CS 260
Design and Analysis of Algorithms
3. Graphs

Mikhail Moshkov

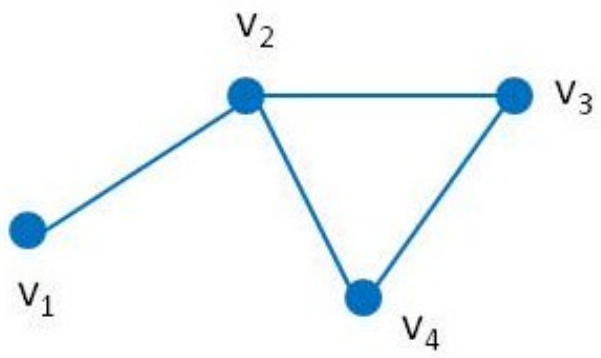
Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

Graphs. Some Definitions

An *undirected graph* G is a pair (V, E) where V is a set of nodes and E is a set of edges. Each edge e is a two-element subset of V: $e = \{u, v\}$. When we work with undirected graphs then sometimes we can use the notation (u, v) instead of $\{u, v\}$.

A *path* in undirected graph is a sequence $v_1, v_2, \ldots, v_{k-1}, v_k$ of nodes such that $\{v_i, v_{i+1}\} \in E$, $i = 1, \ldots, k-1$. This path is called a *cycle* in undirected graph if $k > 3$, the first $k-1$ nodes are pairwise different and $v_1 = v_k$.

Example 3.1. Undirected graph (V, E) where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\}$

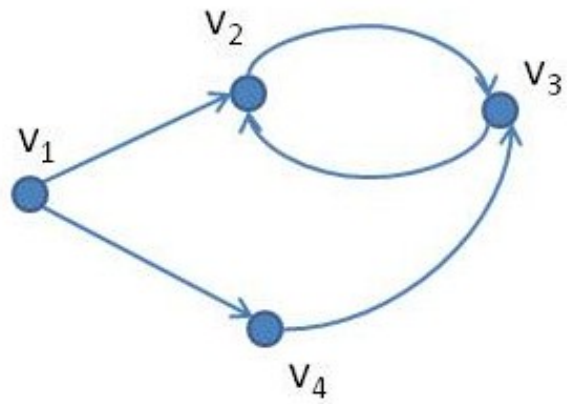


Graphs. Some Definitions

A *directed* graph G is a pair (V, E) where V is a set of nodes and E is a set of directed edges. Each directed edge is an ordered pair (u, v) of nodes.

A *path* in directed graph is a sequence $v_1, v_2, \ldots, v_{k-1}, v_k$ of nodes such that $(v_i, v_{i+1}) \in E$, $i = 1, \ldots, k-1$. This path is called a *cycle* in directed graph if $k > 2$, the first $k-1$ nodes are pairwise different and $v_1 = v_k$.

Example 3.2. Directed graph (V, E) where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_1, v_4), (v_4, v_3), (v_3, v_2)\}$



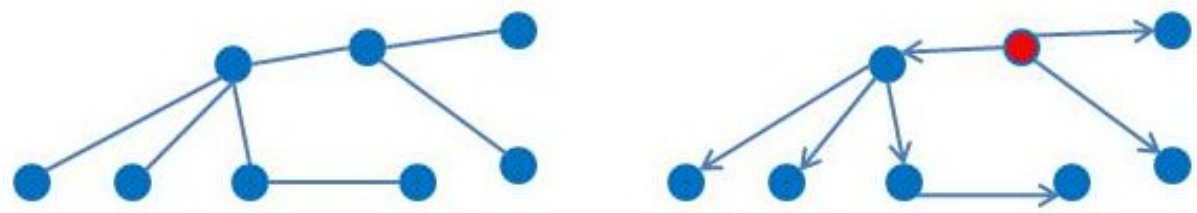
Graphs. Some Definitions

We say that undirected graph is *connected* if for every pair of nodes u and v, there is a path from u to v.

We say that a directed graph is *strongly connected* if, for every two nodes u and v, there is a path from u to v and a path from v to u.

Graphs. Some Definitions

We say that an undirected graph is a *tree*, if it is connected and does not contain a cycle. If we choose one node of a tree as a *root* we obtain a rooted tree in which we "orient" each edge away from the root.



Theorem 3.3. *Let G be an undirected graph with n nodes. Then the following three statements are equivalent:*

(a) *G is connected and does not contain a cycle.*

(b) *G is connected and has $n-1$ edges.*

(c) *G has $n-1$ edges and does not contain a cycle.*

Graphs. Some Definitions

To represent a graph we will use an adjacency list representation. For each node v of undirected graph we have a list of nodes u such that $\{u, v\} \in E$.

For each node v of directed graph, sometimes it is useful to have two lists: the first one contains all nodes u such $(u, v) \in E$, and the second one contains all nodes u such that $(v, u) \in E$.

Breadth-First Search (Undirected Graphs)

Let s be a node of an undirected graph $G = (V, E)$. We now describe an algorithm (BFS) which allows us to find in G all nodes t such that there is a path from s to t.

The considered algorithm constructs so called breadth-first search tree T for the graph G, all nodes of which are divided onto layers.

Breadth-First Search (Undirected Graphs)

The layer L_0 contains the only node s.

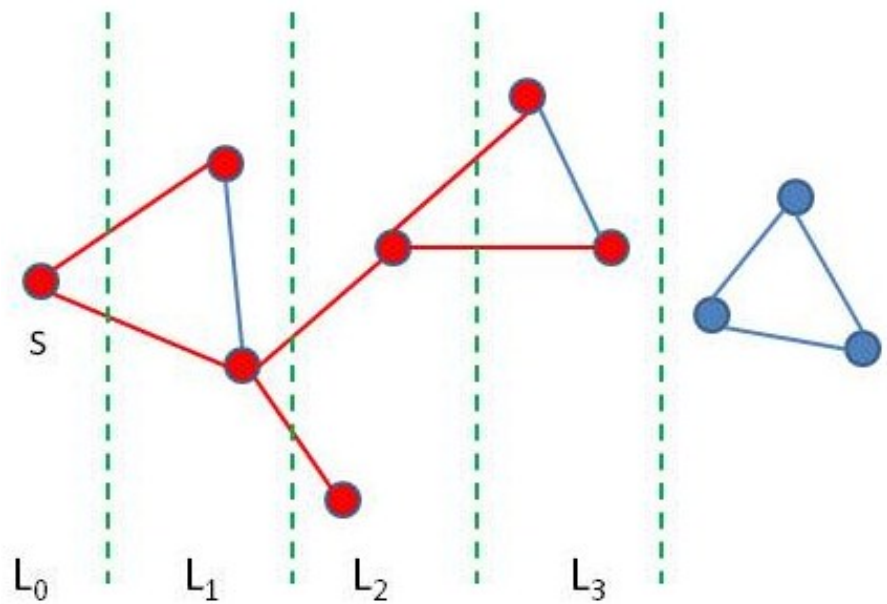
Let we have defined layers $L_0, L_1, \ldots, L_i$.

The layer L_{i+1} consists of all nodes v that do not belong to layers $L_0, L_1, \ldots, L_i$ and have an edge $\{u, v\}$ for a node u from L_i. We add each such node v and corresponding edge $\{u, v\}$ to the constructed tree T.

It is clear that T contains all nodes t from G such that there exists a path from s to t, and only such nodes.

Breadth-First Search (Undirected Graphs)

Example 3.4. Breadth-first search in undirected graph



Breadth-First Search (Undirected Graphs)

Let $|V| = n$ and $|E| = m$. To implement the considered algorithm we will use lists $L[0], L[1], \ldots$ corresponding to layers $L_0, L_1, \ldots$ and array *Discovered* of length n such that $Discovered[v] = true$ if and only if v is already added to T.

Each node belongs to at most one list. Therefore we need $O(n)$ time to set up lists and manage the array *Discovery*.

Breadth-First Search (Undirected Graphs)

Let $\{u, v\} \in E$. The algorithm will work with this edge at most two times: when the algorithm studies the node u, and when the algorithm studies the node v. In the first case we should recognize if $Discovery[v] = false$, and in the second case we should recognize if $Discovery[u] = false$.

Each such step requires $O(1)$ time. So the total time spent considering edges is $O(m)$, and the total time is $O(n + m)$.

Breadth-First Search. Connected Components

We can use BFS to find all connected components of the undirected graph G.

A *connected component* of G is a maximal subgraph of G which is connected.

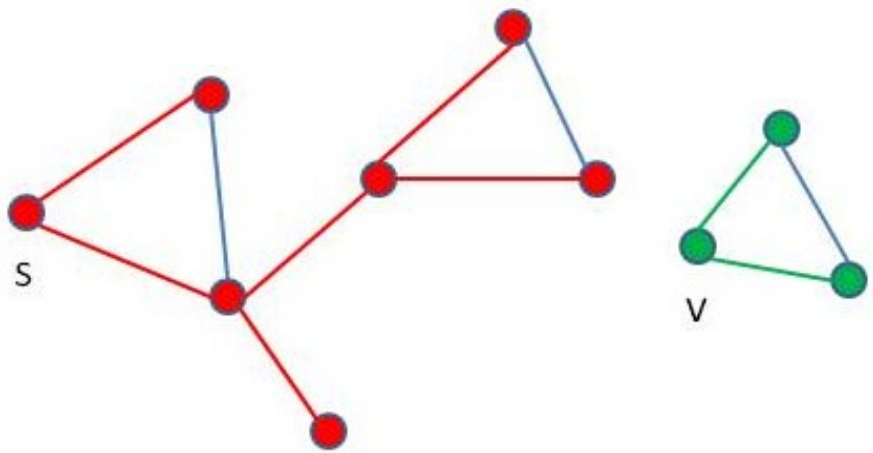
One can show that for any two nodes s and t connected components containing s and t respectively are either identical or disjoint.

Breadth-First Search. Connected Components

We start with an arbitrary node s and use BFS to construct a tree T with the root s such that the set of nodes of T coincides with the set of nodes of connected component containing s.

We then find a node v (if any) that was not visited by the search from s, etc. We continue in this way until all nodes will be visited.

Example 3.5. Connected components of undirected graph



Breadth-First Search (Directed Graphs)

Breadth-first search is almost the same in directed graphs as it is in undirected graphs.

Let $G = (V, E)$ be a directed graph and $s \in V$. The layer L_0 contains the only node s.

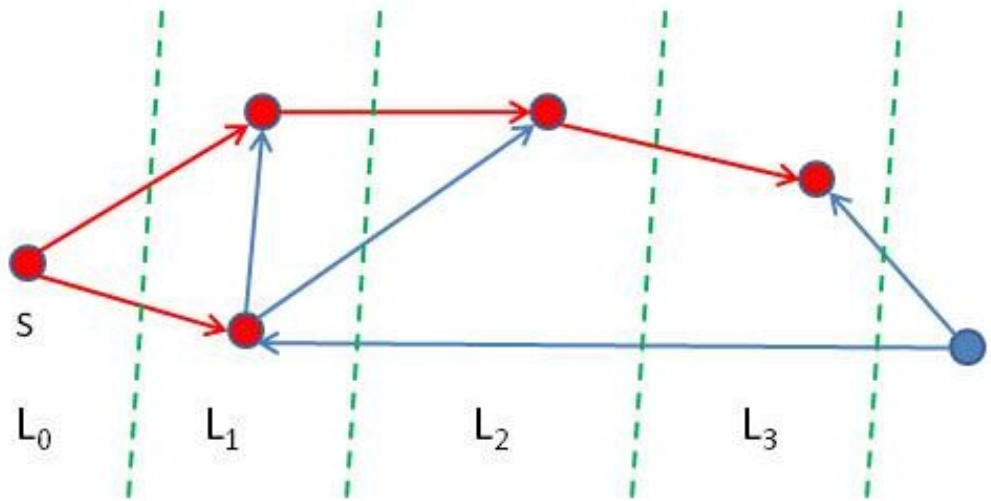
If we have constructed layers $L_0, \ldots, L_i$, then the layer L_{i+1} consists of all nodes v that do not belong to $L_0, \ldots, L_i$, and have an edge (u, v) for a node u from L_i.

Breadth-First Search (Directed Graphs)

One can show that the constructed directed tree T with the root s contains all nodes t from G such that there exists a directed path from s to t, and only such nodes.

One can show also that the running time of this algorithm is $O(m+n)$ where $m = |E|$ and $n = |V|$.

Example 3.6. Breadth-first search in directed graph



Breadth-First Search. Strongly Connected Components

We can use BFS to find all strongly connected components of the directed graph G.

A *strongly connected component* is a maximal subgraph of G which is strongly connected.

One can show that for any two nodes s and t, strongly connected components containing s and t respectively are either disjoint or identical.

Breadth-First Search. Strongly Connected Components

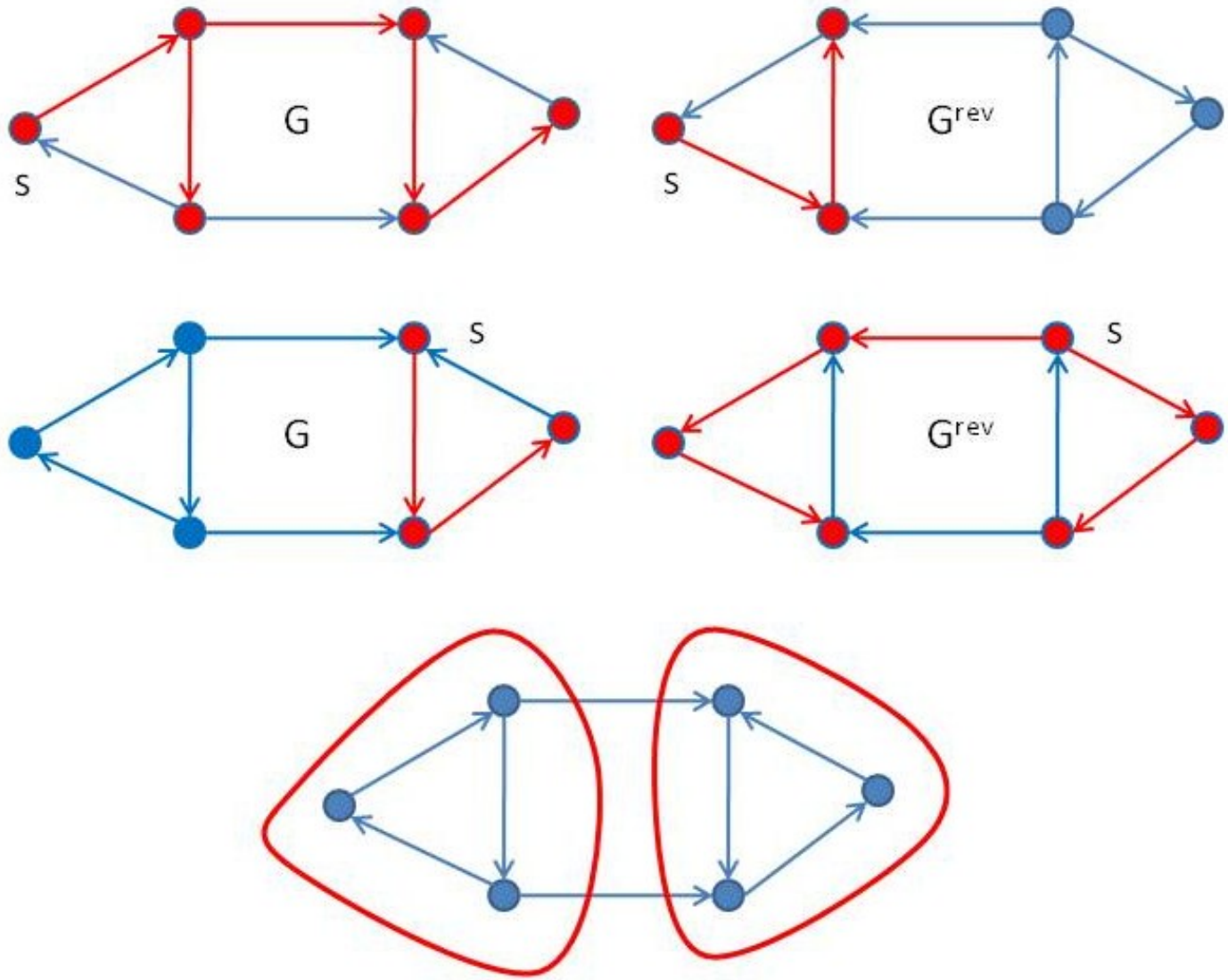
We denote be G^{rev} the directed graph which is obtained from G by reversing the direction of every edge. Let $s, t \in V$. It is clear that there is a path from s to t in G^{rev} if and only if there is a path from t to s in G.

We run BFS starting from s both in G and G^{rev}. As a result we obtain two trees T_1 and T_2 with the set of nodes V_1 and V_2. It is clear that the set of nodes in the strongly connected component containing s is equal to $V_1 \cap V_2$.

If $V_1 \cap V_2 \neq V$ then we choose a node $v \in V \setminus (V_1 \cap V_2)$ and repeat the same operation with the node v, etc.

Breadth-First Search. Strongly Connected Components

Example 3.7. Directed graph and its strongly connected components



Directed Acyclic Graphs and Topological Ordering

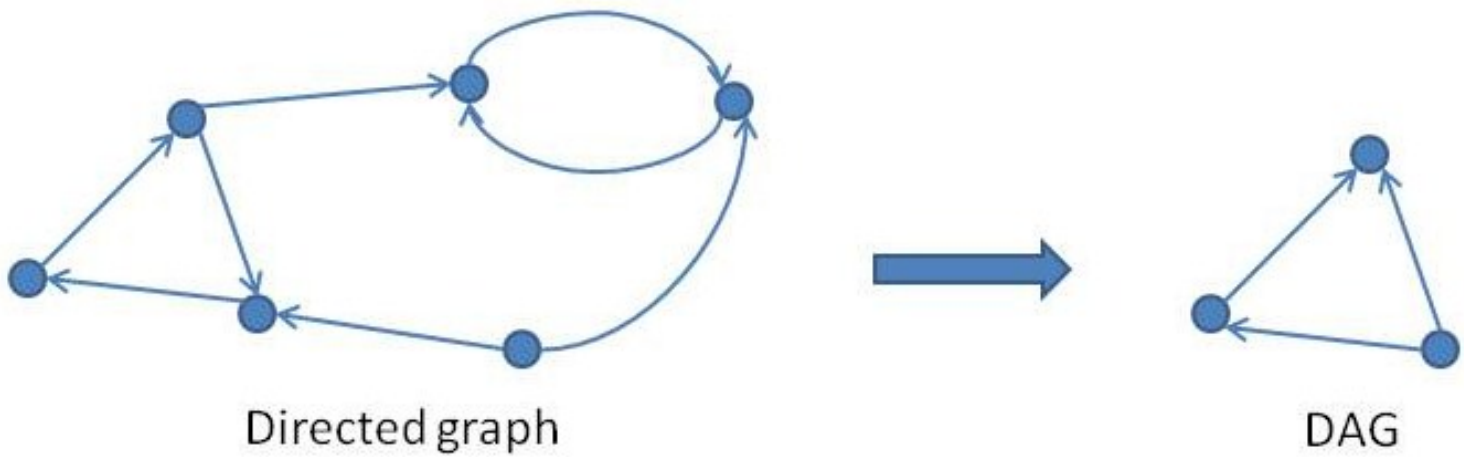
A *directed acyclic graph* (DAG for short) is a directed graph without cycles.

Let G be a directed graph. We "join" each strongly connected component of G into a single meta-node, and draw an edge from one meta-node to another if there is an edge (in the same direction) between components of the considered meta-nodes.

The resulting meta-graph is a DAG: a cycle containing several strongly connected components would "join" these components into a single strongly connected component.

Directed Acyclic Graphs and Topological Ordering

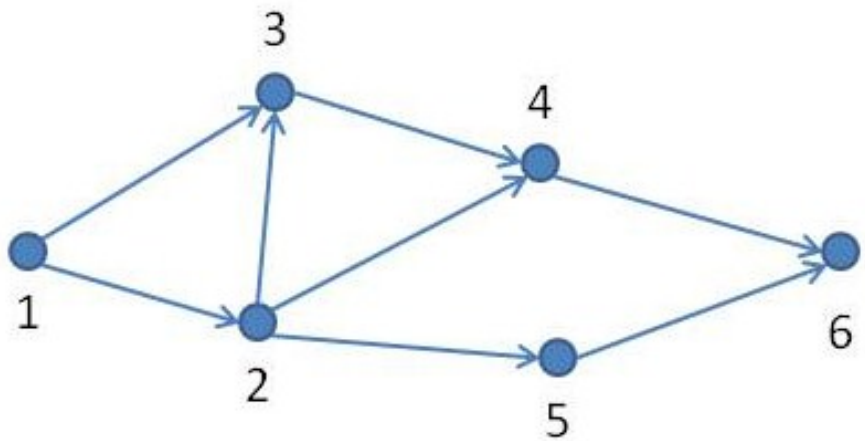
Example 3.8. Directed graph and corresponding meta-graph which is a DAG



Directed Acyclic Graphs and Topological Ordering

For a directed graph G, we say that a *topological ordering* of G is an ordering of its nodes as $v_1, \ldots, v_n$ so that for every edge (v_i, v_j) of G we have $i < j$. If G contains a cycle, then, obviously, G has no a topological ordering.

Example 3.9. Topological ordering of directed acyclic graph (we write i instead of v_i)



Directed Acyclic Graphs and Topological Ordering

Let now $G = (V, E)$ be a DAG. We show that G has a topological ordering.

Since G has no cycles, there exists a directed path in G of maximum length (length here is the number of edges in the path). Let v be the first node in this path. It is clear that v has no incoming edges. This node will be considered as the first node v_1 in a topological ordering.

We remove v_1 from G. The obtained graph G' is also a DAG. In G' there exists a node u without incoming edges. This node will be considered as the second node v_2, etc.

Directed Acyclic Graphs and Topological Ordering

To implement the considered algorithm efficiently, we consider the notion of *active* node – a node which was not deleted by the algorithm. We maintain two things:

(a) For each node w, the number of incoming edges that w has from active nodes.

(b) The set S of all active nodes in G that have no incoming edges from other active nodes.

Directed Acyclic Graphs and Topological Ordering

At the start, all nodes are active, so we can initialize (a) and (b) with a single path through nodes and edges.

Then each iteration consists of selecting a node v from the set S and deleting it. After deleting v, we consider all edges of the kind (v, w) and subtract one from the number of active incoming edges for w. If the obtained number is equal to 0 then we will add w to S.

During the iterations each edge will be considered at most one time. Using this fact one can show that the overall time of the algorithm is $O(n + m)$ where $m = |E|$ and $n = |V|$.

Depth-First Search

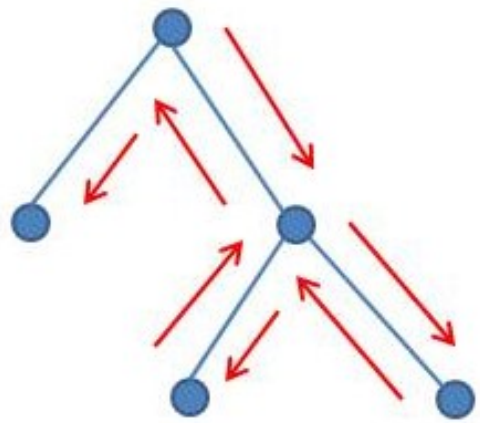
We concentrate in this section on the consideration of BFS. However, there exists another natural way to find nodes reachable from s.

We start from s and try the first edge leading out of s, to a node v. We then follow the first edge leading out of v, and continue in this way until we reach a "dead-end" – a node for which we already explored all neighbors. We then backtrack until we meet a node with a unexplored neighbor, etc.

This algorithm is called *depth-first search* (DFS).

Depth-First Search

Example 3.10. Depth-first search



Planar Graphs

Planar graphs appear naturally in many applications, for example, in the design of processors.

Many NP-hard graph problems such as the clique problem, the bipartite subgraph problem, the maximal cut problem can be solved in polynomial time in the case of planar graphs.

We discuss the notion of planar graph and consider some mathematical and algorithmic problems connected with planar graphs.

Connected Graphs

We consider only finite undirected graphs without loops and multiple edges (simple graphs).

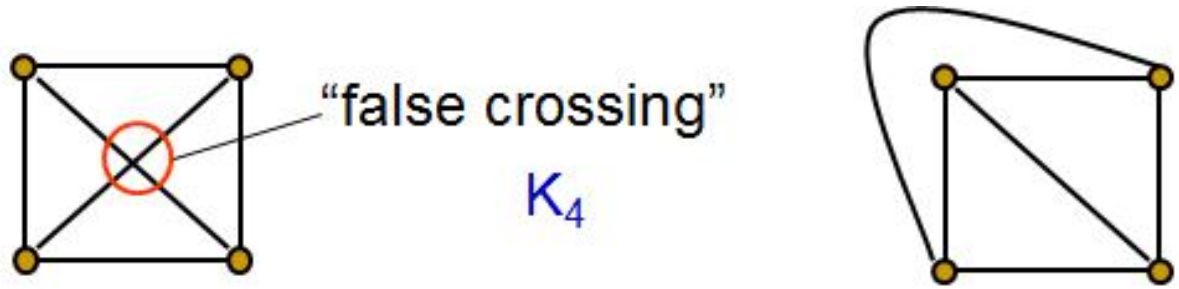
Lemma 3.11. *Let $G = (V, E)$ be a connected graph with p vertices and q edges. Then $q \geq p - 1$.*

Planar Embedding of a Graph

Let $G = (V, E)$ be a graph. We will say that G is *planar* if it is possible to draw it on the plane (vertices are points and edges are continuous non-self-intersecting curves) such that edges are intersected only in vertices.

Such representation of G is called a *planar embedding* of G.

Planar Embedding of a Graph



Let us consider a planar embedding of a planar graph G. Edges of G partition the plane into connected regions. The closures of these regions are called *faces* of G.

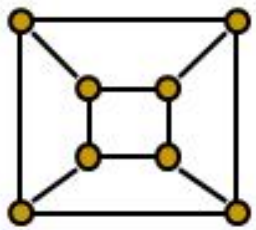
Note that exactly one face is unbounded. This face is called the *external* face.

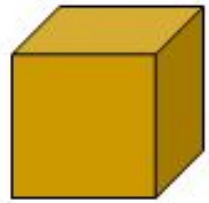
Euler's Formula

Theorem 3.12. *For any planar embedding of a connected planar graph $G = (V, E)$ with p vertices, q edges and r faces the following equality holds:*

$$p - q + r = 2.$$

Euler's Formula

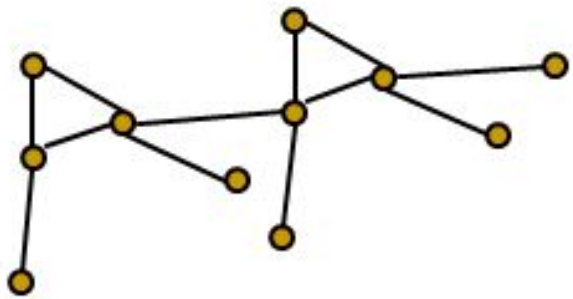




$p = 8, q = 12, r = 6$
$p - q + r = 8 - 12 + 6 = 2$

Leonhard Euler have found the considered formula for the graphs corresponding to polyhedrons.

Euler's Formula



Example 3.13. For the considered graph, $p = 11, q = 12, r = 3$. We have $11 - 12 + 3 = 2$.

Euler's Formula

Proof: We fix p and prove the considered statement by induction on q. From Lemma 3.11 it follows that $q \geq p - 1$.

a). Let $q = p - 1$. By Theorem 3.3, G is a tree. Hence G has no cycles. Therefore $r = 1$ and $p - q + r = p - (p - 1) + 1 = 2$.

Euler's Formula

b). Let $q_0 > p - 1$ and, for each q, $q_0 > q \geq p - 1$, the considered equality hold.

Let us consider a planar embedding of a connected planar graph $G = (V, E)$ with p vertices and q_0 edges. Let r be the number of faces in this embedding.

Since $q_0 > p - 1$, G is not a tree by Theorem 3.3. Therefore G has a cycle. Let e be an edge belonging to this cycle. This edge is a part of the boundary that separates two faces.

Euler's Formula

Let us remove the edge e from the graph G. As a result we obtain a connected planar graph G' in which the considered two faces are joined into one face.

We obtain also a planar embedding of the graph G' with p vertices, $q_0 - 1$ edges, and $r - 1$ faces. By inductive hypothesis, $p - (q_0 - 1) + (r - 1) = 2$ and $p - q_0 + r = 2$. □

Euler's Formula

Corollary 3.14. *Let G be a connected planar graph with p vertices and q edges. Then all planar embeddings of G contain the same number of faces $r = 2 - p + q$.*

Corollary 3.15. *Let G be a connected planar graph with $p \geq 3$ vertices and q edges. Then $q \leq 3p - 6$.*

Graph K_5

Proposition 3.16. *Graph K_5 is not planar.*

Proof: Let us assume that K_5 has a planar embedding. Since K_5 is connected, for the considered embedding the Euler's formula holds: $p - q + r = 2$.

Since $p = 5$ and $q = 10$, we have $r = 2 - p + q = 7$.

For $i = 1, \ldots, 7$, we denote by g_i the number of edges belonging to the i-th face.

Graph K_5

On can show that each edge belongs to at most two faces. Therefore

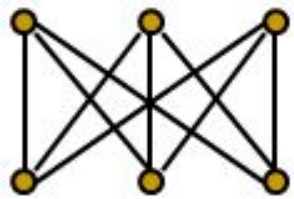
$$\sum_{i=1}^{7} g_i \leq 2q = 20.$$

It is clear that each face contains at least 3 edges (since there are no loops and multiple edges). Hence $g_i \geq 3$ for $i = 1, \ldots, 7$ and

$$\sum_{i=1}^{7} g_i \geq 21$$

which is impossible.

Graph $K_{3,3}$

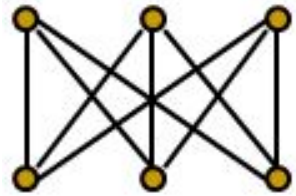


Proposition 3.17. *Graph $K_{3,3}$ is not planar.*

Proof: Let us assume that $K_{3,3}$ has a planar embedding. Since $K_{3,3}$ is connected, for the considered embedding the Euler's formula holds: $p - q + r = 2$.

Since $p = 6$ and $q = 9$, we have $r = 2 - p + q = 5$.

Graph $K_{3,3}$



As in the proof of Proposition 3.16 we have

$$\sum_{i=1}^{5} g_i \leq 2q = 18,$$

where g_i is the number of edges belonging to the i-th face, $i = 1, \ldots, 5$.

It is clear that $K_{3,3}$ has no cycles of the lengths 1, 2 or 3. Hence $g_i \geq 4$ for $i = 1, \ldots, 5$ and

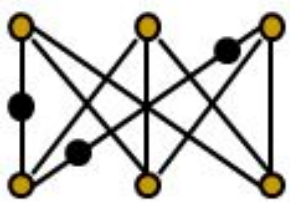
$$\sum_{i=1}^{5} g_i \geq 20,$$

which is impossible.

Subdivision of a Graph

Subdivision of an edge:

A graph H is a *subdivision* of a graph G, if one can obtain H from G by a series of edge subdivisions.



Subdivisions of $K_{3,3}$ and K_5

Kuratowski's Theorem

Theorem 3.18. (Kuratowski, 1930). *A graph G is planar if and only if G does not contain a subdivision of K_5 or $K_{3,3}$ as a subgraph.*

We will prove only a part of this statement:

Let G be a planar graph. Then G does not contain a subdivision of K_5 or $K_{3,3}$ as a subgraph.

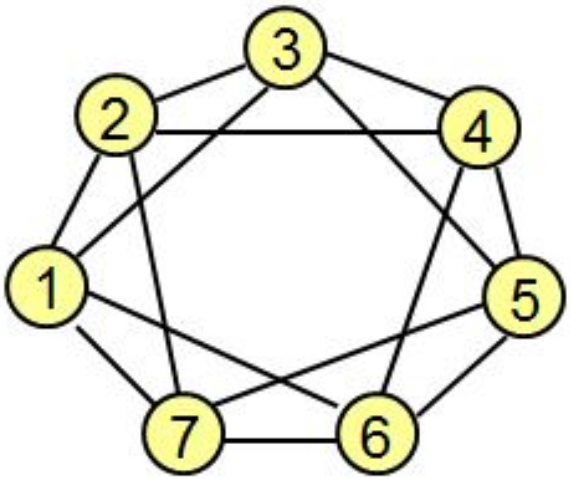
Kuratowski's Theorem

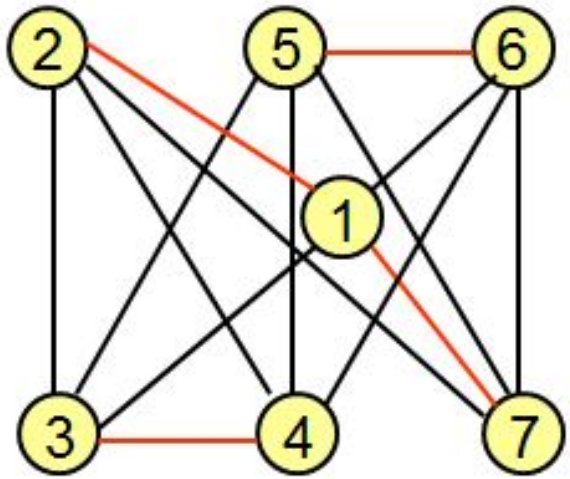
Proof: Let as assume the contrary: G contains a subgraph H which is a subdivision of K_5 or $K_{3,3}$.

We now consider a planar embedding of G. If we remove from this embedding all vertices and edges which do not belong to H we obtain a planar embedding of H.

From the geometrical point of view, H is K_5 or $K_{3,3}$ with additional points on edges. So we have a planar embedding of K_5 or $K_{3,3}$ which is impossible by Propositions 3.16 and 3.17.

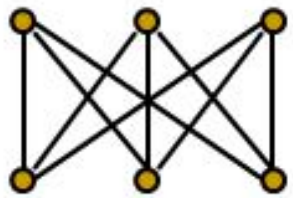
Example of Nonplanar Graph

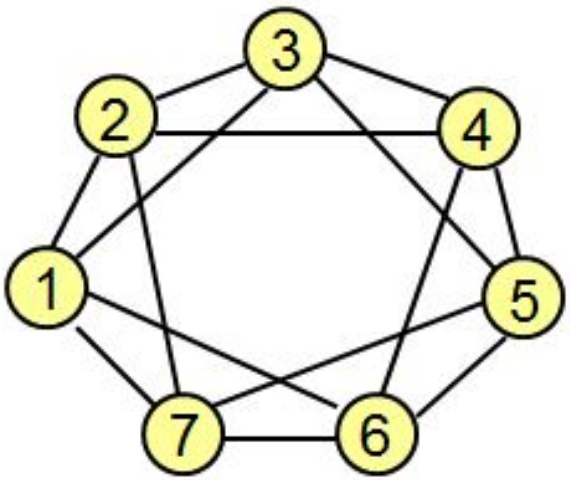


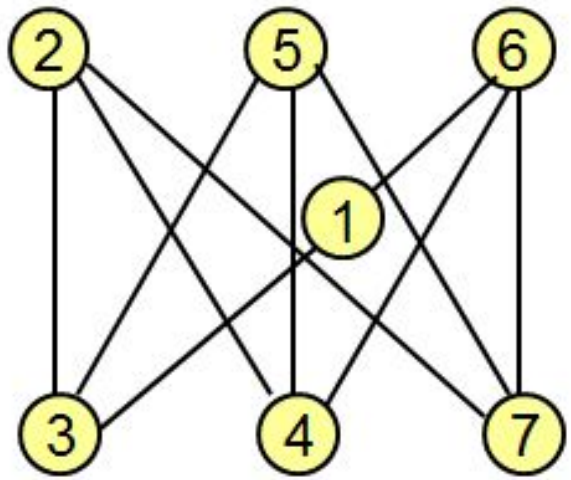


Example of Nonplanar Graph

$K_{3,3}$

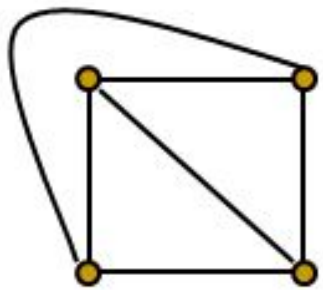


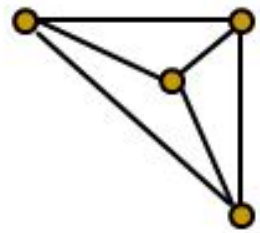




Fáry's Theorem

Theorem 3.19. (Fáry, 1948). *Any planar graph has a planar embedding in which each edge is a straight line segment.*





Planarity Problem

The planarity problem is the following: given a graph G, test if G has a planar embedding and, if so, construct such an embedding.

Algorithms for Planarity Problem

Kuratowski, 1930 – exponential time algorithm

Tutte, 1963 – polynomial time algorithm

Hopcroft and Tarjan, 1974 – linear time algorithm

Algorithms for Planarity Problem

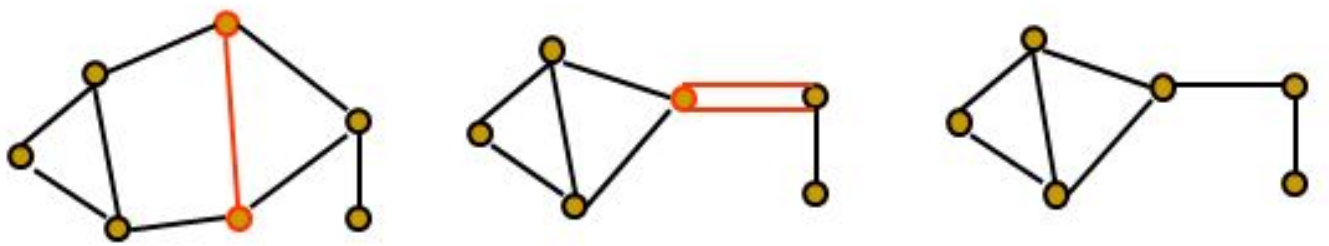
Ramachandran and Reif, 1994 – logarithmic time CRCW PRAM algorithm

CRCW PRAM – Concurrent Read Concurrent Write Parallel Random Access Machine

Minor of a Graph

A graph H is a *minor* of the graph G if it can be obtained from G by contracting edges, removing edges, and removing isolated vertices.

Edge contracting:

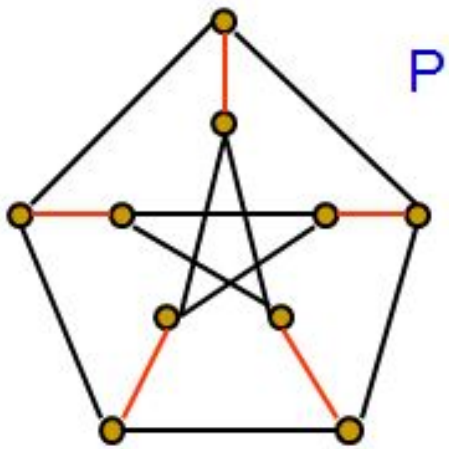


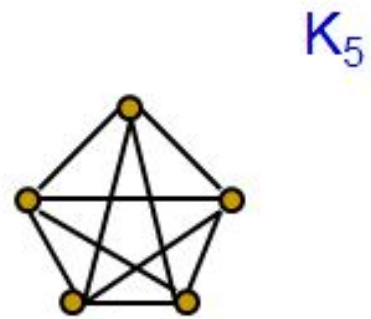
Wagner's Theorem

Theorem 3.20. (Wagner, 1937). *A graph G is planar if and only if it does not have K_5 or $K_{3,3}$ as a minor.*

K_5 and $K_{3,3}$ are forbidden minors for the class of planar graphs.

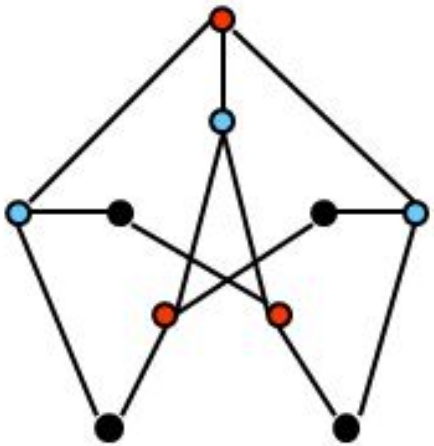
The Petersen Graph





P does not contain a K_5-subdivision, but K_5 is a minor of the Petersen graph P.

The Petersen Graph



$K_{3,3}$-subdivision in the Petersen graph P.

Minor-Closed Sets of Graphs

A set of graphs S is called a *minor-closed* set if S is closed under taking minors: if $G \in S$ then S contains all minors of G.

Examples of minor-closed sets of graphs:

The set of all forests.

The set of all planar graphs.

The set of all graphs that can be embedded without edge intersections in a torus.

The set of all graphs that can be embedded without edge intersections in a fixed arbitrary surface S.

Minor-Closed Sets of Graphs

Let us fix a graph H. Robertson and Seymour proved that there is a polynomial algorithm (with time complexity $O(n^3)$) which for a given graph G checks if G has H as a minor.

Robertson-Seymour Theorem

Theorem 3.21. (Robertson and Seymour, 1985-2005). *Every minor-closed set of graphs has a finite set of forbidden minors.*

Proof: more than 500 pages in 20 papers

As a result, membership of a graph to a fixed minor-closed set of graphs can be checked by a polynomial algorithm (with time complexity $O(n^3)$).

To use this algorithm we should know the set of forbidden minors.