Introduction to graphs

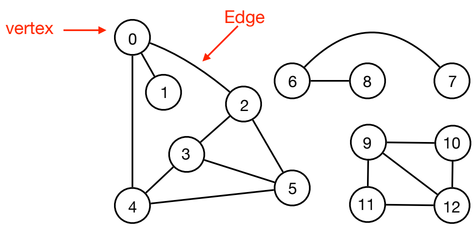
- Undirected graphs
- Representation
- Depth first search
  - Connected components
- Breadth first search
  - Bipartite graphs

Introduction to graphs

- Undirected graphs
- Representing graphs
- Depth first search
  - Connected components
- Breadth first search
  - Bipartite graphs

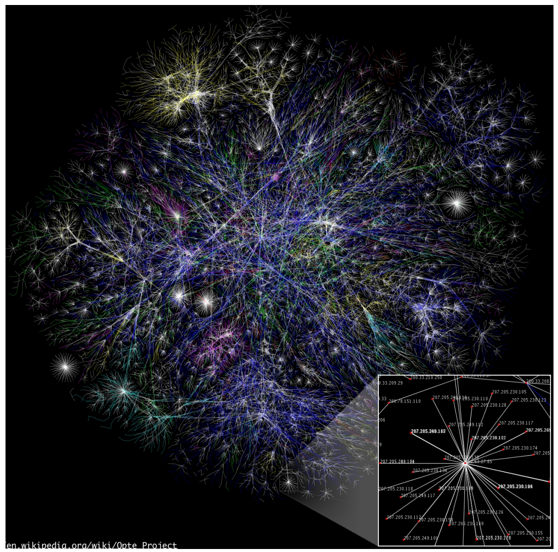
Undirected graphs

- Undirected graph. Set of vertices *(da: knuder)* pairwise joined by edges *(da: kanter)*.



- Why graphs?
  - Models many problems from different areas.
  - Thousands of practical applications.
  - Hundreds of well-known graph algorithms.

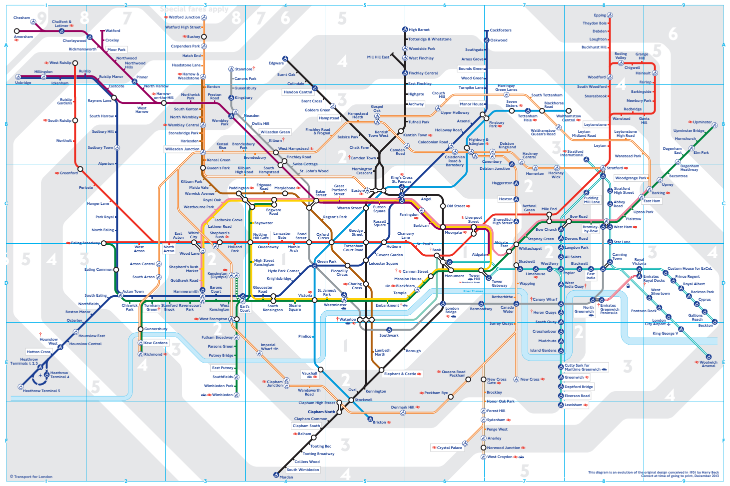
Visualisation of the internet



Visualisation of friendships on Facebook

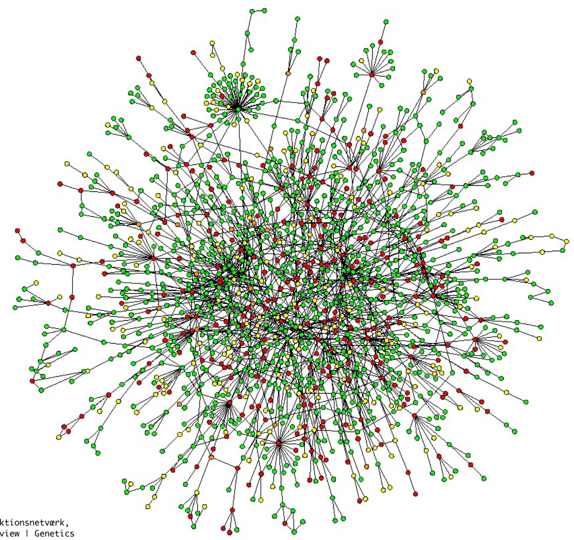
"Visualizing friendships", Paul Butler

London Metro



London metro, London Transport

Protein interactions



Protein-protein interaktionsnetværk,
Jeong et al, Nature Review | Genetics

Examples of applications of graphs

Graph	Vertices	Edges
communication	computers, routers, etc.	cables
transport	intersections	roads
transport	airports	flight routes
games	position	valid move
neural networks	neuron	Synapses
financial network	stocks or currencies	transactions
circuits	logical gates	connections
food chain	species	predator-prey
molecule	atom	bindings

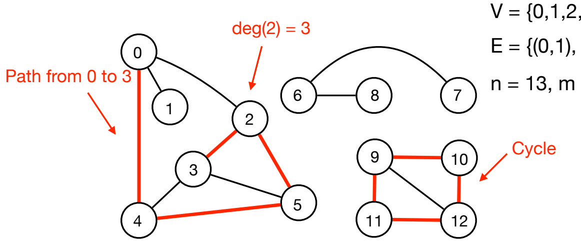
Terminology

- Undirected graph. G = (V, E)
  - V = set of vertices
  - E = set of edges (each edge is a pair of vertices)
  - n = |V|, m = |E|
- Path *(da: sti)*. Sequence of vertices connected by edges.
- Cycle *(da: kreds)*. (Nonempty) path starting and ending at the same vertex.
- Degree *(da: grad)*. deg(v) = the number of neighbours of v, ore dges incident to v.
- Connectivity *(da: sammenhæng)*. A pair of vertices are connected if there is a path between them

V = {0,1,2,…,12}

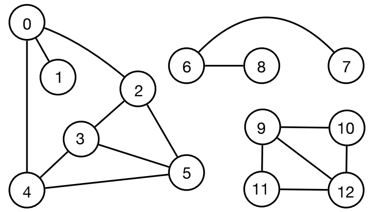
E = {(0,1), (0,2) (0,4),(2,3),…, (11,12)}

n = 13, m = 15



Undirected graphs

- Lemma. $\sum_{v \in V} \deg(v) = 2m$.
- Proof. How many times is each edge counted in this sum?



Algoritmic problems on graphs

- Path. Is there a path connecting s and t?
- Shortest path. What is the shortest path connecting s and t?
- Longest path. What is the longest simple (ie. not self-intersecting) path connecting s and t?

- Cycle. Is there a cycle in the graph?
- Euler tour. Is there a cycle that uses each *edge* exactly once?
- Hamilton cycle. Is there a cycle that visits each *vertex* exactly once?

- Connectivity. Is any pair of vertices connectable by a path?
- Minimum spanning tree. What is the cheapest way of connecting all vertices?
- Biconnectivity. Is there a vertex whose removal would cause the graph to be disconnected?

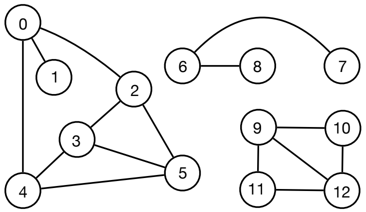
- Planarity. Is it possible to draw the graph in the plane without edge crossings?
- Graph isomorphism. Do these sets of vertices and edges represent the same graph?

Introduction to graphs

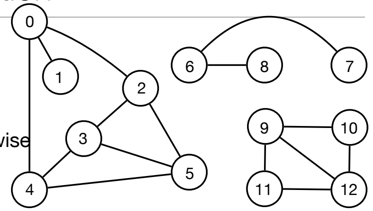
- Undriected graphs
- **Representation**
- Depth first search
  - Connected components
- Breadth first search
  - Bipartite graphs

Representation

- G graph with n vertices and m edges.
- Representation. We need the following operations on graphs.
  - ADJACENT(v, u): determine whether u and v are neighbours.
  - NEIGHBOURS(v): return all neighbours of v.
  - INSERT(v, u): add the edge (v, u) to G (unless it is already there).



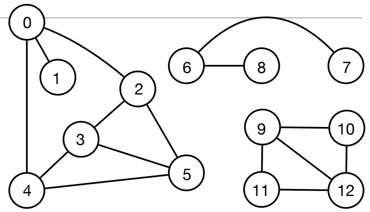
Adjacency matrix representation



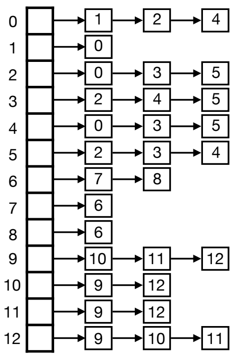
- Graph G with n vertices and m edges.
- Adjacency matrix (*incidensmatrix*).
  - 2D n × n table A.
  - A[i,j] = 1 if i and j er neighbours, 0 otherwise
- Complexity?
- Space. $O(n^2)$
- Time.
  - ADJACENT O(1) time
  - NEIGHBOURS(v) O(n) time.
  - INSERT(v, u) O(1) time.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	1	0	0	0	0	0	0	0
3	0	0	1	0	1	1	0	0	0	0	0	0	0
4	1	0	0	1	0	1	0	0	0	0	0	0	0
5	0	0	1	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	1
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	1	1	0

Adjacency list representation



- Graph G with n vertices and m edges.
- Adjacency list (*da: incidensliste*).
  - Tabel A[0..n-1].
  - A[i] contains a list of all neighbours to i.
- Complexity?
- Space. $O(n + \sum_{v \in V} deg(v)) = O(n + m)$
- Time.
  - ADJACENT, NEIGHBOURS, INSERT O(deg(v)) time.



Repræsentation

Datastruktur	ADJACENT	NEIGHBOURS	INSERT	Plads
adjacency matrix	O(1)	O(n)	O(1)	$O(n^2)$
adjacency list	O(deg(v))	O(deg(v))	O(deg(v))	O(n+m)

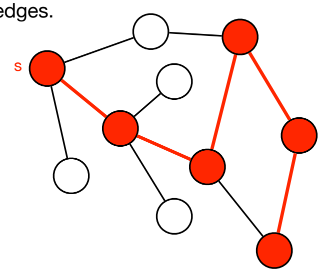
- Real world graphs are often sparse.

Introduction to graphs

- Undirected graphs
- Representation
- Depth first search
  - Connected Components
- Breadth first search
  - Bipartite graphs

Depth first search

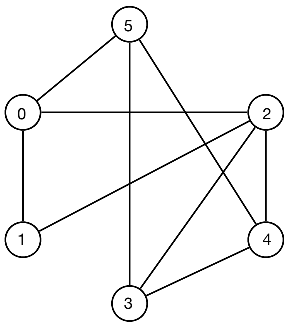
- Algorithm for systematically visiting all vertices and edges.
- Depth first search *(dybde-først)* from vertex s.
  - Initially, all vertices un-marked, and visit vertex s.
  - Visit vertex v:
    - Mark v.
    - Visit all unmarked neighbours of v recursively.



- Intuition.
  - Explore out from s in some direction, until coming to a “dead end”.
  - Go back to the last place where there were unexplored edges. Repeat.
- Discovery time *(starttid)*. First time a vertex is visited.
- Finish time *(sluttid)*. Last time a vertex is visited.

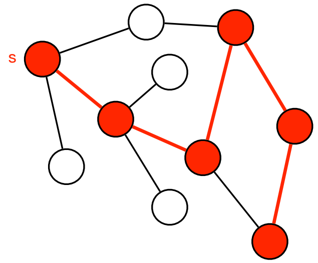
Depth first search

- Exercise. Run DFS (depth first search) from vertex 0, and report discovery time and finish time for each vertex. Assume the adjacency lists are sorted.



Depth first search

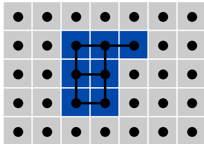
```
DFS(s)
   time = 0
   DFS-VISIT(s)

DFS-VISIT(v)
   v.d = time++
   marker v
   foreach unmarked neighbour u
      DFS-VISIT(u)
      u.π = v
   v.f = time++
```



- Time. (assuming the graph is given in adjacency list representation)
  - Recursion? once per vertex.
  - O(deg(v)) time spent on vertex v.
  - $\Longrightarrow$ total $O(n + \sum_{v \in V} deg(v)) = O(n + m)$ time.
  - Only visits vertices connected to s.

Flood fill

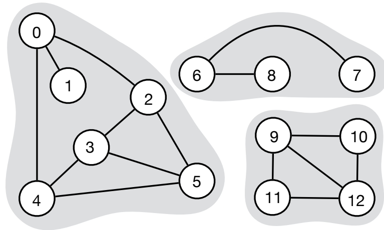
- Flood fill *(farveudfyldning)*. Chance the colour of a connected area of green pixels.



- Algorithm.
  - Build a grid graph and run DFS (depth first search).
  - Vertex: pixel.
  - Edge: goes between neighbouring pixels of same colour.
  - Area: all vertices connected to a given vertex.

Connected components

- Definition. A connected component *(sammenhængskomponent)* is a maximal subset of connected vertices.



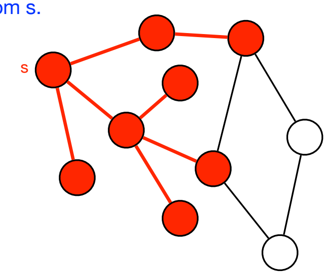
- How does one find all connected components?
- Algorithm.
  - Initially, let all vertices be unmarked.
  - While there is an unmarked vertex:
    - Chose an unmarked vertex v, run DFS from v.
- Time. O(n + m).

Introduction to graphs

- Undirected graphs
- Representation
- Depth first search
  - Connected Components
- Breadth first search
  - Bipartite graphs

Breadth first search

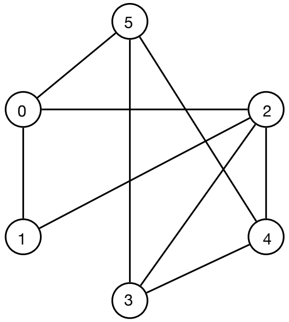
- Breadth first search *(breddeførst søgning)* from s.
  - Initially, let all vertices be unmarked.
  - Mark s, and add s to the queue K.
  - While K is not empty:
    - Excerpt vertex v from K.
    - For all unmarked neighbours u of v
      - Mark u.
      - Add u to K.



- Intuition.
  - Explore, starting from s, in all directions - in increasing distance from s.
- Shortest paths from s.
  - Distance to s in BFS tree = shortest distance to s in the original graph.

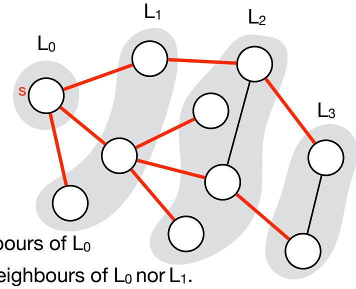
Breadth first search

- Exercise. Run BFS from vertex 0 and indicate the shortest paths. Assume the adjacency lists are sorted.



Shortest paths

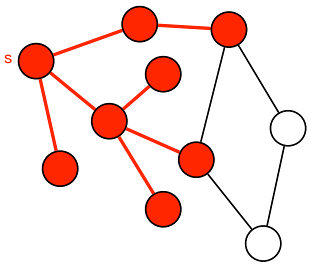
- Lemma. BFS finds the length of the shortest path from s to all other vertices.
- Intuition.
  - BFS assigns vertices to layers. Layer number i contains all vertices of distance i to s.



  - What does each layer contain?
  - L_0 : {s}
  - L_1 : all neighbours of L_0.
  - L_2 : all neighbours if L_1 that are not neighbours of L_0
  - L_3 : all neighbours of L_2 that neither are neighbours of L_0 nor L_1.
  - ...
  - L_i : all neighbours til L_{i-1} not neighbouring L_j for $j < i-1$
    - = all vertices of distance i from s.

Breadth first search

```
BFS(s)
   mark s
   s.d = 0
   K.ENQUEUE(s)
   repeat until K is empty
      v = K.DEQUEUE()
      foreach unmarked neighbour u
         mark u
         u.d = v.d + 1
         u.π = v
         K.ENQUEUE(u)
```



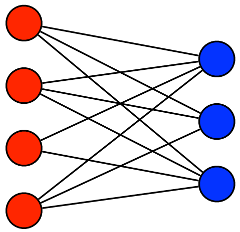
- Time. (assuming adjacency list representation)
  - Each vertex is visited at most once.
  - O(deg(v)) time spent on vertex v.
  - $\Longrightarrow$ total $O(n + \sum_{v \in V} \deg(v)) = O(n + m)$ time.
  - Only vertices connected to s are visited.

Introduction to graphs

- Undirected graphs
- Representation
- Depth first search
  - Connected Components
- Breadth first search
  - Bipartite graphs

Bipartite graphs

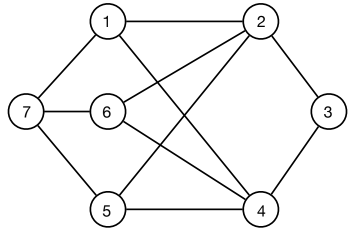
- Definition. A graph is bipartite *(todelt)* if and only if all vertices can be coloured red and blue such that every edge has exactly one red and one blue endpoint.
- Alternativt definition. A graph is bipartite if and only if its vertices can be partitioned into two sets V_1 and V_2 such that all edges go between V_1 and V_2.

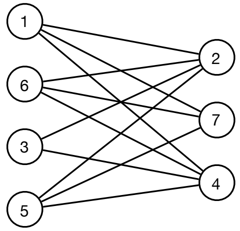


- Application.
  - Scheduling, matching, assigning customers to servers, assigning jobs to machines, assigning students to advisors/labs, …
  - Many graph problems are *easier* on bipartite graphs.

Bipartite graphs

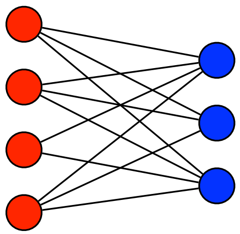
- Challenge. Given a graph G, determine whether G is bipartite.





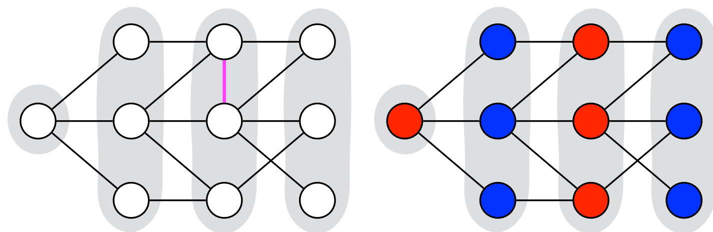
Bipartite graphs

- Lemma. A graph G is bipartite if and only if all cycles in G have even length.
- Proof. $\Longrightarrow$
  - If G is bipartite, all cycles start and end in the same side.



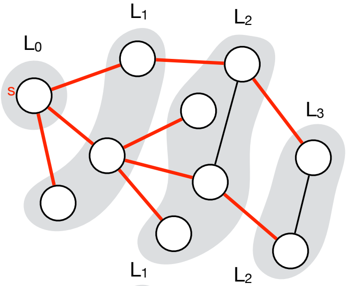
Bipartite graphs

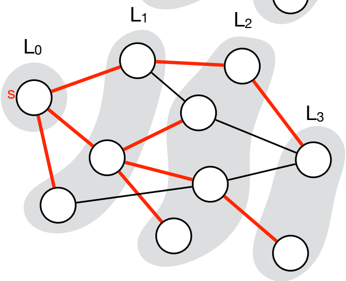
- Lemma. A graph G is bipartite if and only if all cycles in G have even length.
- Proof. $\Longleftarrow$
  - Choose a vertex v and consider BFS layers $L_0, L_1, \ldots, L_k$.
  - All cycles have even length $\Longrightarrow$ There is no edge between vertices of the same layer $\Longrightarrow$ We can assign alternating (red, blue) colours to the layers $\Longrightarrow$ G is bipartite.



Bipartite graphs

- Algorithm.
  - Run BFS on G.
  - For every edge of G, determine whether it goes between vertices of the same layer.
- Time.
  - O(n + m)





Graph algorithms

Algorithm	TIme	Space
Depth first search	O(n + m)	O(n + m)
Breadth first search	O(n + m)	O(n + m)
Connected components	O(n + m)	O(n + m)
Bipartite	O(n + m)	O(n + m)

- All running times assume that G is given in the adjacency list representation.

Introduction to graphs

- Undirected graphs
- Representation
- Depth first search
  - Connected components
- Breadth first search
  - Bipartite graphs