Graph Thoery – Basic Concept

Zhang Mingyuan

Nanyang Technological University

2023.3.17

1. Introduction
   - Overall Structure
   - Training Plan
   - Graph Theory
2. Basic Concept
3. Special Graphs

Overview

- Basic Algorithm

Overview

- Basic Algorithm
- Graph Theory

Overview

- Basic Algorithm
- Graph Theory
- Data Structure

Overview

- Basic Algorithm
- Graph Theory
- Data Structure
- Dynamic Programming

Overview

- Basic Algorithm
- Graph Theory
- Data Structure
- Dynamic Programming
- Stringology

Overview

- Basic Algorithm
- Graph Theory
- Data Structure
- Dynamic Programming
- Stringology
- Mathematics

Training Plan

- 3 Lectures + 3 Individual contests on Saturday (Week 9, 11, 13)

Training Plan

- 3 Lectures + 3 Individual contests on Saturday (Week 9, 11, 13)
- Optional: Hard contests (from World Final and PTZ camps) and seminars on Sunday (from next week)

Training Plan

- 3 Lectures + 3 Individual contests on Saturday (Week 9, 11, 13)
- Optional: Hard contests (from World Final and PTZ camps) and seminars on Sunday (from next week)
- Optional: Individual practice, Codeforces & AtCoder

Training Plan

- 3 Lectures + 3 Individual contests on Saturday (Week 9, 11, 13)
- Optional: Hard contests (from World Final and PTZ camps) and seminars on Sunday (from next week)
- Optional: Individual practice, Codeforces & AtCoder
- Optional: Unofficial team contests

Overview

- Basic Concept

Overview

- Basic Concept
- Connectivity

Overview

- Basic Concept
- Connectivity
- Path and Ring

Overview

- Basic Concept
- Connectivity
- Path and Ring
- Tree

Overview

- Basic Concept
- Connectivity
- Path and Ring
- Tree
- Flow

Overview

- Basic Concept
- Connectivity
- Path and Ring
- Tree
- Flow
- Match, Cover, Independent

Overview

- Basic Concept
- Connectivity
- Path and Ring
- Tree
- Flow
- Match, Cover, Independent
- Graph Counting

Overview

- Basic Concept
- Connectivity
- Path and Ring
- Tree
- Flow
- Match, Cover, Independent
- Graph Counting
- Other topics

Graph, Node, Edge

- A **Graph** is denoted as $G = (V(G), E(G))$.

Graph, Node, Edge

- A **Graph** is denoted as $G = (V(G), E(G))$.
- $V(G)$: **vertex set**, $E(G)$: **edge set**.

Graph, Node, Edge

- A **Graph** is denoted as $G = (V(G), E(G))$.
- $V(G)$: **vertex set**, $E(G)$: **edge set**.
- G is called a **finite graph** when both V and E are finite sets. Otherwise it's called **infinite graph**.

Graph, Node, Edge

- **Undirected Graph**: each (u, v) in E is an unordered pair, representing an **undirected edge**.

Graph, Node, Edge

- **Undirected Graph**: each (u, v) in E is an unordered pair, representing an **undirected edge**.
- **Directed graph**: each (u, v) or $u \rightarrow v$ in E is an ordered pair, representing a **directed edge** (or called an **arc**).

Graph, Node, Edge

- **Undirected Graph**: each (u, v) in E is an unordered pair, representing an **undirected edge**.
- **Directed graph**: each (u, v) or $u \to v$ in E is an ordered pair, representing a **directed edge** (or called an **arc**).
- **Weighted Graph**: edges are specified with a weight.

Graph, Node, Edge

- **Undirected Graph**: each (u, v) in E is an unordered pair, representing an **undirected edge**.
- **Directed graph**: each (u, v) or $u \to v$ in E is an ordered pair, representing a **directed edge** (or called an **arc**).
- **Weighted Graph**: edges are specified with a weight.
- **Order of a Graph** $|V(G)|$: Number of vertices.

Simple Graph

- **Loop**: $e = (u, u)$

Simple Graph

- **Loop**: $e = (u, u)$
- **Multiple Edge**: $e_1 = e_2 = (u, v)$

Simple Graph

- **Loop**: $e = (u, u)$
- **Multiple Edge**: $e_1 = e_2 = (u, v)$
- **Simple Graph**: a graph without any loop or Multiple edge

Simple Graph

- **Loop**: $e = (u, u)$
- **Multiple Edge**: $e_1 = e_2 = (u, v)$
- **Simple Graph**: a graph without any loop or Multiple edge
- **Multigraph**: a graph with at least one loop or one group of multiple edges.

Adjacent and Degree

- Given $e = (u, v)$, u and e, v and e are **incident** or **adjacent**. u and v are **adjacent**.

Adjacent and Degree

- Given $e = (u, v)$, u and e, v and e are **incident** or **adjacent**. u and v are **adjacent**.
- **neighborhood** $N(u)$: contains all vertices that are adjacent to u.

Adjacent and Degree

- Given $e = (u, v)$, u and e, v and e are **incident** or **adjacent**. u and v are **adjacent**.
- **neighborhood** $N(u)$: contains all vertices that are adjacent to u.
- $N(S) = \bigcup_{v \in S} N(v)$.

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.
- There are **even** numbers of vertices whose degree is **odd** in an undirected graph.

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.
- There are **even** numbers of vertices whose degree is **odd** in an undirected graph.
- **isolated vertex**: $d(v) = 0$

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.
- There are **even** numbers of vertices whose degree is **odd** in an undirected graph.
- **isolated vertex**: $d(v) = 0$
- **leaf vertex**: $d(v) = 1$

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.
- There are **even** numbers of vertices whose degree is **odd** in an undirected graph.
- **isolated vertex**: $d(v) = 0$
- **leaf vertex**: $d(v) = 1$
- **even vertex**: $2 \mid d(v)$

Degree

- Degree of vertice $d(v)$: number of edges adjacent to v. (if $e = (v, v)$, $d(v)$ increases by 2)
- **Handshaking lemma**: For any undirected graph, $\sum_{v \in V} d(v) = 2|E|$.
- There are **even** numbers of vertices whose degree is **odd** in an undirected graph.
- **isolated vertex**: $d(v) = 0$
- **leaf vertex**: $d(v) = 1$
- **even vertex**: $2 \mid d(v)$
- **odd vertex**: $2 \nmid d(v)$

Degree Sequence

- Given a degree sequence $d = d_1, d_2, \ldots, d_n$, can we construct the corresponding graph ?

Degree Sequence

- Given a degree sequence $d = d_1, d_2, \ldots, d_n$, can we construct the corresponding graph ?
- Arbitrary Graph (may contain loop or multiple edges): $d_1 + d_2 + \cdots + d_n \equiv 0 \pmod 2$

Degree Sequence

- Given a degree sequence $d = d_1, d_2, \ldots, d_n$, can we construct the corresponding graph ?
- Arbitrary Graph (may contain loop or multiple edges): $d_1 + d_2 + \cdots + d_n \equiv 0 \pmod 2$
- Simple Graph(Havel Theory): suppose $d_1 \geq d_2 \geq \cdots \geq d_n$, then d can be converted to a simple graph iff $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n$ can be converted to a simple graph

Path

- **Walk**: a vertex sequence $u_1, u_2, u_3, \ldots, u_k$, $(u_i, u_{i+1}) \in E$.

Path

- **Walk**: a vertex sequence $u_1, u_2, u_3, \ldots, u_k$, $(u_i, u_{i+1}) \in E$.
- **Path** or **Simple Path**: a walk that $u_i \neq u_j$ if $i \neq j$

Path

- **Walk**: a vertex sequence $u_1, u_2, u_3, \ldots, u_k$, $(u_i, u_{i+1}) \in E$.
- **Path** or **Simple Path**: a walk that $u_i \neq u_j$ if $i \neq j$
- **Circuit**: $u_0 = u_k$

Path

- **Walk**: a vertex sequence $u_1, u_2, u_3, \ldots, u_k$, $(u_i, u_{i+1}) \in E$.
- **Path** or **Simple Path**: a walk that $u_i \neq u_j$ if $i \neq j$
- **Circuit**: $u_0 = u_k$
- **Circle**: $u_0 = u_k$ is the only repeat node.

Subgraph

- **Subgraph**: Given two graphs $G = (V, E), H = (V', E')$, if $V' \subseteq V, E' \subseteq E$, then H is a subgraph of G or $H \subseteq G$.

Subgraph

- **Subgraph**: Given two graphs $G = (V, E), H = (V', E')$, if $V' \subseteq V, E' \subseteq E$, then H is a subgraph of G or $H \subseteq G$.
- if $H \subseteq G$, $\forall u, v \in V'$, if $(u, v) \in E$ then $(u, v) \in E'$. H is an **induced subgraph** of G.

Subgraph

- **Subgraph**: Given two graphs $G = (V, E), H = (V', E')$, if $V' \subseteq V, E' \subseteq E$, then H is a subgraph of G or $H \subseteq G$.
- if $H \subseteq G$, $\forall u, v \in V'$, if $(u, v) \in E$ then $(u, v) \in E'$. H is an **induced subgraph** of G.
- if $H \subseteq G$ and $V' = V$, then H is a **spanning subgraph** of G.

Special Graphs

- **Complete Graph**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**
- **Chain**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**
- **Chain**
- **Tree, Forest**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**
- **Chain**
- **Tree, Forest**
- **JellyFish, Pseudotree, Pseudoforest**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**
- **Chain**
- **Tree, Forest**
- **JellyFish, Pseudotree, Pseudoforest**
- **Cactus, Dessert**

Special Graphs

- **Complete Graph**
- **Edgeless Graph**
- **Tournament Graph**
- **Cycle Graph**
- **Star Graph**
- **Chain**
- **Tree, Forest**
- **JellyFish, Pseudotree, Pseudoforest**
- **Cactus, Dessert**
- **Bipartite Graph**

Usage

- Determine whether the given graph is a special graph

Usage

- Determine whether the given graph is a special graph
- Use its special properties to solve the problem

Usage

- Determine whether the given graph is a special graph
- Use its special properties to solve the problem
- Constructive problems

Thanks!