1)$$

$$E = E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \quad (1.2)$$

$$e_1 = v_1v_2, e_2 = v_2v_4, \dots \quad (1.3)$$

**Definition 1.1.2** (loop, parallel, simple graph). An edge with identical ends is called a **loop**. Two edges having the same ends are said to be **parallel**. A graph without loops or parallel edges is called **simple graph**.

**Definition 1.1.3** (adjacent). Two edges of a graph are **adjacent** if they have a common end, two vertices are **adjacent** if they are jointed by an edge.

### 1.2 Subgraph

**Definition 1.2.1** (subgraph). Given two graphs  $G$  and  $H$ ,  $H$  is a **subgraph** of  $G$  if  $V(H) \subseteq V(G)$ ,  $E(H) \subseteq E(G)$  and an edge has the same ends in  $H$  as it does in  $G$ . Furthermore, if  $E(H) \neq E(G)$  then  $H$  is a proper subgraph.

**Definition 1.2.2** (spanning). A subgraph  $H$  on  $G$  is **spanning** if  $V(H) = V(G)$ .

**Definition 1.2.3** (vertex-induced, edge-induced). For a subset  $V' \subseteq V(G)$  we define an **vertex-induced** subgraph  $G[V']$  to be the subgraph with vertices  $V'$  and those edges of  $G$  having both ends in  $V'$ . The **edge-induced** subgraph  $G[E']$  has edges  $E'$  and those vertices of  $G$  that are ends to edges in  $E'$ .

**Notice:** If we combine node-induced or edge-induced subgraphs  $G(V')$  and  $G(V - V')$ , we cannot always get the entire graph.

**Definition 1.2.4** (degree). Let  $v \in V(G)$ , then the **degree** of  $v \in V(G)$  denote by  $d_G(v)$  is defines to be the number of edges incident of  $v$ . Loops counted twice.

**Problem 1.1.** If  $G$  is a simple graph with at least two vertices, prove that  $G$  has two vertices with the same degree.

*Proof.* □

**Problem 1.2.** Explain clearly, what is the largest possible number of vertices in a graph with 19 edges and all vertices of degree at least 3. Explain why this is the maximum value.

**Theorem 1.1.** For any graph  $G = (V, E)$

$$\sum_{v \in V} d(v) = 2|E| \quad (1.4)$$

*Proof.*  $\forall$  edge  $e = \mu v$  with  $\mu \neq v$ ,  $e$  is and counted once for  $\mu$  and once for  $v$ , a total of two altogether. If  $e = \mu\mu$ , a loop, then it is counted twice for  $\mu$  □

**Corollary 1.1.1.** Every graph has an even number of odd degree vertices.

*Proof.*

$$V = V_E \cup V_O \Rightarrow \sum_{v \in V} d(v) = \sum_{v \in V_E} d(v) + \sum_{v \in V_O} d(v) = 2|E| \quad (1.5)$$

□





# Chapter 2

## Paths, Trees, and Cycles

### 2.1 Walk

**Definition 2.1.1** (walk). A **walk** in a graph  $G$  is a finite sequence  $w = v_0 e_1 v_1 e_2 \dots e_k v_k$ , where for each  $e_i = v_{i-1} v_i$  the edge and its ends exists in  $G$ . We say that walk  $w$  to  $v_k$  on  $(v_0, v_k)$ -walk.

**Example.**

$$w = v_2 e_4 v_3 e_4 v_2 e_5 v_3 \quad (2.1)$$

is a walk, or  $(v_2, v_3)$ -walk

**Definition 2.1.2** (origin, terminal, internal, length). For  $(v_0, v_k)$ -walk, The vertices  $v_0$  and  $v_k$  are called the **origin** and the **terminal** of the walk  $w$ ,  $v_1 \dots v_{k-1}$  are called **internal** vertices. The integer  $k$  is the **length** of the walk. Length of  $w$  equals to the number of edges.

We can create a reverse walk  $w^{-1}$  by reversing  $w$ .

$$w^{-1} = v_k e_k v_{k-1} e_{k-1} \dots e_2 v_1 \quad (2.2)$$

(The reverse walk is guaranteed to exist because it is an undirected graph)

Given two walks  $w$  and  $w'$  we can create a third walk denoted by  $ww'$  by concating  $w$  and  $w'$ . The new walk's origin is the same as terminal.

### 2.2 Path and Cycle

**Definition 2.2.1** (trail). A **trail** is a walk with no repeating edges. e.g.,  $v_3 e_4 v_2 e_5 v_3$

**Definition 2.2.2** (path). A **path** is a trail with no repeating vertices. e.g.,  $v_3 e_4 v_2$

**Notice:** Paths  $\subseteq$  Trails  $\subseteq$  Walks

**Definition 2.2.3** (closed, cycle). A path is **closed** if it has positive length and its origin and terminal are the same. e.g.,  $v_1 e_2 v_2 e_4 v_3 e_3 v_1$ . A closed trail where origin and internal vertices are distinct is called a **cycle** (The only time a vertex is repeated is the origin and terminal)

**Definition 2.2.4** (even/odd cycle). A cycle is **even** if it has a even number of edges otherwise it is **odd**.

**Problem 2.1.** Prove that if  $C_1$  and  $C_2$  are cycles of a graph, then there exists cycles  $K_1, K_2, \dots, K_m$  such that  $E(C_1) \Delta E(C_2) = E(K_1) \cup E(K_2) \cup \dots \cup E(K_m)$  and  $E(K_i) \cap E(K_j) = \emptyset$ . (For set  $X$  and  $Y$ ,  $X \Delta Y = (X - Y) \cup (Y - X)$ , and is called the symmetric difference of  $X$  and  $Y$ )

**Definition 2.2.5** (connected vertices). Two vertices  $u$  and  $v$  in a graph are said to be **connected** if there is a path between  $u$  and  $v$ .

**Definition 2.2.6** (component). Clearly connectivity between vertices is an equivalence relation on  $V(G)$ , if  $V_1, \dots, V_k$  are the corresponding equivalent classes then  $G[V_1] \dots G[V_k]$  are **components** of  $G$ . If graph has only one component, then we say the graph is connected. A graph is connected iff every pair of vertices in  $G$  are connected, i.e., there exists a path between every pair of vertices.

### 2.3 Tree and forest

**Definition 2.3.1** (acyclic graph). A graph is called **acyclic** if it has no cycles

**Definition 2.3.2** (forest, tree). A acyclic graph is called a **forest**. A connected forest is called a **tree**.

**Problem 2.2.** Prove that if  $T$  is a tree, then  $T$  has exactly one more vertex than it has edges.

*Proof.* For any tree  $T$ , if we remove one edge in the tree, the graph after removal of the edge will have two components, one of the following situations will happen:

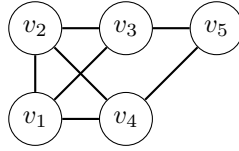
- Both components are single vertex.
- One of components is a single vertex, the other is a tree.
- Both components are tree.

□

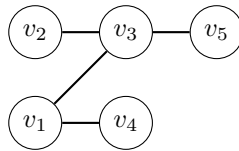
## 2.4 Spanning tree

**Definition 2.4.1** (spanning tree). A subgraph  $T$  of  $G$  is a **spanning tree** if it is spanning ( $V(T) = V(G)$ ) and it is a tree.

**Example.** In the following graph



This is a spanning tree



**Problem 2.3.** Prove that if  $T_1$  and  $T_2$  are spanning trees of  $G$  and  $e \in E(T_1)$ , then there exists a  $f \in E(T_2)$ , such that  $T_1 - e + f$  and  $T_2 + e - f$  are both spanning trees of  $G$ .

**Theorem 2.1.** Every connected graph has a spanning tree.

*Proof.* Prove by constructing algorithm:  $\square$

---

**Algorithm 1** Find a spanning tree for connected graph

---

**Require:** a connected graph  $G$  and an enumeration  $e_1, \dots, e_m$  of the edges of  $G$

**Ensure:** a spanning tree  $T$  of  $G$

- 1: Let  $T$  be the spanning subgraph of  $G$  with  $V(T) = V(G)$  and  $E(T) = \emptyset$
  - 2:  $i \leftarrow 1$
  - 3: **while**  $i \leq |E|$  **do**
  - 4:   **if**  $T + e_i$  is acyclic **then**
  - 5:      $T \leftarrow T + e_i$
  - 6:      $i \leftarrow i + 1$
  - 7:   **end if**
  - 8: **end while**
- 

**Notice:** This algorithm can be improved, one idea is to make summation of edges in spanning subgraph less or equation to  $|V| - 1$

To prove algorithm we need to show the output is a spanning tree, which means three properties must hold:

- spanning (Step 1)
- acyclic (We never add an edge that create a cycle)

- connected (Proof by contradiction)

So it is sufficient to show that the output will be connected.

*Proof.* (Proof by Contradiction) Suppose the output graph  $T$  of the algorithm is NOT connected. Let  $T_1$  be a component of  $T$ , let  $x \in T_1$  and  $y \notin T_1$ . But  $G$  is a connected graph (given from the beginning), so there must be a path in  $G$  that connects  $x$  and  $y$ . Let such a path in  $G$  be  $p = xe_1v_1e_2 \dots v_{k-1}e_ky$ . Clearly,  $p \notin T_1$ . So there must be a first vertex in  $P$  that not in  $T_1$ . So  $e_i \notin E(T)$ , the only way this can happen when applying the algorithm is if  $T + e_i$  creates a cycle  $C$ , i.e.,  $e_i \in C$ , so  $C - e_i$  is a path connecting  $v_{i-1}$  and  $v_i$ . So  $C - e_i \in T$ , so  $v_{i-1}$  is connected to  $v_i \in T$ . Contradiction.  $\square$

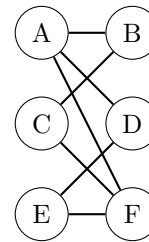
## 2.5 Special Graphs

**Definition 2.5.1.** A **complete** graph  $K_n$  ( $n \geq 1$ ) is a simple graph with  $n$  vertices and with exactly one edge between each pair of distinct vertices.

**Definition 2.5.2.** A **cycle** graph  $C_n$  ( $n \geq 3$ ) consists of  $n$  vertices  $v_1, \dots, v_n$  and  $n$  edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}$

**Definition 2.5.3.** A **wheel** graph  $W_n$  ( $n \geq 3$ ) is a simple graph obtains by adding one vertex to the cycle graph  $C_n$ , and connecting this new vertex to all vertices of  $C_n$

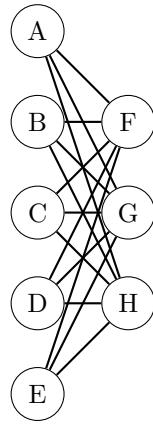
**Definition 2.5.4.** A simple graph is said to be **bipartite** if the vertex set can be expressed as the union of two disjoint non-empty subsets  $V_1$  and  $V_2$  such that every edges has one end in  $V_1$  and another end in  $V_2$



**Definition 2.5.5** (complete bipartite). The **complete bipartite** graph  $K_{mn}$  is the bipartite graph  $V_1$  containing  $m$  vertices and  $V_2$  containing  $n$  vertices such that each vertex in  $V_1$  is adjacent to every vertex in  $V_2$

**Example.** Here is an example for  $K_{53}$

**Problem 2.4.** Prove that a graph  $G$  is bipartite iff every cycle is even.



## 2.6 Complexity

This part is going to be moved to Algorithm notes (or I will just delete this part because of duplication). We want to know guaranteed performances - "worse case" scenarios - for any algorithm working on any problem instance. FIXME

The following example is for addition of two matrices:

---

**Algorithm 2** Add two  $m \times n$  matrices  $A, B$  to get matrix  $C$

---

```

for  $i = 1, 2, \dots, m$  do
  for  $j = 1, 2, \dots, n$  do
     $C_{ij} = A_{ij} + B_{ij}$ 
  end for
end for

```

---

The "running time" of an algorithm is measured by the number of basic operational steps.

For so called "basic" steps, it includes

- $+, -, \times, \div$
- assignments and storage of a variable
- comparisons

For the example above

- $c_1 mn$  for addition  $C_{ij} = A_{ij} + B_{ij}$
- $c_2 mn$  for saving  $C_{ij}$
- $c_3 mn$  for comparison and assignment for  $i$  and  $j$

$c_1, c_2, c_3$  does not matter, the number of steps are  $m \times n$ , we say the algorithm runs  $O(mn)$  (big O notation, the worse case)



## Chapter 3

# Shortest-Path Problem



## Chapter 4

# Minimum Spanning Tree Problem





## Chapter 5

# Maximum Flow Problem



## Chapter 6

# Minimum Cost Flow Problem



## Chapter 7

# Assignment and Matching Problem



## Chapter 8

# Graph Algorithms



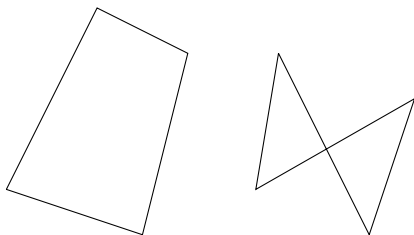


# Chapter 9

## Polygon Triangulation

### 9.1 Types of Polygons

**Definition 9.1.1** (simple polygon). A **simple polygon** is a closed polygonal curve without self-intersection.

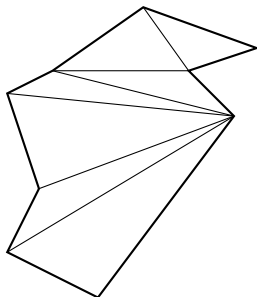


Simple Polygon      Non-simple Polygon

Polygons are basic building blocks in most geometric applications. It can model arbitrarily complex shapes, and apply simple algorithms and algebraic representation/manipulation.

### 9.2 Triangulation

**Definition 9.2.1** (Triangulation). **Triangulation** is to partition polygon  $P$  into non-overlapping triangles using diagonals only. It reduces complex shapes to collection of simpler shapes. Every simple  $n$ -gon admits a triangulation which has  $n - 2$  triangles.



Triangulation

**Theorem 9.1.** *Every polygon has a triangulation*

**theorem 9.2.** *Every polygon with more than three vertices has a diagonal.*

*Proof.* (by Meisters, 1975) Let  $P$  be a polygon with more than three vertices. Every vertex of a  $P$  is either *convex* or *concave*. W.L.O.G. (any polygon must have convex corner) Assume  $p$  is a convex vertex. Denote the neighbors of  $p$  as  $q$  and  $r$ . If  $\bar{q}r$  is a diagonal, done, and we call  $\triangle pqr$  is an *ear*. If  $\triangle pqr$  is not an ear, it means at least one vertex is inside  $\triangle pqr$ , assume among those vertices inside  $\triangle pqr$ ,  $s$  is a vertex closest to  $p$ , then  $\bar{p}s$  is a diagonal.  $\square$

### 9.3 Art Gallery Theorem

**Theorem 9.3.** *Every  $n$ -gon can be guarded with  $\lfloor \frac{n}{3} \rfloor$  vertex guards*

**theorem 9.4.** *Triangulation graph can be 3-colored.*

**Problem 9.1.** The floor plan of an art gallery modeled as a simple polygon with  $n$  vertices, there are guards which is stationed at fixed positions with 360 degree vision but cannot see through the walls. How many guards does the art gallery need for the security? (Fun fact: This problem was posted to Vasek Chvatal by Victor Klee in 1973).

*Proof.* -  $P$  plus triangulation is a planar graph  
- 3-coloring means there exist a 3-partition for vertices that no edge or diagonal has both endpoints within the same set of vertices.

- Proof by Induction:

- Remove an ear (there will always exist ear)
- Inductively 3-color the rest
- Put ear back, coloring new vertex with the label not used by the boundary diagonal.  $\square$

### 9.4 Triangulation Algorithms

### 9.5 Shortest Path