Algorithms and Data Structures

Exercise Session 9

https://n.ethz.ch/~ahmala/and

https://vis.ethz.ch/en/events/624/

- General
  - Forum Rules and Annoucem...
- Forums
  - Organization
  - Lectures
  - Programming exercise
  - Theory Exercises
- Theory exercises
  - Theory exercise 2
  - Theory exercise 3
  - Theory exercise 4
  - Theory exercise 4
  - Theory exercise 5
  - Theory exercise 6
  - Theory exercise 7
  - Theory exercise 8
  - Theory exercise 9
- Test exam
  - Java Documentation
  - Test exam

Test exam

Quiz Settings Questions Results Question bank More

Opens: Tuesday, 19 December 2023, 4:00 PM
Closes: Tuesday, 19 December 2023, 6:00 PM

Preview quiz

To attempt this quiz you need to know the quiz password

Time limit: 1 hour 30 mins

Grading method: Highest grade

This quiz is currently not available.

◄ Java Documentation

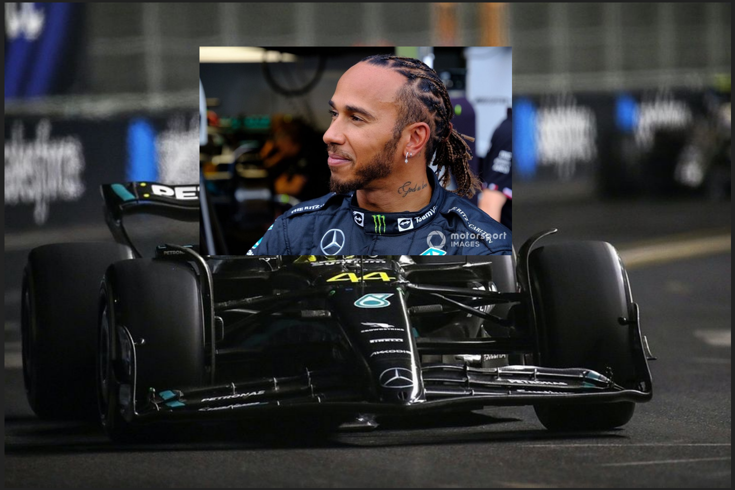
Jump to...

Terminology Recheck

Can a graph be cyclic or a cycle?

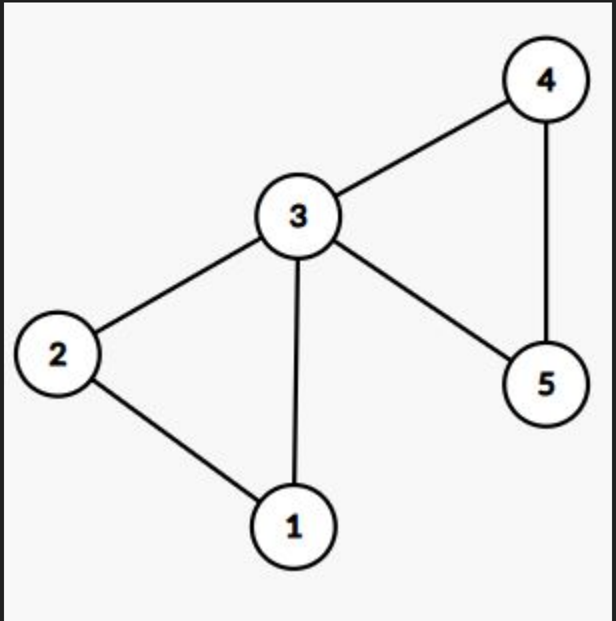
Is it correct

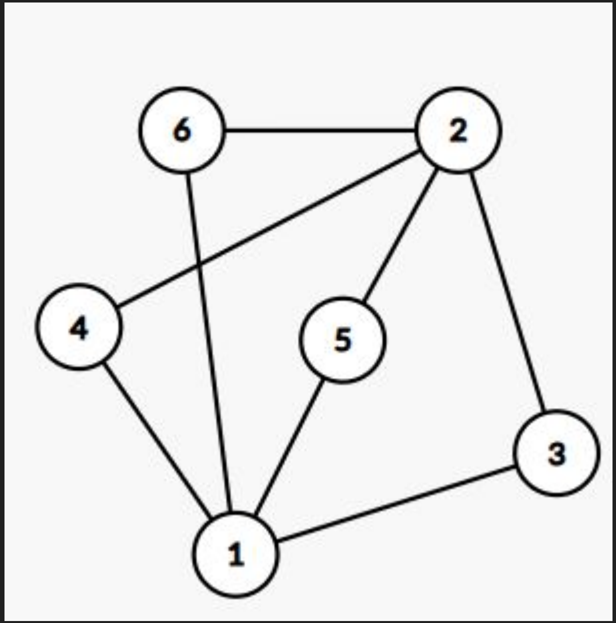
---->



Eulerian ----> Hamiltonian

Eulerian ----> Hamiltonian

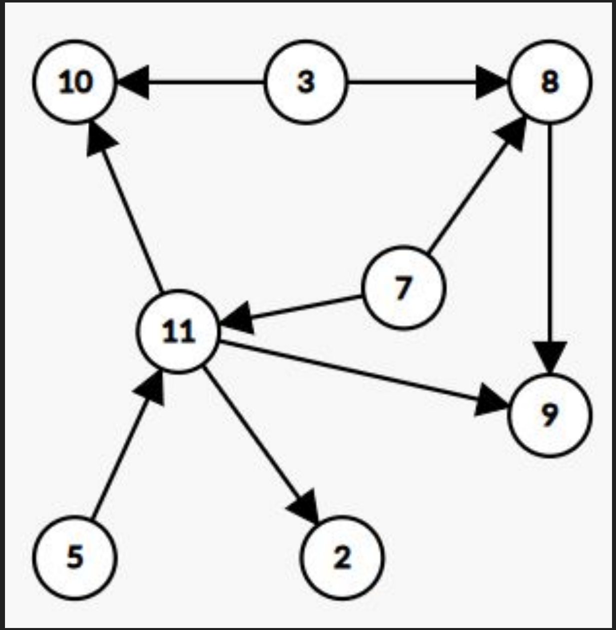




Bipartite GraphK_{2,4}

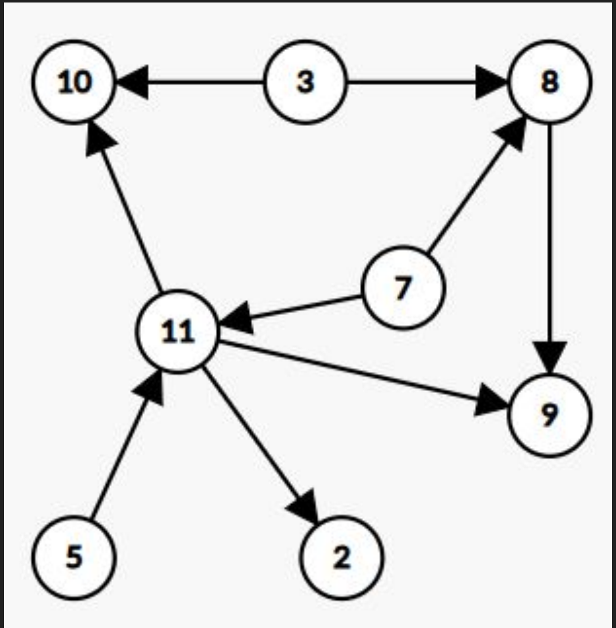
Topological Sorting/Ordering (only in ?)

- ∀ (u,v) in E, u comes before v in the ordering



Topological Sorting/Ordering (only in directed acyclic graphs)

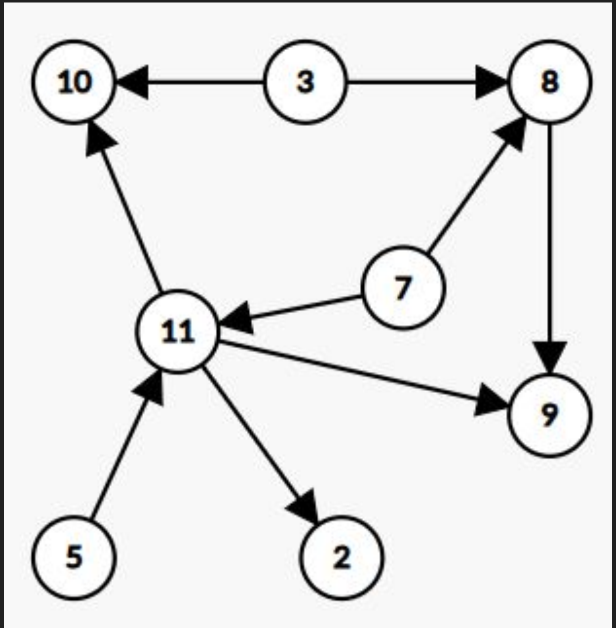
- ∀ (u,v) in E, u comes before v in the ordering



Topological Sorting/Ordering (only in directed acyclic graphs)

∀ (u,v) in E, u comes before v in the ordering

- 5, 7, 3, 11, 8, 2, 9, 10
- 3, 5, 7, 8, 11, 2, 9, 10
- 3, 5, 7, 8, 11, 2, 10, 9
- 5, 7, 3, 8, 11, 2, 10, 9
- 7, 5, 11, 3, 10, 8, 9, 2
- 5, 7, 11, 2, 3, 8, 9, 10
- 3, 7, 8, 5, 11, 10, 2, 9

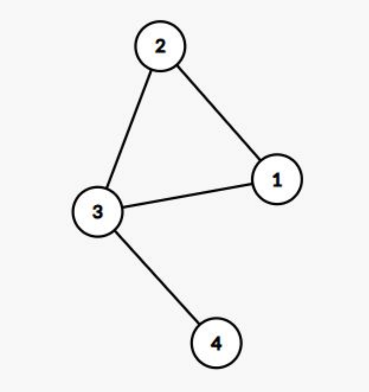


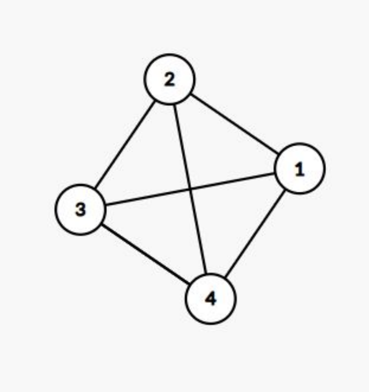
Exercise 9.1 *Transitive graphs.*

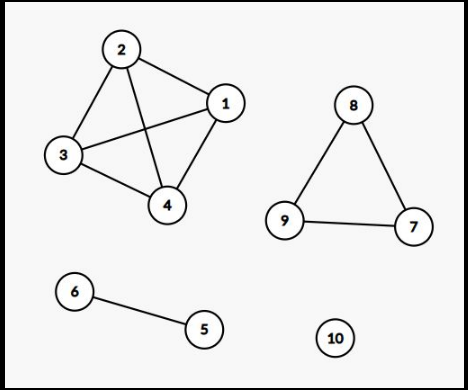
Let $G = (V, E)$ be an undirected graph. We say that G is

- **transitive** when, for any two edges $\{u, v\}$ and $\{v, w\}$ in E, the edge $\{u, w\}$ is also in E;
- **complete** when its set of edges is $\{\{u, v\} \mid u, v \in V, u \neq v\}$;
- the **disjoint sum** of $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ iff $V = V_1 \cup \cdots \cup V_k$, $E = E_1 \cup \cdots \cup E_k$, and the $(V_i)_{1 \leq i \leq k}$ are pairwise disjoint.

Show that a undirected graph G is transitive if, and only if, it is a disjoint sum of complete graphs.







Graphs drawn in https://csacademy.com/app/graph_editor/

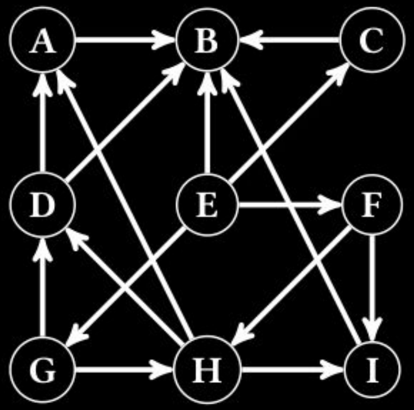
Exercise 9.2 *Short statements about graphs (cont'd)* **(1 point)**.

In the following, let $G = (V, E)$ be a directed graph. For each of the following statements, decide whether the statement is true or false. If the statement is true, provide a proof; if it is false, provide a counterexample.

(a) If for every vertex $v \in V$ its in-degree $\deg_{\text{in}}(v)$ is even, then $|E|$ is even.

(b) For a longest directed path $P : v_0, \dots, v_\ell$ in G, the endpoint has to be a sink.

(c) The following graph has a topological sorting. If so, give a topological sorting; if not, prove why no topological sorting can exist.

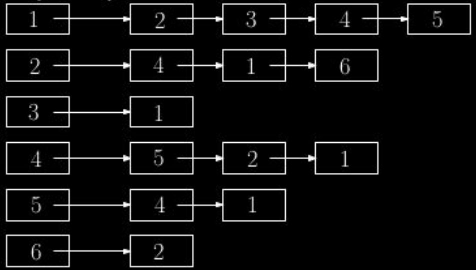


Exercise 9.3 *Data structures for graphs.*

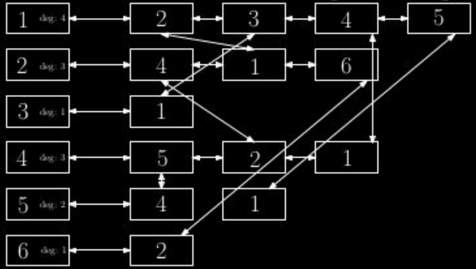
Consider three types of data structures for storing an undirected graph G with n vertices and m edges:

1) Adjacency matrix.

2) Adjacency lists:



3) Adjacency lists, and additionally we store the degree of each node, and there are pointers between the two occurences of each edge. (An edge appears in the adjacency list of each endpoint).



For each of the above data structures, what is the required memory (in Θ-Notation)?

Which runtime (worst case, in Θ-Notation) do we have for the following queries? Give your answer depending on n, m, and/or $\deg(u)$ and $\deg(v)$ (if applicable).

(a) Input: A vertex $v \in V$. Find $\deg(v)$.

(b) Input: A vertex $v \in V$. Find a neighbor of v (if a neighbor exists).

(c) Input: Two vertices $u, v \in V$. Decide whether u and v are adjacent.

(d) Input: Two adjacent vertices $u, v \in V$. Delete the edge $e = \{u, v\}$ from the graph.

(e) Input: A vertex $u \in V$. Find a neighbor $v \in V$ of u and delete the edge $\{u, v\}$ from the graph.

(f) Input: Two vertices $u, v \in V$ with $u \neq v$. Insert an edge $\{u, v\}$ into the graph if it does not exist yet. Otherwise do nothing.

(g) Input: A vertex $v \in V$. Delete v and all incident edges from the graph.

For the last two queries, describe your algorithm.

Exercise 9.4 *Number of paths in DAGs* **(1 point)**.

Let $G = (V, E)$ be a directed graph without directed cycles[1] (i.e., a directed acyclic graph or short DAG). Assume that $V = \{v_1, \ldots, v_n\}$ (for $n = |V| \in \mathbb{N}$). Further assume that v_1 is a source and v_n is a sink. The goal of this exercise is to find the number of paths from v_1 to v_n.

(a) Prove that there exists a topological sorting of G that has v_1 as first and v_n as last vertex.

Using part (a), we assume from now on that the sorting $v_1, v_2, \ldots, v_n$ of the vertices is a topological sorting. We can achieve this by renaming the vertices. Part (a) tells us then that we do not need to rename v_1 and v_n.

(b) Prove that for any directed v_1-v_n-path $P : v_1 = v_{i_0}, v_{i_1}, \ldots, v_{i_\ell} = v_n$ we have $i_0 < i_1 < \cdots < i_\ell$.

(c) Describe a bottom-up dynamic programming algorithm that, given a graph G with the property that $v_1, \ldots, v_n$ is a topological sorting, returns the number of v_1-v_n paths in G in $O(|V| + |E|)$ time. You can assume that the graph is provided to you as a pair (n, Adj) of the integer $n = |V|$ and the adjacency lists Adj. Your algorithm can access $Adj[u]$, which is a list of vertices to which u has a direct edge, in constant time. Formally, $Adj[u] := \{v \in V \mid (u, v) \in E\}$.

Hint: *Define the entry of the DP table as $DP[i]$ = number of paths in G from v_i to v_n.*

(d)* What happens if the vertices v_1 and v_n are not a source respectively a sink? Can we still find the number of v_1-v_n paths using a similar approach as above?

Exercise 9.5 *Strongly connected vertices* **(1 point)**.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. We say two distinct vertices $v, w \in V$ are *strongly connected* if there exists both a directed path from v to w, and from w to v.

Describe an algorithm which find a pair $v, w \in V$ of strongly connected vertices in G, or decides that no such pair exists. The runtime of your algorithm should be at most $O(n + m)$. You are provided with the number of vertices n, and the adjacency list Adj of G.

Hint: *Use DFS to traverse the graph. Maintain a global array* $status[1 \ldots n]$ *which keeps track for each vertex whether (1) it has not been reached yet; (2) it has been reached, but some of its descendants have not; or (3) it, and all of its descendants, have been reached. What should the initial status of each vertex be? What can you say when the DFS reaches a vertex with status (2)? What can you say when all vertices have status (3)?*

Hint: *Make sure your DFS reaches every vertex at some point before terminating!*

Algorithm 1

```
Input: integer n. Adjacency list Adj[1 ... n].

Let status[1 ... n] be a global array, with all entries initialized to UNVISITED.

function visit(u)
    status[u] ← VISITING
    for each v in Adj[u] do                          ▷ Iterate over all neighbours v.
        if status[v] = VISITING then                 ▷ There is a directed cycle containing u and v.
            Output (u, v) and terminate
        if status[v] = UNVISITED then
            visit(v)
    status[u] ← VISITED.
for u = 1, 2, ..., n do
    if status[u] = UNVISITED then
        visit[u]
Output ”no strongly connected vertices exist”
```

Peer Grading

Exercise 9.4

You will find the file to peergrade in your polybox folder

While emailing your peer grading to me please include the group you corrected their work in cc.

https://docs.google.com/spreadsheets/d/1owPsJsd9THBWInwFcVjKCc0f_r6n4pGwwDKMDdwaCjM/edit?usp=sharing