Graphs

What is a graph: \( G = (V, E) \)

\( V = \text{set of vertices (nodes)} \)

**Undirected**: \( E \) is a set of edges

each edge is a pair \( \{u,v\} \), where \( u \in V, v \in V, u \neq v \).

\[
\begin{align*}
V &= \{1,2,3,4\} \\
E &= \{ \{1,2\}, \{1,4\}, \{2,3\}, \{2,4\} \}
\end{align*}
\]

**Directed**: each edge is an ordered pair \( (u,v) \), where \( u \in V, v \in V, u \neq v \) ("one-way street")

\[
\begin{align*}
V &= \{1,2,3,4\} \\
E &= \{ (1,2), (1,4), (2,3), (3,2) \}
\end{align*}
\]
Examples of graphs

* map (= network of roads)
  
* facebook: vertices ⇨ users [end of 2020: 2.7 billion]
  
  edge \{u,v\} ⇨ u and v are friends

* WWW: vertices ⇨ webpages
  
  edge \(u,v\) ⇨ webpage u has a link to webpage v

* scheduling exams

  vertices ⇨ courses taught this term

  edge \{u,v\} ⇨ \exists\ student who takes both course u and course v

  \(\therefore\) exams for u and v cannot be scheduled at the same time

Let \(k\) be the smallest integer such that the following is possible:

1. each vertex gets as label one element of \(\{1, 2, \ldots, k\}\)
2. for each edge \{u,v\}: u and v have different labels
Then we can make an exam schedule with k time slots \( t_1, t_2, \ldots, t_k \):
all vertices (= courses) with label i have their exam in time slot \( t_i \).
In this way there are no conflicts.
Note: Computing k is very difficult!

Undirected graph \( G = (V, E) \).

- degree \( (u) = \# \) edges that contain \( u \)

\[
\sum_{u \in V} \text{degree}(u) = 2|E|.
\]
How to store a graph?

\( G = (V, E) \), \( V = \{v_1, v_2, \ldots, v_n\} \)

* **Adjacency matrix**: \( n \times n \) matrix

  - if \( G \) is undirected:
    
    \[
    \text{entry } (i, j) = \begin{cases} 
    1 & \text{if } \{v_i, v_j\} \text{ is an edge} \\
    0 & \text{otherwise}
    \end{cases}
    \]

    this gives a symmetric matrix

  - if \( G \) is directed:
    
    \[
    \text{entry } (i, j) = \begin{cases} 
    1 & \text{if } (v_i, v_j) \text{ is an edge} \\
    0 & \text{otherwise}
    \end{cases}
    \]

**Advantage**: in \( O(1) \) time, we can test if there is an edge between two given vertices.
Disadvantage:
- uses $\Theta(n^2)$ space for any graph
- find all neighbors of a given vertex takes $\Theta(n)$ time.

* Adjacency lists: each vertex $u$ stores a list.
  - if $G$ is undirected: the list of $u$ stores all neighbors of $u$: all $v$ for which $\{u,v\} \in E$
  - if $G$ is directed: the list of $u$ stores all $v$ for which $(u,v) \in E$ (outgoing edges)

Advantage:
- space = $\Theta(1V + 1E)$
- all neighbors of vertex $u$ can be found in $O(1 + \text{degree}(u))$ time.
Disadvantage: Testing if \(\{u,v\}\) (or \((u,v)\)) is an edge takes \(O(1 + \text{degree}(u))\) time.

For most algorithms: adjacency lists are the best choice.

Exploring an undirected graph \(G = (V, E)\).

Given: vertex \(v\).

Task: find all vertices that can be reached from \(v\).

Algorithm `explore(v)`:

- `visited(v) = true;`
- `previsit(v);` // see later
- for each edge \(\{v, u\} \in E\):
  - if `visited(u) = false`: `explore(u)`
- `postvisit(v)` // see later

Initially, `visited(u) = false` for every vertex \(u\) before the first call to `explore`.
Run explore (A) in the for-loop: use alphabetical order (i.e., adjacency lists are sorted alphabetically). Each time an edge \( \{v,u\} \) is traversed (because \( \text{visited}(u) = \text{false} \)): \( u \) is discovered for the first time;

- draw \( \{v,u\} \) as a solid edge.
- all other edges: dotted.
Solid edges form a tree (connected, no cycles)
these edges are called: tree edges

dotted edges: back edges

Why is algorithm explore(v) correct?

Why does it terminate: number of vertices u with visited(u) = false decreases in each recursive call.