Lecture 5
Algorithms on graphs.
Introduction to graphs and basic algorithms on graphs

Analysis and Development of Algorithms

УНИВЕРСИТЕТ ИТМО

Overview

1. Graphs and their applications
2. Trees and forests
3. Graph representations
4. Real world graphs
5. Depth-first search and its applications
6. Breadth-first search and its applications

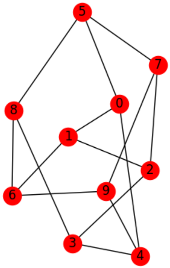
An (*undirected*) *graph* is a pair $G = (V, E)$, where V is a set whose elements are called *vertices* (or *nodes*), and E is a set of two-sets (sets with two distinct elements) of vertices, whose elements are called *edges* (or *links*).

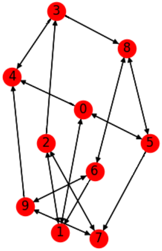
The number of vertices is usually denoted by $|V|$. The number of edges is usually denoted by $|E|$.

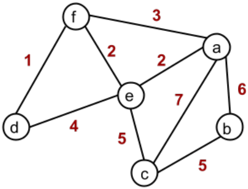
A *directed graph* is a graph in which edges have orientations.

A *weighted graph* is a graph in which a *weight* is assigned to each edge.

A *simple graph* allows only one edge between a pair of vertices. A *multigraph* is a generalization that allows multiple edges between a pair of vertices.







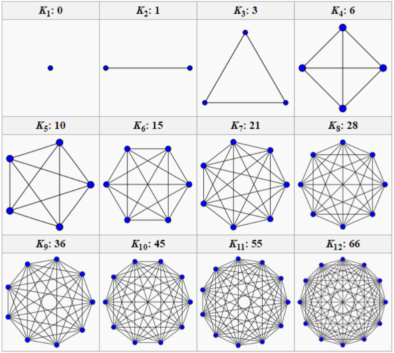
Graphs and their applications

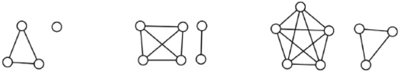
A *complete graph* is a graph in which each pair of vertices is joined by an edge.

A *path* in a graph is a sequence of distinct edges which join a sequence of distinct vertices. The *length* of a path is the sum of the weights of the traversed edges.

A pair of vertices (v_1, v_2) is called *connected* if there is a path from v_1 to v_2. Otherwise, a pair of vertices is called *disconnected*.

A *connected* graph is one in which every pair of vertices is connected. Otherwise, it is a *disconnected* one.



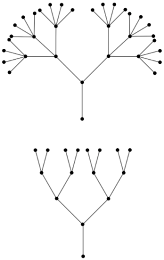


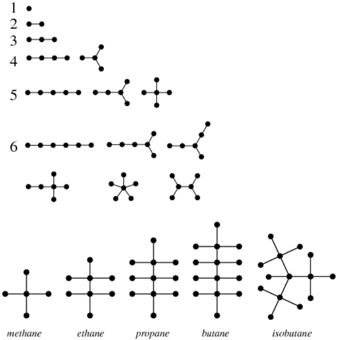
We will usually consider **simple undirected (unweighted or weighted) graphs**.

Question: What is a walk and a trail? Compare with a path.

Demonstration: Applications of graphs in real life

A *tree* is an undirected graph in which any two vertices are connected by exactly one path.

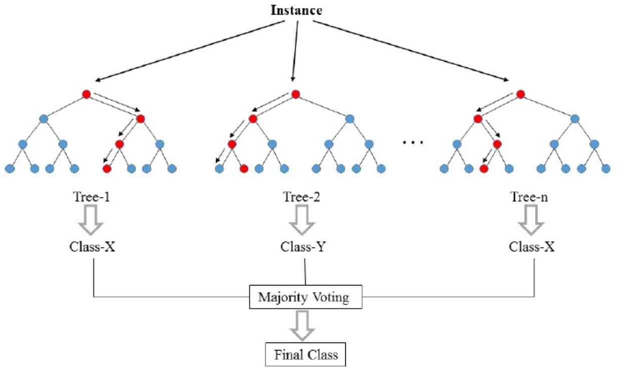




Demonstration: Decision tree, Fractal tree

Special types of graphs: forests

A *forest* is an undirected graph in which any two vertices are connected by at most one path, or equivalently a disjoint union of trees.



Demonstration: Random forest classifier

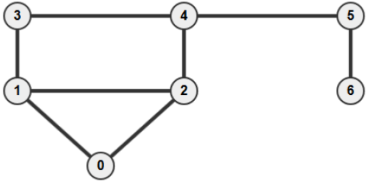
Graph representations: adjacency matrix and adjacency list

The *adjacency matrix* is a matrix whose rows and columns are indexed by vertices and whose cells contain a Boolean value that indicates whether the corresponding vertices are adjacent (for weighted graphs, it contains corresponding weights instead of 1s.). The matrix (stored as a 2D array) requires $O(|V|^2)$ of space.

The *adjacency list* is a collection of lists containing the set of adjacent vertices of a vertex. The list (stored as an 1D array of lists) requires $O(|V| + |E|)$ of space.

For a *sparse graph*, i.e. a graph in which most pairs of vertices are not connected by edges, $|E| \ll |V|^2$, an adjacency list is significantly more space-efficient than an adjacency matrix.

Graph



Adjacency matrix

	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0
2	1	1	0	0	1	0	0
3	0	1	0	0	1	0	0
4	0	0	1	1	0	1	0
5	0	0	0	0	1	0	1
6	0	0	0	0	0	1	0

Adjacency list

Vertex	Adjacent vertices
0	1, 2
1	0, 2, 3
2	0, 1, 4
3	1, 4
4	2, 3, 5
5	4, 6
6	5

Real world graphs (networks)

Link 1: Stanford Large Network Dataset Collection
Link 2: Network Repository

Data & Network Collections. Find and interactively VISUALIZE and EXPLORE hundreds of network data

ANIMAL SOCIAL NETWORKS	816	INTERACTION NETWORKS	29	SCIENTIFIC COMPUTING	11
BIOLOGICAL NETWORKS	37	INFRASTRUCTURE NETWORKS	8	SOCIAL NETWORKS	77
BRAIN NETWORKS	116	LABELED NETWORKS	104	FACEBOOK NETWORKS	114
COLLABORATION NETWORKS	19	MASSIVE NETWORK DATA	21	TECHNOLOGICAL NETWORKS	12
CHEMINFORMATICS	646	MISCELLANEOUS NETWORKS	2668	WEB GRAPHS	33
CITATION NETWORKS	4	POWER NETWORKS	8	DYNAMIC NETWORKS	115
ECOLOGY NETWORKS	6	PROXIMITY NETWORKS	13	TEMPORAL REACHABILITY	38
ECONOMIC NETWORKS	16	GENERATED GRAPHS	221	BHOSLIB	36
EMAIL NETWORKS	6	RECOMMENDATION NETWORKS	36	DIMACS	78
GRAPH 500	8	ROAD NETWORKS	15	DIMACS10	84
HETEROGENEOUS NETWORKS	15	RETWEET NETWORKS	34	NON-RELATIONAL ML DATA	211

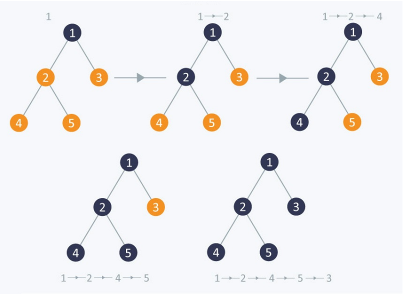
NB: For consistency, I recommend to distinguish (graph, vertex, edge) and (network, node, link).

Depth-first search (DFS) and its applications

Depth-first search (DFS) is an algorithm for traversing a graph. The algorithm starts at a chosen root vertex and explores as far as possible along each branch before backtracking.

Applied for: searching connected components, searching loops in a graph, testing bipartiteness, topological sorting, etc.

The **time complexity** of DFS is $O(|V|+|E|)$.



A *connected component* of a graph is a subgraph in which any two vertices are connected by paths, and which is not connected to any vertex in the rest graph.

Demonstration: DFS and Search of connected components

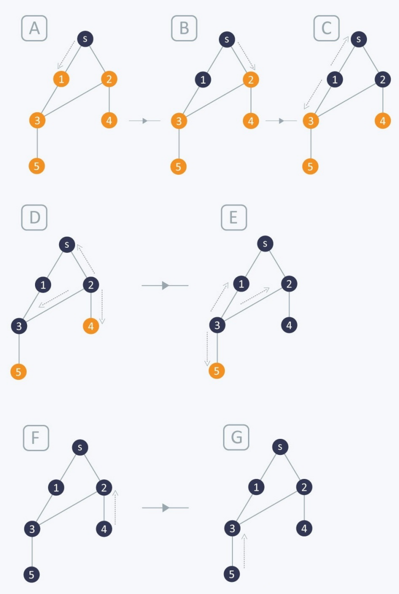
Breadth-first search (BFS) is an algorithm for traversing or searching a graph. The algorithm starts at a chosen root vertex and explores all of the neighbour vertices at the present depth prior to moving on to the vertices at the next depth level.

It uses an **opposite strategy to DFS**, which instead explores the vertex branch as far as possible before being forced to backtrack and expand other vertices.

Applied for: searching shortest path

The **time complexity** of BFS is $O(|V|+|E|)$.

Demonstration: BFS



Active Learning

Why the time complexity of both algorithms is $O(|V| + |E|)$?

Thank you for your attention!