Lecture Notes for MAT4004

Suggested Citation: (2018), "Lecture Notes for MAT4004", : Vol. xx, No. xx, pp 1–18.
DOI: 10.1561/XXXXXXXXX.

now
the essence of knowledge
Boston — Delft

Contents

Lecture Notes for MAT4004

ABSTRACT

This document is the typed lecture notes for MAT4004

(2018), "Lecture Notes for MAT4004", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

1

Definition

Definition 1.1 (Graph)**.** A graph $G = (V, E)$ consists of a non-empty finite set V of elements called *vertices*; and a finite family of *unordered* pairs of vertices called *edges*.

1. The edge $e = \{v, w\}$ is said to join the vertices v and w

2. The vertex v is *adjacent* to the vertex w if there exists an edge $\{v, w\}$

3. The edge e is also said to be *incident* to v.

Definition 1.2 (Simple)**.** Multiple (parallel) edges are the edges joining the same pair of vertices; Loops are the edges of the form $\{v, v\}$; A *loopless* graph with *no* multiple edges is called a *simple* graph.

Definition 1.3 (Isomorphism)**.** Two graphs G_1 and G_2 are *isomorphic* if there is a one-to-one correspondence between the vertices of G_1 and the vertices of G_2 such that the number of edges joining any two vertices of G_1 equals the number of edges joining the corresponding pair of vertices in G_2.

Definition 1.4 (Adjacent & Incident)**.** Two vertices are *adjacent* if there is an edge joining them; the edges are *incident* to the edge; two edges are *adjacent* if they share a common vertex.

Definition 1.5. The *degree* of a vertex v, say $\deg(v)$, is the number of edges incident to v.

In particular, a loop contributes 2 to the $\deg(v)$; a vertex with degree 0 is an *isolated* vertex; a vertex of degree 1 is an *end-vertex*.

Definition 1.6 (Degree sequence)**.** The *degree sequence* of a graph is the sequence of degrees of its vertices, written in *non-decreasing* order.

Theorem 1.1. In any graph, the sum of all the vertex-degrees is an *even* number.

Corollary 1.2. In any graph, the number of vertices with *odd* degree is *even*.

Definition 1.7. A graph H is a *subgraph* of a graph $G = (V, E)$, if each of its vertices belong to $V(G)$, and each of its edges belongs to $E(G)$.

Definition 1.8. The subgraphs can be obtained by the following operations:

1. $G - e$: removing edge e
2. $G - F$: removing the set of edges F
3. $G - v$: removing vertex v and all its incident edges
4. $G - S$: removing the vertices in the set S and all edges incident to any vertex in S
5. $G \setminus e$: *contracting* edge e. (question: identifying endpoints of e and delete the possible parallel edges)

Definition 1.9 (Complement)**.** If G is a simple graph, its complement, denoted as $\bar{G}$, has the same vertex set, while two vertices are adjacent in $\bar{G}$ if and only if thery are not adjacent in G.

Definition 1.10 (Null & Complete)**.** A *null* graph is the one where the edge set is empty; a simple graph where each distinct pair of vertices are adjacent is a *complete* graph; a complete graph on n vertices is denoted as K_n.

Definition 1.11 (Walk & Path & Cycle)**.**

1. A walk consists of a sequence of edges, one following after another
2. A walk in which no vertex appears *more than once* is called a path
3. A walk in which no vertex appears *more than once*, except for begining and end vertices which coincide, is called a *cycle*.

Definition 1.12 (Bipartite)**.** If the vertex set of a graph consists of the union of two *disjoint* sets A and B such that each edge of G joints a vertex in A and a vertex in B, then G is a *bipartite* graph. Moreover, if each vertex in A is joined to each vertex in B by an edge, it is a *complete bipartite graph*, denoted as $K_{|A|,|B|}$.

Definition 1.13 (Directed Graph)**.** The graph $G = (N, A)$ is a *directed graph* (or *digraph*) if V is a finite set of nodes and A is a finite family of *directed edges* (*arcs*). Each arc in A, denoted as (v, w), is an ordered pair of nodes.

The diagraph D is simple if the arcs are all distinct, and there are no loops.

Definition 1.14 (Underlying Graph)**.** The underlying graph $G = (N, E)$ of a directed graph $G = (N, A)$ has the same node set; every edge $e = \{v, w\}$ of E corresponds to an arc $(v, w) \in A$.

Question: does underlying graph mean the *undirected* graph?

Definition 1.15 (Out & In-degree)**.** Recall that the edge (v, w) is *incident* from v and *incident* to w. The *out-degree* of vertex v is the number of edges *incident from* v; the *in-degree* of vertex v is the number of edges *incident to* v.

Theorem 1.3. In any digraph, the sum of all the *in-degrees* is equal to the sum of all the *out-degrees*.

Proof. Question Each edge (v, w) contribute 1 to the in-degree and 1 to the out-degree. □

Definition 1.16 (Adjacency matrix)**.** The *adjacency matrix* for an *undirected graph* $G = (V, E)$ is a $|V| \times |V|$ square matrix where the element (i, j) is the number of edges joining the vertices i and j.

Definition 1.17 (Incidence matrix)**.** The *incidence matrix* for an *undirected graph* $G = (V, E)$ is a $|V| \times |E|$ matrix, where the element (i, j) is 1 if the vertex i is incident to edge j.

Definition 1.18 (Node-Arc Incidence matrix)**.** The *node-arc incidence matrix* for a directed graph $G = (V, E)$ is a $|V| \times |E|$ matrix where the element (i, j) is 1 if edge j is incident from vertex i, and is -1 if edge j is incident to i.

2

Connectivity

2.1 Bipartite Graphs

Definition 2.1 (Cycle)**.** A *cycle* is a sequence of adjacent edges

$$(v_0, v_1), \quad (v_1, v_2), \quad \ldots, \quad (v_{m-2}, v_{m-1}), \quad (v_{m-1}, v_m = v_0)$$

where all the vertices $v_0, v_1, \ldots, v_{m-1}$ are *distinct*. The number of deges in the cycle is called its *length*.

Definition 2.2 (Bipartite)**.** A graph $G = (V, E)$ is *bipartite* if its vertex-set is a union of two disjoint sets, say A and B, such that every edge in E joins a vertex in A to one in B.

Theorem 2.1. A graph G is *bipartite* if and only if every cycle of G has even length.

Proof. Necessity. Take any cycle and "trace" its edges, the cylce must have even length.

Sufficiency. w.l.o.g., G is connected. Construct A as the vertices s.t. the shortest path to v has even length. Then B is the set of vertices that are not in A.

We claim that (A, B) forms a bipartite graph, since otherwise we can construct a cycle with odd length. □

2.2 Bounds on the number of edges

Definition 2.3 (Connected)**.** A graph is *connected* if there is a path from every vertex to any other vertex

Definition 2.4 (Simple)**.** A graph is *simple* if there are no parallel edges nor loops.

The goal is to give a bound on a simple connected graph with n vertices.

Definition 2.5 (Connected)**.**
1. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, and the vertex sets V_1 and V_2 are disjoint, then their *union* is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$
2. A graph is *connected* if it cannot be expressed as a union of graphs, and disconnected otherwise.

Definition 2.6 (Component)**.** Any *disconnected* graph G can be expressed as a union of connected graphs, each of which is called a *component* of G.

Theorem 2.2. Let G be a simple graph on n vertices with k componets. The number m of edges of G satisfies

$$n - k \leq m \leq (n - k)(n - k + 1)/2.$$

Lower Bound (By induction on the number of edges): The lower bound is true for a null graph G. We assume the lower bound holds true for number of edges less than m_0. Consider a graph G (suppose that it has n vertices and k components) with *minimal* edges, say m_0, i.e., removing one edge will increase the number of components by 1. Therefore, the resultant graph has n vertices, $k + 1$ components and $m_0 - 1$ edges. By the induction hypothesis, we imply

$$n - (k + 1) \leq m_0 - 1 \implies n - k \leq m_0.$$

Upper bound: It suffices to consider the case where each component of G is a complete graph (since we are considering the upper bound). Suppose that there are two components C_i and C_j with n_i and n_j vertices, respectively, and $n_i \geq n_j > 1$.

Now we argue that n_i should be larger than n_j as much as possible ($n_i + n_j$ keeps the same) to obtain more vertices: Consider the components B_i and B_j with $n_i + 1$ and $n_j - 1$ vertices, respectively. Then the toal number of vertices remains unchanged, but the total number of edges is more than the previous one:

$$\frac{n_i(n_i+1)}{2}+\frac{(n_j-2)(n_j-1)}{2}-\frac{(n_i-1)n_i}{2}-\frac{n_j(n_j-1)}{2} = n_i-n_j+1 > 0.$$

Therefore, in order to obtain the maximum number of edges, G must consists of a complete graph with $n - k + 1$ vertices and $k - 1$ isolated vertices.

Corollary 2.3. If a simple graph on n vertices has more than $(n-1)(n-2)/2$ edges, it must be connected.

Proof. The key insight is that the graph with more componets should have less vertices. The vertices for the graph with more than 1 components is upper bounded by $(n-1)(n-2)/2$, by applying Theorem (2.2). □

2.3 Edge & Vertex Connectivity

Definition 2.7 (Disconneting Set)**.** A *disconnecting (edge) set* of a graph G is a set of edges whose removal increases the number of components of G

Definition 2.8 (Cutset)**.** A *cutset* of a graph G is a *minimal disconnecting set*; a cut-set of size 1 is called a *bridge*

Definition 2.9 (Edge-Connectivity)**.** For connected graph G, its *edge-connectivity* $\lambda(G)$ is the number of edges of the smallest cutset of G. In other words, $\lambda(G)$ is the fewest number of edges to be removed such that the resulting graph is dis-connected.

We say G is *k-edge connected* if $\lambda(G) \geq k$

Definition 2.10 (Separating Vertex Set)**.** A *separating vertex set* of a connected graph G is a set of vertices whose deletion (together with their incident edges) disconnects G; we say v is a *cut-vertex* if a separating set has only one vertex v.

Definition 2.11 (Vertex-Connectivity)**.** For connected graph G, its *vertex-connectivity* $\kappa(G)$ is the minimum size S such that $G-S$ is disconnected or has only one vertex.

We say G is k *(-vertex) connected* if $\kappa(G) \geq k$.

For example, $\kappa(K_n) = n - 1$, where K_n denotes a complete graph with n vertices.

The Theorem (2.4) shows that the vertex-connectivity is more intrinsical than the edge-connectivity.

Theorem 2.4. For a connected graph G, we have

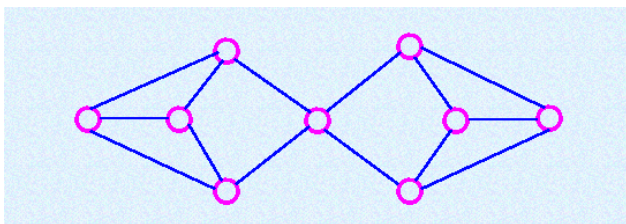
$$\kappa(G) \leq \lambda(G) \leq \delta(G),$$

where $\delta(G)$ denotes the smallest vertex-degree of G.

Proof. Upper Bound: The upper bound is trivial, since removing all the edges of the vertex with smallest degree will lead to a disconnected graph, i.e., $\lambda(G) \leq \delta(G)$.

Lower Bound: Skipped, since not understand. □

Example 2.1. It's possible for both inequality in Theorem (2.4) to be strict:



In such case, $\kappa(G) = 1, \lambda(G) = 2, \delta(G) = 3$.

Theorem 2.5 (Menger's Theorem)**.** A graph G is *k-edge-connected* if and only if any two distinct vertices of G are joined by at least k disjoint paths, i.e., no two of which have any edge in common.

Theorem 2.6 (Menger's Theorem)**.** A graph with at least $k+1$ vertices is k-connected if and only if any two distinct vertices of G are joined by at least k disjoint paths.

Definition 2.12 (Disconnecting Set)**.** A vw-disconnecting set of a graph G is a set of edges F of G such that every path from v to w includes an edge of F.

Theorem 2.7 (Menger's Theorem (edge form))**.** The *maximum number of edge-disjoint paths* connecting two distinct vertices v and w of a connected graph is equal to the *minimum number of edges in a vw-disconnecting set*.

Proof. It's trivial that the the maximum number of edge-disjoint paths cannot be more than the number of edges in a vw-disconnecting set.

The reverse direction relies on the induction on the number of edges. □

From Eulerian Problem into the Chinese Postman Problem Recall that a connected graph is *Eulerian* if there exists a closed trail that includes every edge of G. The necessary and sufficient condition is that every vertex has even degree.

When a graph is not Euliean, we want to find a walk that covers all the edges, but avoid repeating edges as much as possible. Or equivalently, how many and which edges need to be duplicated such that the resultant is Eulerian? This motivates the Chinese Postman Problem.

Definition 2.13 (The Chinese Postman Problem)**.** A postman has to deliver letters to a given neighbourhood. He needs to walk through all the streets in the neighbourhood and back to the post office. How can he design his route so that he walks the shortest distance?

Definition 2.14 (Connected)**.** A directed graph G is connected if the corresponding underlying graph is connected

Definition 2.15 (Strongly Connected)**.** A directed graph G is *strongly connected* if for any vertex v and w, there is a directed path from v to w

Definition 2.16 (Orientable)**.** We say an *undirected* graph is *orientable* if each edge can be *directed* such that the resulting graph is *strongly connected*.

Theorem 2.8. A connected graph is *orientable* if and only if every edge of G lies in at least one cycle.

Sufficiency. Choose any cycle C and direct its edges *cyclically*. If all edges of G have been oriented, the proof is complete; otherwise choose an edge, say e, that is not in C but adjacent to an edge of C. By hypothesis, the edge e is in some cycle C', and orient the edges in C' cyclically except for those already directed.

We proceed this operation, and thus directing at least one more edge for each time. Therefore, we are done until all edges have been oriented. The oriented edges form a strongly-connected graph. □

3

Hamiltonian Graphs

Motivation We are interested in the Eulerian graph, i.e., whether there exists a closed trail (that travels each edge once and only once), for a connected graph. There is a simple necessary and sufficient condition for a graph to be Eulerian: *every vertex has even degree*. Similarly, is there a cycle that includes every vertex once and only once, for a connected graph?

Definition 3.1 (Hamiltionian)**.** 1. A *Hamiltonian cycle* in a graph is a *cycle* that includes every vertex (once and only once).

2. A graph is *Hamiltonian* if it contains a Hamiltonian cycle.

3. A non-Hamiltonian graph is *semi-Hamiltonian* if there exist a *path* through every vertex.

However, not all graphs are *Hamitonian*. We are studying necessary or sufficient conditions for a graph to be Hamiltonian.

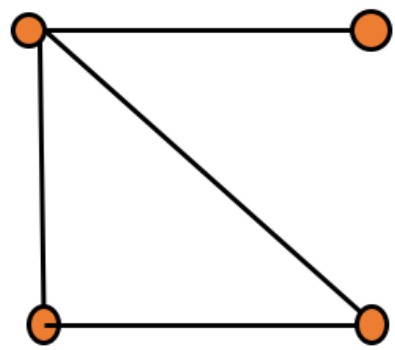
3.1 Necessary Conditions

Theorem 3.1. If G is Hamitonian, then for each non-empty set $S \subseteq V$, the graph $G - S$ has at most $|S|$ componenets.

As a result, *every Hamitonian graph must be* 2*-connected.*

Proof. Consider a Hamiltionian cycle in G, which must pass through all vertices including both $G - S$ and S. Each time when leaving a component of $G - S$, it must go next to a vertex in S, and must be a distinct vertex in S. Therefore, the number of components of $G - S$ cannot be more than the number of vertices in S. □

Remark 3.1. However, the converse may not be necessarily true. Consider the counter-example



3.2 Sufficient Conditions

Most sufficient conditions are of the form: *If the graph has enough edges, then it is Hamiltonian.*

Theorem 3.2 (Ore's theorem)**.** If G is a simple graph with $n \geq 3$ vertices, and if

$$\deg(v) + \deg(w) \geq n,$$

for each pair of non-adjacent vertices v and w, then G is Hamitonian.

Proof. Suppose on the contrary that G is not, and w.l.o.g., G is maximal. Thus there must be a path $v_1 \to v_2 \to \cdots \to v_n$ passing through each

vertex but v_1 and v_n are non-adjacent. Therefore, $\deg(v_1)+\deg(v_n) \geq n$, i.e., there must be a vertex v_i adjacent to v_1 with the property that v_{i-1} is adjacent to v_n, which leads to a Hamitonian cycle:

$$v_1 \to \cdots \to v_{i-1} \to v_n \to v_{n-1} \to v_{n-2} \to \cdots \to v_i \to v_1$$

□

Corollary 3.3. If G is a simple graph with $n \geq 3$ vertices, and $\deg(v) \geq \frac{1}{2}n$ for each vertex v, then G is Hamiltionian.

Theorem 3.4. If G is a simple graph with $n \geq 3$ vertices, and if G has at least $(n-1)(n-2)/2+2$ edges, then G is Hamitionian.

Proof. 1. If every vertex is adjacent to every other vertex, then G is complete, and therefore Hamitonian

2. Otherwise, let v and w be two non-adjacent vertices, and $H = G - \{v, w\}$ contains $(n-2)$ vertices, with maximum number of edges $(n-2)(n-3)/2$. There exists at least $(n-1)(n-2)/2+2-(n-2)(n-3)/2 = n$ vertices incident to either v or w. Therefore, $\deg(v) + \deg(w) \geq n$. By applying Ore's theorem, G must be Hamitonian.

□

Theorem 3.5. Let $G = (V_1 \cup V_2)$ be a bipartite graph. If G is Hamitonian, then $|V_1| = |V_2|$.

If G is semi-Hamitonian, then $|V_1|$ and $|V_2|$ differ by at most 1.

Proof. Consider the Hamitonian cycle. □

Corollary 3.6. The converse of Theorem (3.5) also holds if G is a *complete* bipartite graph with at least 3 vertices.

Proof. Construct the Hamitonian cycle. □

Definition 3.2 (Hamiltonian Closure)**.** The *Hamiltonian closure* of a graph $G = (V, E)$ is the graph $C(G)$ obtained from G by iteratively adding edges joining pairs of *non-adjacent* vertices whose degree sum is at least $|V|$, until no such pair remains.

If $G = C(G)$, then we say G is closed.

The closure does not depend on the sequence by which the edges are added.

Lemma 3.7. The *Hamitonian closure* of a graph $G = (V, E)$ is well-defined.

Proof. Suppose we add sequence of edges $\{e_1, \ldots, e_f\}$ to form G_1 and $\{f_1, \ldots, f_s\}$ to form G_2.

Note that a pair of vertices (u, v) are added iff theya are non-adjacent and the sum of degree is at least $n = |V|$. Thus by definition, f_1 is addable to G, and must be added at some point in the first sequence, i.e., $f_1 \in G_1$. Inductively, $f_1, f_2, \ldots, f_s \in E(G_1)$, which implies $E(G_2) \subseteq E(G_1)$. Similarly, $E(G_1) \subseteq E(G_2)$. □

Theorem 3.8 (Bondy-Chvatal)**.** A simple graph G is Hamitionian if and only if its closure $C(G)$ is Hamitionian

Proof. Clearly, if G is Hamitonian, so is $C(G)$. Conversely, let $\{G_0 := G, G_1, \ldots, C(G)\}$ be a sequence of graphs where each G_i is formed by performing a single closure step to G_{i-1}. Let k be the minimal value such that G_k is Hamiltonian. Let (w, u) be the edge added to form G_k from G_{j-1} and let C be a Hamiltionian cycle in G_k. Therefore, C must contain the edge (w, u), and suppose it is of the form

$$u := v_1 \to \cdots \to v_n = w \to u.$$

Now we argue that G_{k-1} is Hamiltonian. Since there are n edges incident to either u or v, there exists a vertex v_i incident to u such that v_{i-1} is incident to w in G_{k-1}, which leads to a Hamitonian cycle in G_{k-1}:

$$u = v_1 \to v_i \to v_{i+1} \to \cdots \to v_n = w \to v_{i-1} \to v_{i-2} \to \cdots \to v_2 \to v_1 = u$$

□

Theorem 3.9 (Chvatal)**.** Let G be a simple graph on $n \geq 3$ vertices with degrees $d_1 \leq d_2 \leq \cdots \leq d_n$. If for all $i < n/2$, either $d_i > i$ or $d_{n-i} \geq n - i$, then G is Hamiltonian.

Proof. It suffices to consider the case where G is closed. Suppose on the contrary that $G = C(G)$ is non-Hamitonian, which is not complete.

Among all pairs of non-adjacent vertices, let (u, v) be the pair with maximum degree sum, w.l.o.g., $\deg(u) \leq \deg(v)$. Since G is closed, we have $\deg(u) + \deg(v) < n$, i.e., $\deg(u) < n/2$. Suppose that $\deg(u) = k$, then every other vertex in V not adjacent to v must have degree no larger than $\deg(u) = k$; there are $n - 1 - \deg(v)$ such vertices. Note that $\deg(u) + \deg(v) \leq n - 1$ implies that $n - 1 - \deg(v) \geq \deg(u) = k$. Therefore, we have found k vertices of degree at most k.

Similarly, every vertex that is not adjacent to u has degree at most $\deg(v)$. By assumption, $\deg(v) < n - k$. There are $(n-1) - \deg(u)$ such vertices. Also, u it self has the degree no larger than $\deg(v)$, thus there are $n - k$ vertices with degree smaller than $n - k$. □

Consider for some sequence $(d_1, \ldots, d_n)$ we have $d_h \leq h$ and $d_{n-h} \leq n - h - 1$ for some $0 < h < n/2$. Now we want to construct a graph with this degree sequence that is non-Hamitonian.

Consider one possible example:

$$(\underbrace{h, \ldots, h}_{h \text{ times}}, \underbrace{n-h-1, \ldots, n-h-1}_{n-2h \text{ times}}, \underbrace{n-1, \ldots, n-1}_{h \text{ times}},)$$

The graph has two components: a complete bipartite graph K_{hh} with $V_1 = \{v_1, \ldots, v_h\}$ and $V_2 = \{v_{n-h+1}, \ldots, v_n\}$ and a complete graph K_{n-h} with $V = \{v_{h+1}, \ldots, v_n\}$.

Definition 3.3 (Hamiltonian for Directed Graph)**.** Consider a *directed* graph D. We say D is *Hamiltonian* if there is a directed cycle that includes every vertex of D.

A non-Hamiltonian digraph that contains a directed path through every vertex is semi-Hamiltonian.

We are considering the necessary and sufficient conditions for a digraph to be Hamiltonian; and are there some special types of graphs more likely to be Hamiltonian?

Definition 3.4 (Tournament)**.** A directed graph D is a *tournament* if there is exactly one arc between every pair of vertices. In other words, a tournament is an orientation of a complete simple graph.

Theorem 3.10 (Redei)**.** Every non-Hamiltonian tournament is semi-Hamiltonian, i.e., every tournament contains a Hamiltonian path

Proof. We show this result by induction on $|V| = n$. Suppose the tournament $T - v$ contains a Hamitonian path, say $v_1 \to \cdots \to v_{n-1}$, and show that T has a Hamitonian path. Consider three cases:

1. There is an arc $(v, v_1) \in T$
2. There is an arc $(v_{n-1}, v) \in T$
3. There is no arc for (v, v_1) and (v_{n-1}, v)

In previous two cases, the result is trivial. In the third case, define i to be the smallest index such that (v, v_i) is an arc in T. Therefore, we construct a Hamitonian path:

$$v_1 \to v_2 \to \cdots \to v_{i-1} \to v \to v_i \to v_{i+1} \to \cdots \to v_{n-1}$$

□

Theorem 3.11 (Camion)**.** Every strongly connected tournament is Hamitonian.

Proof. For a strongly connected tournament T, it must have at least 3 vertices. We will show that a tournament with $|V| = n$ contains cycles of length $3, 4, \ldots, n$ by induction.

1. Base Case: Consider any vertex v in T. Since T is strongly connected, there must be at least one vertex incident to v and at least one vertex incident from v. Let V_{in} be the set of vertices incident to v, V_{out} the set of vertices incident from v. Since T is strongly connected, there must be an arc $(v_o, v_i) \in V_{\text{in}} \times V_{\text{out}}$, and therefore a 3-cycle exists.

2. It remains to show that if there is a cycle C of length k, then there is also a cycle of length $k + 1$. Suppose there is a vertex c not in the cycle such that there exists arcs incident to c from some vertices in C, and arcs incident from c to some vertices in C. Thus there must be an arc (u, v) in C such that T contains arcs (u, c) and (c, v). Thus we have a $(k + 1)$-cycle.

3. If no such vertex exists, then for each vertex w not in the cycle, its arcs that are joined to vertices in the cycle are either all incident from w, or all incident to w. Let A and B be the two (disjoint) sets of vertices with such properties. Since T is strongly connected, neither A nor B can be empty. There must exist an arc from some vertex $b \in B$ to some vertex $a \in A$. Therefore, we can replace two consecutive arcs in the k-cycle, say (u, c) and (c, v) by the arcs $(u, b), (b, a), (a, v)$ to get a $(k + 1)$-cycle.

□

Theorem 3.12. Let D be a strongly connected digraph with n vertices. If $\text{outdeg}(v) \geq n/2$ and $\text{indeg}(v) \geq n/2$ for every vertex v, then D is Hamitonian.

The Travelling Salesman Problem Given the distance between each pair of cities (on a given list), we want to find a Hamiltonian cycle (that visits all the cities on the list and return to his starting point), with mimimum distance. How many Hamiltonian cycles in a complete graph? $\frac{(n-1)!}{2}$!

4

Trees

4.1 Characterisation for Trees

Definition 4.1 (Tree)**.** An undirected graph is *acyclic* if it has no cycles. A connected acyclic undirected graph is a *tree*.

- A vertex of degree 1 in a tree is called a *leaf*
- An acyclic (undirected) graph but *not connected* is called a *forest*

Remark 4.1. Trees are simple graphs. They are the simplest non-trivial graphs, with some nice properties.

Theorem 4.1. Let T be an undirected graph with n vertices. TFAE:

1. T is a tree
2. T contains no cycles and has $n-1$ edges
3. T is connected and has $n-1$ edges
4. T is connected and each edge of T is a bridge (i.e., a cutset of a *single* edge)
5. Any two vertices of T is connected by a unique path

6. T is acylic, but the addition of any new edge creates *exactly* one cycle

Proof. (1) implies (2): The proof is by induction on n. The removal of any edge of T disconnects the graph into two components, each of which is a tree. Therefore, applying the induction hyphothesis on each components gives the desired result. (2) implies (3): Suppose T is disconnected. Since T is acyclic, then each component is a tree. Therefore, the number of edges in each component tree is one fewer than the number of vertices, which implies the graph T has fewer than $n-1$ edges, which is a contradiction.

(3) implies (4): Suppose the removal of edge e of T does not disconnect the graph. Since $T-e$ is a tree, and the number of edges is $n-1$, we derive a contradiction.

(4) implies (5): Since T is connected, there is at least one pathbetween any pair of vertices. If a pair of vertices is connected by two distinct paths, then they must contain a cycle. The removal of an edge on this cycle does not disconnect the graph. Thus the edge is not a bridge.

(5) implies (6): T does not contain a cycle. Consider an edge e that is not in $V(T)$, and there is a path in T connecting the two end-vertices of e. Therefore, the addition of e creates a cycle. If there are two cycles in $T+e$, then both cycles must contain e, which implies there exists a cycle in $T+e$ not containing e, which is a contradiction.

(6) implies (1): Suppose that T is disconnected, then it's possible to add an edge to T that joins two different components but not create a cycle. □

4.2 Spanning Trees

Consider a simple connected graph G on n vertices. We are interested in the subgraphs of G that are trees.

Definition 4.2 (Spanning Tree)**.** A subgraph T of $G=(V,E)$ that is a tree and contains all the vertices V of G is a *spanning tree* of G.

Theorem 4.2. If $T=(V,F)$ is a spanning tree of a connected graph $G=(V,E)$, then

1. Each cutset of G contains an edge in T
2. Each cycle of G contains an edge in $\bar{T} := (V, E - F)$.

Proof. 1. Suppose K is a cutset of G that disconnects G into two components G_1 and G_2. Since T is a spanning tree, it must contain an edge joining one vertex in G_1 and one vertex in G_2

2. If C is a cycle in G with no edge in common with $E - F$, then all the edges in C are in Fm which contradicts that T is acyclic.

□

Theorem 4.3. If T and T' are spanning trees of a connected graph G and edge $e \in E(T) - E(T')$, then there is an edge $e' \in E(T') - E(T)$ such that $T - e + E'$ is a spanning tree of G.

Proof. By part (4) in Theorem (4.1), each edge of T is a cutset of T. Let k and k' be two components of $T - e$. Since T is a spanning tree of G, it is connected and there exists an edge $e' \in T'$ that joins a vertex in K to a vertex in K'. Therefore, the constructed $T - e + e'$ is connected and has $n - 1$ edges, therefore a spanning tree of G. □

4.3 Prufer Code

Motivation How many different (i.e. non-isomorphic) graphs are there with a given property? In particular, how many regular graph (i.e., each vertex has the same degree) of degree 2? How many regular graph of degree n? Unfortunately, it is almost impossible to express these results by simple closed-form formulas.

One application is in the enumeration of chemical molecules, i.e., how to count the saturated hydrocarbons with the formula C_nH_{2n+2}? Note that it is a tree since it is connected with $n + (2n + 2)$ vertices and $(4n + (2n + 2))/2 = 3n + 1$ edges.

Therefore, the problem turns into the enumeration of labelled trees with a given number of vertices.

In particular, we assume the graphs with label $1 - 2 - 3 - 4$ and $4 - 3 - 2 - 1$ are the same since they have the same adjacent matrix, but they are both different to $1 - 2 - 4 - 3$.

Theorem 4.4. There are $2^{n(n-1)/2}$ distinct labelled simple graphs with n vertices; and n^{n-2} distinct labelled trees with n vertices.

4.3.1 Prufer's Code

Now we aim to represent each tree with n vertices as a sequence of $n-2$ numbers selected from $\{1, 2, 3, \ldots, n\}$

Algorithm 1 Prufer Coding for a tree T

Input:
A tree T with n labelled vertices
Output:
The sequence $(p_1, \ldots, p_{n-2}) \subseteq \{1, 2, 3, \ldots, n\}$
1: Initialize $T_1 = T$ and $i = 1$;
2: **Stop** if T_i has only one edge;
3: Otherwise, let v_i be the lowest labelled *leaf* of T_i, and p_i be its neighbour. Let $T_{i+1} = T_i - v_i$;
4: Repeat previous step;

Given a sequence of $n-2$ numbers selected from $\{1, \ldots, n\}$, we can decode it into the corresponding labelled tree. Consider the nontrivial case $n \geq 3$:

Algorithm 2 Constructing a Tree from a (Prufer) Sequence

Input:
The sequence $(p_1, \ldots, p_{n-2}) \subseteq \{1, 2, 3, \ldots, n\}$ A tree T with n labelled vertices
Output:
A tree T with n labelled vertices
1: Initialize $V_0 = \{1, \ldots, n\}$ and $i = 1$;
2: Let $P_i = (p_i, p_{i+1}, \ldots, p_{n-2})$, and v_i be the lowest labelled vertex that do not appear in P_i Join it into vertex p_i, and let $V_i = V_{i-1} \backslash \{v_i\}$;
3: If $i = n-2$, then join the two remaining vertices in V_i. **STOP**;
4: Otherwise, increment i by 1 and repeat from step 2;

4.4 Minimum Spanning Tree

Given a network with costs labelled for each edges, we are interested in finding a spanning tree with minimum cost. The computation cost is high for finding each of the n^{n-2} trees.

Kruskal's Algorithm Let $G = (V, E)$ be a connected graph with n vertices with weights on the edges. Then the following algorithm gives a spanning tree $H = (V, F)$ of G with a minimum edge weight:

Algorithm 3 Kruskal's Algorithm

1: Initialize $E(H) = \emptyset$ and let the edges in E be sorted in non-decreasing order;
2: Let e be an edge in E of smallest weight. Add this edge to H as long as it does not create a cycle in H;
3: Repeat above step until obtaining $n - 1$ edges in H.

Theorem 4.5 (Optimality for finding MST)**.** Let T be a spanning tree of G. Let $e = \{v_1, v_2\}$ be an edge of T. The removal of e disconnects T into two components, say T_1 and T_2 with vertex sets V_1 and V_2 respectively. Let $K(G)$ be an edge cutset of G that disconnects G into two components with vertex sets V_1 and V_2.

Then T is a MST of G if and only if

$$\forall e \in \{v_1, v_2\} \in E(T), w(e) \leq w(f), \quad \forall f \in K(G)$$

Jarnik-Prim's Algorithm We denote $[W, V \setminus W]$ as the edge cutset that disconnects G into two components with vertex-sets W and $V \setminus W$. In other words, $[W, V \setminus W]$ is the set of edges in G that joins a vertex in $W \subseteq V$ to a vertex not in W.

Algorithm 4 Jarnik-Prim's Algorithm

1: Initialize $V_1 = \{v_1\}$ (any vertex), $T_1 = (V_1, E_1 = \emptyset)$, $k = 1$, $d(v_1) = 0, d(v) = \infty, \forall v \neq v_1$, $n(v) = v_1, \forall v \neq v_1$;
2: **Stop** if $V_k = V$;
3: Otherwise for $v \neq v_k$, let $d(v) = \min\{d(v), w(\{v, v_k\})\}$, and $n(v) = v_k$ if $d(v) = w(\{v, v_k\})$
4: Let $v_{k+1} = \arg\min_{v \neq V_k} d(v)$ (essentially looking for the edge of least weight in cutset $[V_k, V \setminus V_k]$), $V_{k+1} = V_k \cup \{v_{k+1}\}, T_{k+1} = (V_{k+1}, E_k \cup \{(v_{k+1}, n(v_{k+1}))\})$.
Increment k by 1 and go to step 2

5

Shortest Path

Problem setting Now we turn the attention to directed graphsm where there is a cost associated with each arc:

- How to find a shortest path from a given (origin) node to a given (destination) node?
- How to find a shortest path from a given (origin) node to each other node?

5.1 Dijkstra's Algorithm

Remark 5.1. The Dijkstra's algorithm does not work for negative arc lengths.

Efficiency of Dijkstra's Algorithm

- Initialization: $O(n)$
- Iteration: repeat for n times
  - Termination: $O(1)$

Algorithm 5 Dijkstra's Algorithm

Input:

The graph $G = (N, A)$ with cost function c on arcs

Output:

Shortest path from origin s to destination t

1: Initialize OPEN list $S := \emptyset$, $T = N$; $d(i) = M$ for each $i \in N$; $d(s) = 0$, $\text{pred}(s) = 0$;

2: **While** $|S| < n$**, do**

- Let $i = \arg\min\{d(k) \mid k \in T\}$; $S := S \cup \{i\}$; $T := T \setminus \{i\}$;

  - for each $(i, j) \in A$,

    if $d(j) > d(i) + c(i, j)$, then $d(j) = d(i) + c(i, j)$; $\text{pred}(j) = i$;

End While

  - Finding minimum: $O(n)$ comparisons. Indeed, the data structure technique can help to reduce into $O(m \log n)$ operations using binary heaps, and $O(m + n \log n)$ operations using Fiboniacci heaps.

- Updating: $O(m)$ in total

Thus Dijkstra's algorithm takes $O(n)+n(O(1)+O(n))+O(m) = O(n^2)$ steps.

5.2 All-Pairs Shortest Paths

Given a directed graph $G = (N, A)$ with arc lengths c. Here we allow negative arc length but assume G has no cycles of negative length.

Theorem 5.1 (Optimality Condition)**.** For every pair of nodes (i, j), let $d[i, j]$ denote the length of a directed path from i to j. Then $d[i, j]$ is the shortest path from i to j for all $i, j \in N$ if and only if

1. $d[i, j] \leq d[i, k] + d[k, j]$ for $\forall i, j, k \in N$

2. $d[i,j] \leq c(i,j)$ for all $(i,j) \in A$

An obvious label correcting type algorithm is developed based on the optimality condition. We start with some labels $d[i,j]$, and update the labels until the optimality conditions are satisfied.

Algorithm 6 Dijkstra's Algorithm

1: Initialize

$$\begin{aligned} d[i,j] &= \infty, \forall [i,j] \in N \times N \\ d[i,i] &= 0, \forall i \in N \\ d[i,j] &= c(i,j), \text{ for each } (i,j) \in A \end{aligned}$$

2: While there exists 3 nodes i, j, k such that $d[i,j] > d[i,k] + d[k,j]$,

$$\text{do } d[i,j] := d[i,k] + d[k,j]$$

End While

Algorithm 7 Floyd-Warshall Algorithm

1: Initialize

$$\begin{aligned} d[i,j] &= \infty, \forall [i,j] \in N \times N \\ pred[i,j] &= 0, \forall [i,j] \in N \times N \\ d[i,i] &= 0, \ \forall i \in N; d[i,j] \qquad = c(i,j), \ \forall (i,j) \in A \end{aligned}$$

2: For each $k = 1 : n$, for each $[i,j] \in N \times N$, if $d[i,j] > d[i,k] + d[k,j]$, then

$$d[i,j] = d[i,k] + d[k,j] \text{ and } pred[i,j] = pred[k,j]$$

Remark 5.2. By adding one more test when updating the labels in the Floyd-Warshall algorithm, we can detect negative cycles if they exist in the graph:

- If $i = j$, check if $d[i, i] < 0$
- If $i \neq j$, check if $d[i, j] < -nC$, where C is the largest arc length.

If either two cases are true, then the graph contains a negative cycle.

Faster Implementation of Dijkstra's Algorithm Consider large and sparse graphs, i.e., the number of edges m is much smaller than n^2. We can apply the modified Dijkstra's algorithm n times, which is faster than Floyd-Warshall algorithm. With a d-heap implementation, which is a technique in Data structure, the Dijkstra's algorithm takes

$$O(m \log_d n + nd \log_d n) \text{ operations}$$

Taking $d = \max\{2, \lceil m/n \rceil\}$, we conclude that Dijkstra's algorithm take $O(m \log_2 n)$ steps in this case.