

Week 8: Introduction to Graph Theory

Algorithms & Data Structures

Thorben Klabunde

www.th-kl.ch

November 10, 2025

Agenda

- 1 Mini-Quiz
- 2 Assignment
- 3 What is a Graph?
- 4 Properties of Graphs
- 5 Eulerian Walks
- 6 Additional Practice
- 7 Peer Grading

Mini-Quiz

For the longest ascending subsequence (LAT), which partial problem corresponds to the following recursion?

$$DP(i, \ell) = \begin{cases} 1, & \text{if there is } j < i \text{ with } DP(j, \ell - 1) = 1 \text{ and } A[j] < A[i] \\ 0, & \text{otherwise} \end{cases}.$$

- a. $DP(i, \ell) = "A[i]"$ is the smallest ending of an ascending subsequence of length ℓ in $A[1 \dots i]$.
- b. $DP(i, \ell) =$ "there is an ascending subsequence of length ℓ ending in i ".

Your answer is correct.

In the definition, we only ask that there exists a j smaller than i where the DP array is 1 and the element there is smaller than the element at position i . This precisely means that we can extend an increasing subsequence of length $\ell - 1$ to one of length ℓ .

The correct answer is: $DP(i, \ell) =$ "there is an ascending subsequence of length ℓ ending in i ".

Recall the setting of the Subset Sum problem. We are given a list of n integers and a number b and are asked to answer whether there exists a subset in the list that sums to b .

If b is at most $10n$, there exists a $O(n^2)$ time algorithm that solves the problem.

☒ True ✓

☐ False

We have seen from the lecture an algorithm that runs in time $O(bn)$. Setting $b = 10n$, we see that the statement is true.

The correct answer is 'True'.

Consider the Subset Sum problem. Let x be the number of different values $s \geq 0$ which can be expressed as a subset sum of $A[1..i]$. Similarly, let y be the number of different values $s \geq 0$ which can be expressed as a subset sum of $A[1..i+1]$. Then, $y \leq 2x$.

☒ True 

☐ False

As we saw in the lecture, there are two ways to achieve a number when including the $(i+1)$ -th element as a possible summand. If we do not put it in the subset, we have x numbers possible. If we do, there are again x possible numbers (each of the form $a + A[i+1]$, where a is a number equal to a subset sum from $A[1..i]$). The claim follows.

The correct answer is 'True'.

Consider the approximation algorithm we saw in the lecture for Knapsack. Let OPT be an optimal solution (i.e. set of elements chosen) for the original problem and let $\widetilde{OPT}$ be an optimal solution for the problem with rounded profits. What does the algorithm compute?

- ☐ a. OPT
- ☒ b. $\widetilde{OPT}$ ✓
- ☐ c. None of the above.

Your answer is correct.

We saw this verbatim in lecture.

The correct answer is: $\widetilde{OPT}$

Let $G = (V, E)$ be a graph with 5 vertices v_1, v_2, v_3, v_4, v_5 . Suppose that

$$\deg(v_1) = 2, \deg(v_2) = 2, \deg(v_3) = 3, \deg(v_4) = 3, \deg(v_5) = 4.$$

How many edges does G have? (Your answer should consist of a single integer.)

Answer:



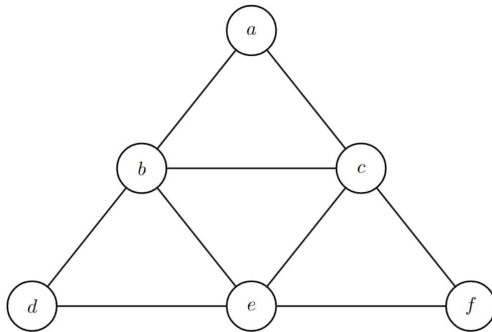
We know from the lecture that, for any graph, $\sum_{v \in V} \deg(v) = 2|E|$. So in this case, $2|E| = 2 + 2 + 3 + 3 + 4$, meaning $|E| = 7$.

The correct answer is: 7

Let G be a graph with x connected components.

True	False			
		Removing any edge yields a graph with at most $x + 1$ connected components.		
		Removing any vertex (along with the incident edges) yields a graph with at most $x + 1$ connected components.		
		Removing any vertex (along with the incident edges) with degree at most 4 yields a graph with at most $x + 3$ connected components.		

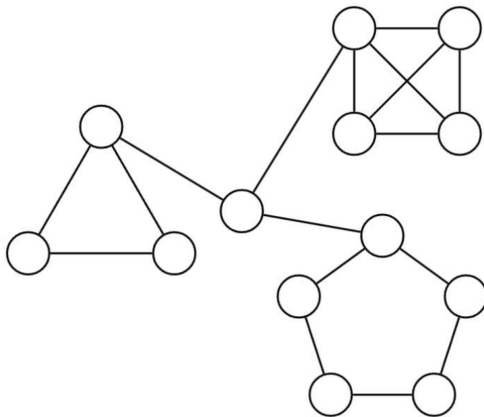
Scoring method: **Subpoints** 



The above graph contains a closed Eulerian walk.

☒ True ✓

☐ False



True	False			
☐	☐	The above graph has a Hamilton path.	☒	
☐	☐	It is possible to add one edge so that the graph contains a Hamilton path.	☒	

There cannot be a Hamilton path here, since if we remove the central vertex, the graph collapses into three connected components. This means that we have to use it to move from one to another. In an alleged Hamilton path, we would visit vertices v_1, v_2, v_3 in some order, where each of these vertices belongs to a different connected component. Now, the central vertex exists only once in the alleged Hamilton path, which means that one of the subpaths from v_1 to v_2 or v_2 to v_3 does not include the central vertex, which cannot happen.

We have pinpointed the problem, and one way to fix it is to add an edge across two of the connected components. We still need to be careful about this choice, but one exists. For example, the edge between the two vertices geometrically closest to the central vertex.

The above graph has a Hamilton path.: False

It is possible to add one edge so that the graph contains a Hamilton path.: True

Answer the following True/False questions.

True	False			
☒	☐	A polynomial time algorithm is known for deciding whether a closed Eulerian walk exists in a graph.	☒	
☐	☒	A polynomial time algorithm is known for deciding whether a Hamilton path exists in a graph.	☒	

Scoring method: **Subpoints** 

Assignment

Exercise 7.1 1-3 subset sums (1 point).

Let $A[1, \dots, n]$ be an array containing n positive integers, and let $b \in \mathbb{N}$. We want to know if there exists a subset $I \subseteq \{1, 2, \dots, n\}$, together with multipliers $c_i \in \{1, 3\}$, $i \in I$ such that:

$$b = \sum_{i \in I} c_i \cdot A[i].$$

If this is possible, we say b is a 1-3 subset sum of A . For example, if $A = [16, 4, 2, 7, 11, 1]$ and $b = 61$, we could write $b = 3 \cdot 16 + 4 + 3 \cdot 2 + 3 \cdot 1$.

Describe a DP algorithm that, given an array $A[1, \dots, n]$ of positive integers, and a positive integer $b \in \mathbb{N}$ returns True if and only if b is a 1-3 subset sum of A . Your algorithm should have asymptotic runtime complexity at most $O(b \cdot n)$.

3. *Recursion:* DP can be computed recursively as follows:

$$DP[0][s] = \text{False} \quad 1 \leq s \leq b \quad (1)$$

$$DP[a][0] = \text{True} \quad 0 \leq a \leq b \quad (2)$$

$$DP[a][s] = DP[a-1][s] \text{ or } DP[a-1][s - A[a]] \text{ or } DP[a-1][s - 3 \cdot A[a]] \quad 1 \leq a \leq n, \quad (3)$$

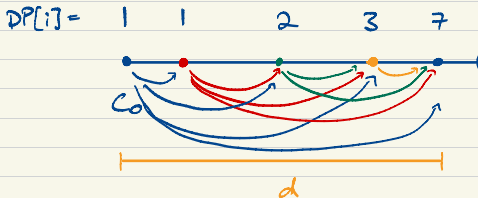
$$1 \leq s \leq b.$$

Note that in equation (3), the entries ' $DP[a-1][s - A[a]]$ ' and ' $DP[a-1][s - 3 \cdot A[a]]$ ' might fall outside the range of the table, in which case we treat them as False.

Exercise 7.3 Road trip.

You are planning a road trip for your summer holidays. You want to start from city C_0 , and follow the only road that goes to city C_n from there. On this road from C_0 to C_n , there are $n - 1$ other cities $C_1, \dots, C_{n-1}$ that you would be interested in visiting (all cities $C_1, \dots, C_{n-1}$ are on the road from C_0 to C_n). For each $0 \leq i \leq n$, the city C_i is at kilometer k_i of the road for some given $0 = k_0 < k_1 < \dots < k_{n-1} < k_n$.

You want to decide in which cities among $C_1, \dots, C_{n-1}$ you will make an additional stop (you will stop in C_0 and C_n anyway). However, you do not want to drive more than d kilometers without making a stop in some city, for some given value $d > 0$ (we assume that $k_i < k_{i-1} + d$ for all $i \in [n]$ so that this is satisfiable), and you also don't want to travel backwards (so from some city C_i you can only go forward to cities C_j with $j > i$).



1. *Dimensions of the DP table:* The DP table is linear, and its size is $n + 1$.
2. *Subproblems:* $DP[i]$ is the number of possible routes from C_0 to C_i (which stop at C_i).
3. *Recursion:* Initialize $DP[0] = 1$.

For every $i > 0$, we can compute $DP[i]$ using the formula

$$DP[i] = \sum_{\substack{0 \leq j < i \\ k_i \leq k_j + d}} DP[j]. \quad (4)$$

This recursion is correct since city C_i can be reached from C_j (for $0 \leq j < i$) if and only if $k_i \leq k_j + d$. Summing up the number of routes from C_0 to these C_j , we get the number of routes from C_0 to C_i .

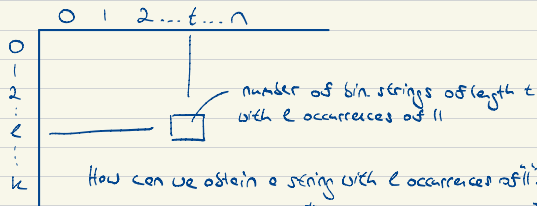
4. *Calculation order:* We can calculate the entries of DP as follows:

for $i = 0 \dots n$ **do**
 Compute $DP[i]$.
5. *Extracting the solution:* All we have to do is read the value at $DP[n]$.
6. *Running time:* For $i = 0$, $DP[0]$ is computed in $O(1)$ time. For $i \geq 1$, the entry $DP[i]$ is computed in $O(i)$ time (as we potentially need to take the sum of i entries). Therefore, the total runtime is $O(1) + \sum_{i=1}^n O(i) = O(n^2)$.

Exercise 7.4 String counting (1 point).

Given a binary string $S \in \{0,1\}^n$ of length n , let $f(S)$ be the number of times "11" occurs in the string, i.e. the number of times a 1 is followed by another 1. In particular, the occurrences do not need to be disjoint. For example $f(\underline{11}10\underline{11}) = 3$ because the string contains three 1 (underlined) that are followed by another 1. Given n and k , the goal is to count the number of binary strings S of length n with $f(S) = k$.

Describe a DP algorithm that, given positive integers n and k with $k < n$, reports the required number. Your solution should have complexity at most $O(nk)$.



this does not work!

to either $x0, x \in \{0,1\}^*$ s.t. x cont. k times "11"

or $x1, x \in \{0,1\}^*$ s.t. x cont. $k-1$ "11" & add 1

$x \in \{0,1\}^*$

$(x0 \Rightarrow k \text{ occ. in } x)$

$(x1 \Rightarrow k-1 + 1 = k)$

2D does not allow us to keep track of all the required information on the ending!

$\Rightarrow$ Use $DP[1 \dots k][1 \dots n][0 \dots 1]$ (two additional dimensions for the ending)

1. *Dimensions of the DP table:* $DP[1 \dots n][0 \dots k][0 \dots 1]$.
2. *Subproblems:* The entry $DP[i][j][l]$ describes the number of strings of length i with j occurrences of "11" that end in l .
3. *Recursion:* The base cases for $i = 1$ are given by $DP[1][0][0] = 1$ (the string "0"), $DP[1][0][1] = 1$ (the string "1"), $DP[1][j][0] = 0$ and $DP[1][j][1] = 0$ for $1 \leq j \leq k$. The update rule is as follows: For $1 < i \leq n$ and $0 \leq j \leq k$, to get a string of length i with j occurrences of "11" ending in 0, we can append "0" to a string of length $i - 1$ with j occurrences of "11" ending in 0 or 1, which gives

$$DP[i][j][0] = DP[i-1][j][0] + DP[i-1][j][1].$$

no new occurrences of 11 added by extending with 0

To get a string of length i with j occurrences of "11" ending in 1, we can append "1" to a string of length $i - 1$ with j occurrences of "11" ending in 0 or to a string of length $i - 1$ with $j - 1$ occurrences of "11" ending in 1 (if $j > 0$). Thus,

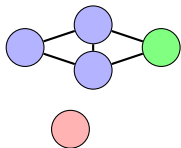
$$DP[i][j][1] = \begin{cases} DP[i-1][j][0], & \text{if } j = 0 \\ \underbrace{DP[i-1][j][0]}_{\substack{\times 01 \\ \text{no new 11}}} + \underbrace{DP[i-1][j-1][1]}_{\substack{\times 11 \\ \text{one new "11"}}}, & \text{if } j > 0. \end{cases}$$

What is a Graph?

Motivation: Graphs Are Everywhere

Three examples of real-world problems:

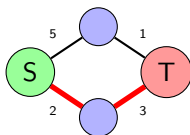
1. Reachability



Social Networks

"Can Alice reach Bob through friends?"

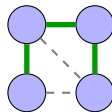
2. Shortest Paths



Navigation

"Fastest route from home to university?"

3. Spanning Trees



Network Design

"Connect all cities with minimum railway length"

Key Insight

Graphs provide a **unifying language** to model relationships and solve a large number of problems!

Definition: What is a Graph?

Definition 1.1 (Graph)

A **graph** G is a pair $G = (V, E)$, where:

- V is a finite, non-empty set of **vertices** (or *nodes*)
 - $E \subseteq \binom{V}{2} := \{\{x, y\} \mid x, y \in V, x \neq y\}$ is a set of **edges**
-
- An edge $\{u, v\} \in E$ connects vertices u and v
 - We write $\{u, v\}$ for an edge

Note: By this definition, we exclude loops (edges from a vertex to itself) and multiple edges between the same pair of vertices.

Example: A Simple Graph

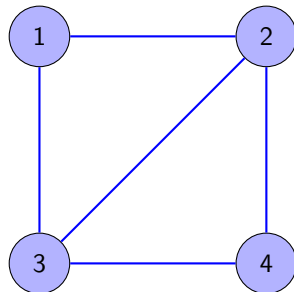
Graph $G = (V, E)$

$$V = \{1, 2, 3, 4\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$$

Properties:

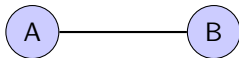
- 4 vertices: $|V| = 4$
- 5 edges: $|E| = 5$



Directed vs. Undirected Graphs

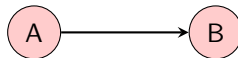
Undirected Graph

- Edges have no direction
- $\{u, v\} = \{v, u\}$
- Examples: friendships, physical connections



Directed Graph (Digraph)

- Edges have direction (arrows)
- $(u, v) \neq (v, u)$
- Examples: one-way streets

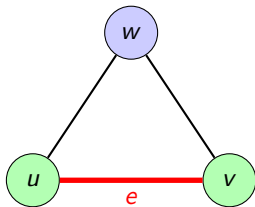


Today's focus: We'll primarily work with **undirected graphs**, but the concepts extend naturally to directed graphs.

Adjacency and Incidence

Let $G = (V, E)$ be a graph, $u, v \in V$, and $e \in E$.

- Vertices u and v are **adjacent** if $\{u, v\} \in E$
- u and v are called the **endpoints** of edge $e = \{u, v\}$
- A vertex u and an edge e are **incident** if u is an endpoint of e



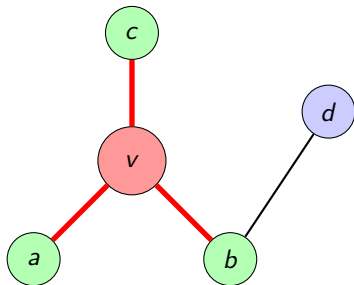
- u and v are **adjacent**
- e is **incident** to both u and v
- u and w are also adjacent

Neighborhood and Degree

Definition 1.2 (Neighborhood and Degree)

For a vertex $v \in V$ in graph $G = (V, E)$:

- The **neighborhood** of v is: $N_G(v) := \{u \in V \mid \{v, u\} \in E\}$
- The **degree** of v is: $\deg_G(v) := |N_G(v)|$



For vertex v :

- $N(v) = \{a, b, c\}$
- $\deg(v) = 3$

Other degrees:

- $\deg(a) = 1$
- $\deg(b) = 2$
- $\deg(c) = 1$
- $\deg(d) = 1$

Connectedness and Connected Components

Definition: Reachability

A vertex u **reaches** v (written $u \rightsquigarrow v$) if there exists a path from u to v .

Definition: Connected

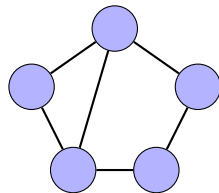
A graph G is **connected** if every vertex reaches every other vertex.

Definition: Connected Component

A **connected component** is a maximal connected subgraph and the equivalence class of the "reaches" relation.

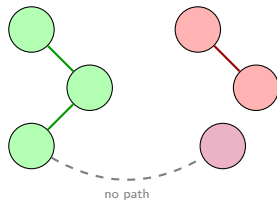
Note: Connected components partition the vertices. Each vertex belongs to exactly one component.

Connected Graph



1 connected component

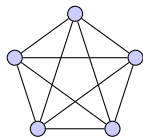
Disconnected Graph



3 connected components

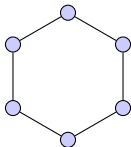
Special Types of Graphs

Complete Graph K_n



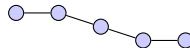
All vertices pairwise connected
 K_5 (5 vertices, 10 edges)

Cycle C_n



Circular connection
 C_6 (6 vertices, 6 edges)

Path P_n



Linear connection
 P_5 (5 vertices, 4 edges)

These are **canonical examples** that appear frequently in graph theory and algorithms.

Your Turn: Maximum Number of Edges

Determine the maximum number of edges a Graph $G = (V, E)$ can have, i.e., the number of edges in the complete graph K_n in terms of n .

Discuss with your neighbor (2-3 minutes)

Maximum Number of Edges

Determine the maximum number of edges a Graph $G = (V, E)$ can have.

Recall that $E \subseteq \binom{V}{2}$, where $\binom{V}{2} = \{ \{u, v\} \mid u, v \in V \}$

The unique graph $G = (V, E)$ on $|V| = n \in \mathbb{N}$ vertices that satisfies $E = \binom{V}{2}$ is the complete graph.

Therefore the maximum number of edges must be $\binom{|V|}{2} = \binom{n}{2} = \frac{n!}{(n-2)! \cdot 2!} = \frac{n(n-1)}{2}$

Alternative:

Let $G = (V, E)$ be K_n , s.t. $|V| = n$.

By Handsh.L.: $\sum_{v \in V} \deg(v) = 2 \cdot |E|$. Notice: $\deg(v) = n-1$ for $v \in V$ since $G = K_n$.

$$\Rightarrow \sum_{v \in V} \deg(v) = n \cdot (n-1) = 2 \cdot |E| \Rightarrow |E| = \frac{n(n-1)}{2}.$$

Special Types of Graphs - Trees

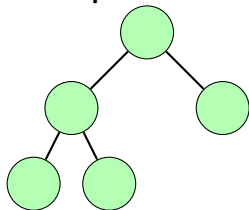
Definition 1.4 (Tree)

An undirected graph $G = (V, E)$ is a **tree** if it is:

- **Connected**, and
- **Acyclic** (contains no cycles).

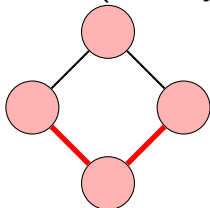
A **forest** is an acyclic graph (i.e., a collection of disjoint trees).

Example: A Tree



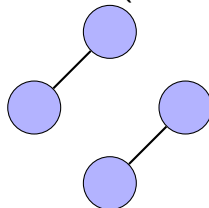
Connected and Acyclic

Not a Tree (Has a cycle)



Connected, but not acyclic

Not a Tree (A forest)

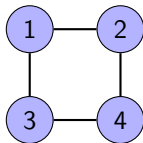


Acyclic, but not connected

How to Represent a Graph?

Two main data structures for storing graphs:

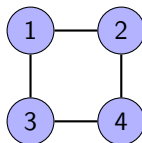
Adjacency Matrix



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$A[i][j] = 1$ if edge $\{i, j\}$ exists

Adjacency List



1: {2, 3}
2: {1, 4}
3: {1, 4}
4: {2, 3}

Each vertex stores its neighbors

Adjacency Matrix vs. Adjacency List: The Tradeoff

Operation	Adjacency Matrix	Adjacency List
Space	$O(V ^2)$	$O(V + E)$
Check if edge $\{u, v\}$ exists	$O(1)$	$O(\min\{\deg(u), \deg(v)\})$
Find all neighbors of v	$O(V)$	$O(\deg(v))$
Add edge	$O(1)$	$O(1)$
Remove edge $\{u, v\}$	$O(1)$	$O(\deg(u) + \deg(v))$

Use Adjacency Matrix when:

- Dense graphs ($|E| \approx |V|^2$)
- Need fast edge lookups
- Graph is small

Use Adjacency List when:

- Sparse graphs ($|E| \ll |V|^2$)
- Need to iterate over neighbors
- Graph is large

Most real-world graphs are sparse, so adjacency lists are typically preferred!

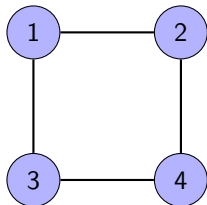
Properties of Graphs

The Handshake Lemma

Theorem 1.2 (Handshake Lemma)

For every graph $G = (V, E)$:

$$\sum_{v \in V} \deg(v) = 2|E|$$



Degrees:

- $\deg(1) = 2$
- $\deg(2) = 2$
- $\deg(3) = 2$
- $\deg(4) = 2$

$$\sum \deg(v) = 2 + 2 + 2 + 2 = 8$$

$|E| = 4$, so $2|E| = 8 \checkmark$

Proof of the Handshake Lemma

Proof.

Intuition: Each edge contributes 2 to the sum of degrees (once for each endpoint).

- Consider the sum $\sum_{v \in V} \deg(v)$
- This counts how many times vertices are incident to edges
- Each edge $e = \{u, v\}$ has exactly 2 endpoints: u and v
- So edge e is counted exactly twice in the sum:
 - Once when counting $\deg(u)$
 - Once when counting $\deg(v)$
- Since there are $|E|$ edges, and each is counted twice:

$$\sum_{v \in V} \deg(v) = 2|E|$$



A Surprising Consequence

Corollary 1.3

In every graph $G = (V, E)$, the **number of vertices with odd degree is even**.

Let's prove this together!

- Let V_{even} = vertices with even degree
- Let V_{odd} = vertices with odd degree
- We know: $V = V_{\text{even}} \cup V_{\text{odd}}$ (disjoint union)

Your task: Use the Handshake Lemma to prove $|V_{\text{odd}}|$ is even.

Proof: Number of Odd-Degree Vertices is Even

Proof:

Work through the proof with your neighbor (2-3 minutes)

Hint: What can you say about the parity of $\sum_{v \in V} \deg(v)$?

Solution: Odd-Degree Vertices

Proof.

Partition V into V_{even} (even degree) and V_{odd} (odd degree).

By the Handshake Lemma:

$$\sum_{v \in V} \deg(v) = \sum_{v \in V_{\text{even}}} \deg(v) + \sum_{v \in V_{\text{odd}}} \deg(v) = 2|E|$$

Key observations:

- $\sum_{v \in V_{\text{even}}} \deg(v)$ is even (sum of even numbers)
- $2|E|$ is even
- Therefore: $\sum_{v \in V_{\text{odd}}} \deg(v)$ must be even

But a sum of odd numbers is even **only if** there are an **even number** of them!

$\Rightarrow |V_{\text{odd}}|$ is even.



Eulerian Walks

Important Concepts: Walks, Paths, and Cycles

Definitions

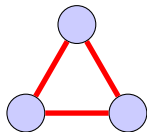
Let $G = (V, E)$ be a graph. A sequence of vertices $(v_0, v_1, \dots, v_k)$ is:

- A **walk** if $\{v_i, v_{i+1}\} \in E$ for all $0 \leq i \leq k-1$
 - The **length** is k (number of edges)
 - Vertices can be repeated
- A **closed walk** if it is a walk, $k \geq 2$, and $v_0 = v_k$
 - Starts and ends at the same vertex
- A **path** if it is a walk and all vertices are distinct ($v_i \neq v_j$ for $i \neq j$)
 - No vertex is visited twice
- A **cycle** if it is a closed walk, $k \geq 3$, and all vertices except $v_0 = v_k$ are distinct
 - A closed path (returns to start)

Eulerian Walks and Hamiltonian Paths

Eulerian Concepts (about edges):

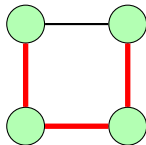
- An **Eulerian walk** is a walk that contains every edge exactly once
- A **closed Eulerian walk** is a closed walk that contains every edge exactly once



Traverses all edges

Hamiltonian Concepts (about vertices):

- A **Hamiltonian path** is a path that contains every vertex
- A **Hamiltonian cycle** is a cycle that contains every vertex



Visits all vertices

Key Distinction: Eulerian = *edges*, Hamiltonian = *vertices*

Euler's Answer: When Does a Closed Eulerian Walk Exist?

Theorem 1.31 (Euler's Theorem)

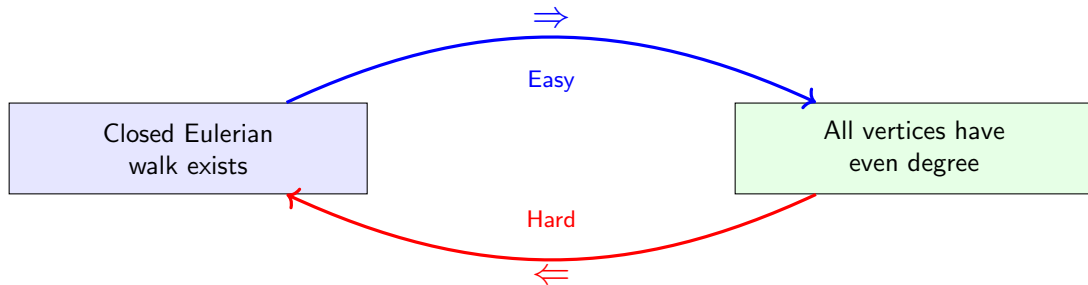
A **connected** graph G has a closed Eulerian walk **if and only if** every vertex has even degree.

Why does this make sense?

- Every time we *enter* a vertex, we must also *leave* it
 - This uses up 2 edges at that vertex
 - If we traverse all edges exactly once, we must enter/leave each vertex the same number of times
- $\implies$ Each vertex must have even degree!

Proof Strategy

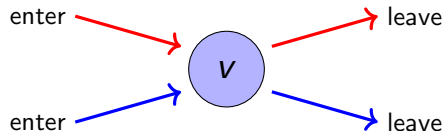
We prove both directions:



Proof: Direction $\Rightarrow$ (Easy)

Assume: There exists a closed Eulerian walk W

Show: All vertices have even degree



Each visit uses 2 edges

Pick any vertex v . In walk W , every time we pass through v , we use 2 edges (one in, one out). This holds even for the start/end vertex.

Therefore: $\deg(v) = 2 \times (\# \text{ of visits to } v)$ is even.



Proof: Direction $\Leftarrow$ (Hard) - Part 1

Assume: All vertices have even degree and G is connected

Show: A closed Eulerian walk exists

Construction: Start at any vertex v_0 . Walk randomly, never repeating an edge.

Key Observation

Claim: This walk must eventually return to v_0 .

Proof: Suppose the walk gets stuck at vertex $u \neq v_0$. During the walk, we visited u some number of times (say k times) before getting stuck. Each visit used 2 edges. Plus the final arrival used 1 edge.

Total edges used at u : $2k + 1$ (odd number). But $\deg(u)$ is even! So there must be unused edges at u .

Contradiction! We're not stuck. Therefore $u = v_0$.

So we have built a closed walk W starting and ending at v_0 .

Proof: Direction $\Leftarrow$ (Hard) - Part 2

We have a closed walk W . Does it use all edges?

Case 1: Yes, W uses all edges. Then we're done! W is our closed Eulerian walk. ✓

Case 2: No, some edges remain unused.

Notice

After deleting edges of W , the remaining graph may be disconnected. But each connected component still has all vertices with even degree (we removed 2 edges from each vertex on W). So we can apply the same procedure to each component independently!

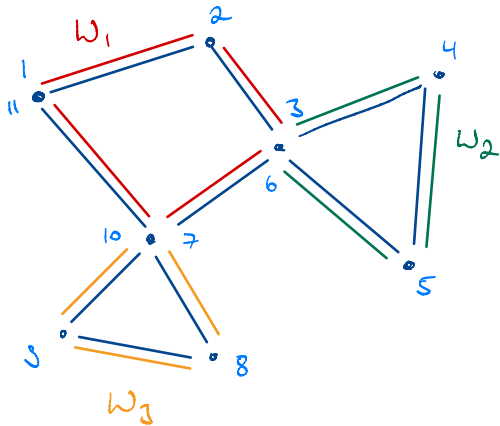
Extension Step: Since G is connected, there exists a vertex u on walk W that has unused edges. Starting from u , build a new closed walk W' (using the same method as before).

Merge the walks: Insert W' into W at vertex u :

Follow W until u $\xrightarrow{\text{detour}}$ Follow W' $\xrightarrow{\text{resume}}$ Continue W

Repeat this process. Since G has finitely many edges, we eventually use them all. □

Visual Example: Building a Closed Eulerian Walk



What About Eulerian Walks (Not Closed Walks)?

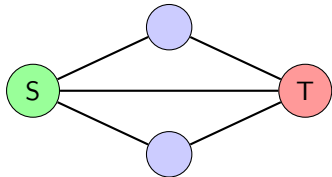
Relaxation: What if we don't need to return to the start?

Theorem (Eulerian Walk)

A connected graph has an Eulerian **walk** (but not a closed walk) if and only if it has **exactly 2 vertices of odd degree**.

Intuition:

- The walk must *start* at one odd-degree vertex
- It must *end* at the other odd-degree vertex
- All other vertices must have even degree (enter = leave)



Degrees: $S=3$, $A=2$, $B=2$, $T=3$

Walk: $S \rightarrow A \rightarrow T \rightarrow B \rightarrow S \rightarrow T$

(Starts at S, ends at T)

Questions?

Additional Practice

Sufficient Condition for Cycles

Lemma: If a graph G has $\deg(v) \geq 2$ for every vertex v , then G must contain a cycle.

Sufficient Condition for Cycles

Lemma: If a graph G has $\deg(v) \geq 2$ for every vertex v , then G must contain a cycle.

Let $p = (v_0, v_1, \dots, v_k)$ be a longest path in G .

Consider v_k and notice that since p is a path, v_k cannot have been visited before, i.e., $\forall i \in \{0, \dots, k-1\} (v_i \neq v_k)$. Accordingly, one edge of v_k remains unused since by assumption $\deg(v_k) \geq 2$.

Let this be $e = \{v_k, u\}$.

Notice that if $u \in \{v_0, \dots, v_{k-1}\}$ we can build a strictly longer path, a contradiction.

Then $u \neq v_{k-1}$ since $\{v_{k-1}, v_k\}$ is in the path p .

Thus, $u = v_i$ for $i \in \{0, \dots, k-2\}$ and there exists a cycle $(v_i, v_{i+1}, \dots, v_k, v_i)$ since $(v_i, \dots, v_k)$ is a path.



CodeExpert - Determine if a Graph is Eulerian

Like the last two weeks, you can find a CodeExpert template on my website. **Copy** the contents of the **main file** into the main file of the "**Welcome**" exercise. Also copy the **custom.in** and **custom.out** test-cases into the corresponding files in "Welcome".

You are given a graph in the form of an **adjacency list**, where **each vertex** is represented by a **natural number** $0 \leq v \leq n - 1$, $n = |V|$, and the **neighbors** of v can be **accessed through** **`E.get(v)`** (returns a list of the neighboring vertices).

Implement the `eulerian()` method such that it determines if the given graph is Eulerian or not.

Note, one condition requires you to check the **connectedness**, which you have not looked at in the lecture yet. However, the algorithm to do so is quite intuitive. Give it a shot!

Peer Grading

This week's peer-grading exercise is **Exercise 7.1**
Please follow the usual process:

- Grade the assigned group's submission.
- Give constructive feedback and upload your feedback to Moodle by 23.59 tonight, at the latest.
- Contact me if you don't receive a submission to grade.