Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs

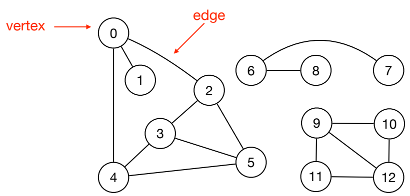
Philip Bille

Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs

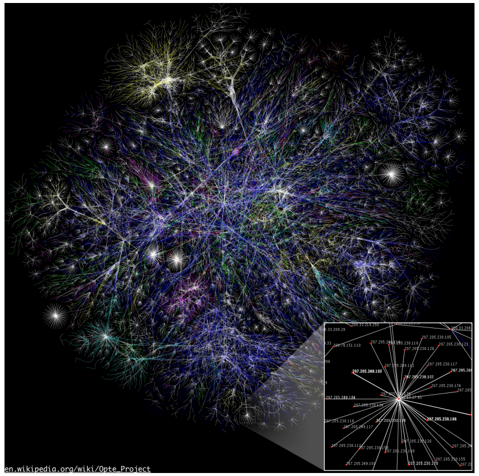
Undirected graphs

- Undirected graph. Set of vertices pairwise joined by edges.



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

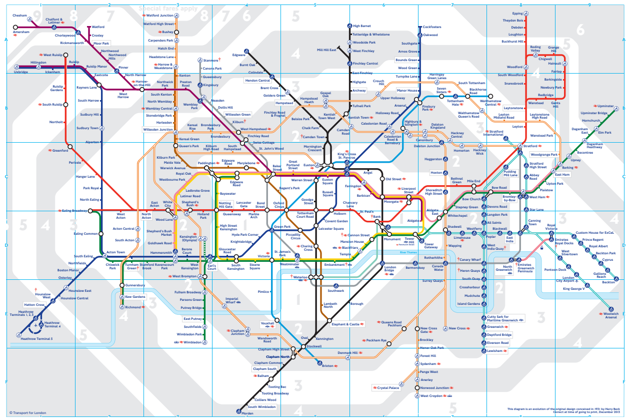
Visualizing the Internet



Visualizing Friendships on Facebook

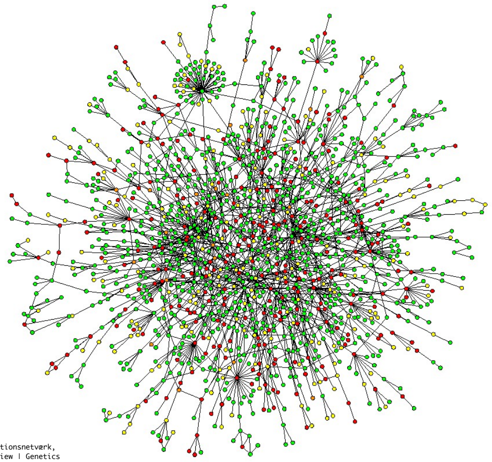
"Visualizing friendships", Paul Butler

London Metro



London metro, London Transport

Protein Interaction Networks



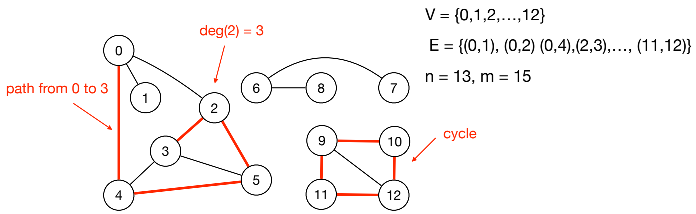
Protein-protein interaktionsnetværk,
Jeong et al, Nature Review | Genetics

Applications of Graphs

Graph	Vertices	Edges
communication	computers	cables
transport	intersections	roads
transport	airports	flight routes
games	position	valid move
neural network	neuron	synapses
financial network	stocks	transactions
circuit	logical gates	connections
food chain	species	predator-prey
molecule	atom	bindings

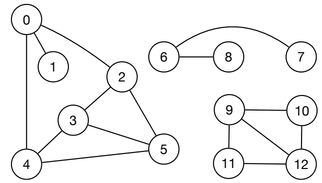
Terminology

- 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. Sequence of vertices connected by edges.
- Cycle. Path starting and ending at the same vertex.
- Degree. deg(v) = the number of neighbors of v, or edges incident to v.
- Connectivity. A pair of vertices are connected if there is a path between them



Undirected Graphs

- Lemma. $\sum_{v \in V} \deg(v) = 2m$.
- Proof. How many times is each edge counted in the 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 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 uses each vertex exactly once?

- Connectivity. Are all pairs of vertices connected?
- Minimum spanning tree. What is the best way of connecting all vertices?
- Biconnectivity. Is there a vertex whose removal would cause the graph to be disconnected?

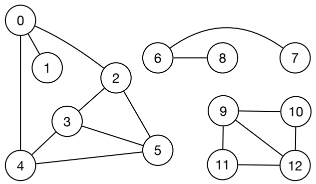
- Planarity. Is it possible to draw the graph in the plane without edges crossing?
- Graph isomorphism. Do these sets of vertices and edges represent the same graph?

Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs

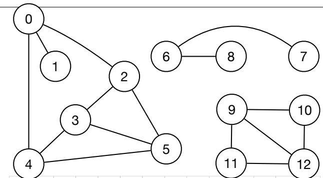
Representation

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



Adjacency Matrix

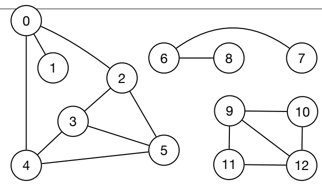
- Graph G with n vertices and m edges.
- Adjacency matrix.
  - 2D n × n array A.
  - A[i,j] = 1 if i and j are neighbors, 0 otherwise
- Space. O(n^2)
- Time.
  - ADJACENT and INSERT in O(1) time.
  - NEIGHBOURS in O(n) time.

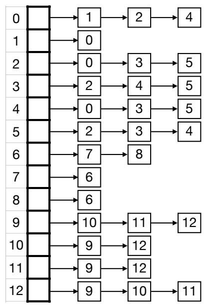


	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	1	0	0	0	0	0	0	0
3	0	0	1	0	1	1	0	0	0	0	0	0	0
4	1	0	0	1	0	1	0	0	0	0	0	0	0
5	0	0	1	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0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

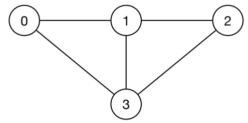
- Graph G with n vertices and m edges.
- Adjacency list.
  - Array A[0..n-1].
  - A[i] is a linked list of all neighbors of i.
- Space. O(n + $\sum_{v \in V}$ deg(v)) = O(n + m)
- Time.
  - ADJACENT, NEIGHBOURS, INSERT O(deg(v)) time.





Adjacency List

```
n = 4
adj = [[] for i in range(n)]
adj[0].append(1)
adj[1].append(0)
adj[0].append(3)
adj[3].append(0)
adj[1].append(2)
adj[2].append(1)
adj[1].append(3)
adj[3].append(1)
adj[2].append(3)
adj[3].append(2)
```



[[1, 3], [0, 2, 3], [1, 3], [0, 1, 2]]

Representation

Data structure	ADJACENT	NEIGHBOURS	INSERT	space
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)

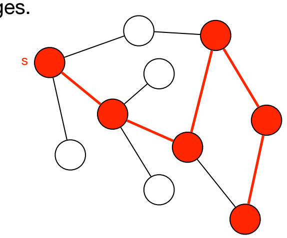
- Real world graphs are often sparse.

Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs

Depth-First Search

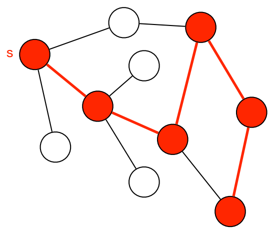
- Algorithm for systematically visiting all vertices and edges.
- Depth first search from vertex s.
  - Unmark all vertices and visit s.
  - Visit vertex v:
    - Mark v.
    - Visit all unmarked neighbours of v recursively.



- Intuition.
  - Explore from s in some direction, until we read dead end.
  - Backtrack to the last position with unexplored edges.
  - Repeat.
- Discovery time. First time a vertex is visited.
- Finish time. Last time a vertex is visited.

Depth-First Search

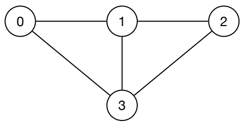
```
DFS(s)
   time = 0
   DFS-VISIT(s)

DFS-VISIT(v)
   v.d = time++
   mark v
   for each unmarked neighbor u
      u.π = v
      DFS-VISIT(u)
   v.f = time++
```



- Time. (on 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.

Depth-First Search

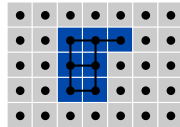
```
visited = [False for i in range(n)]

def dfs(s):
   if (visited[s]):
      return
   visited[s] = True
#   print(s)
   for u in adj[s]:
      dfs(u)

dfs(0)
```



0
1
2
3

Flood Fill

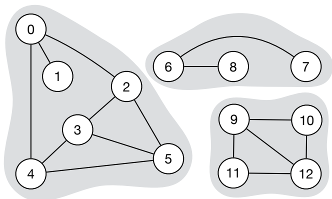
- Flood fill. Chance the color of a connected area of green pixels.



- Algorithm.
  - Build a grid graph and run DFS.
  - Vertex: pixel.
  - Edge: between neighboring pixels of same color.
  - Area: connected component

Connected Components

- Definition. A connected component is a maximal subset of connected vertices.



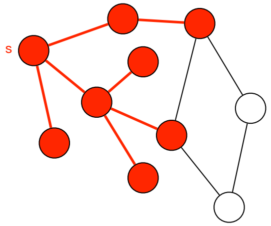
- How to find all connected components?
- Algorithm.
  - Unmark all vertices.
  - While there is an unmarked vertex:
    - Chose an unmarked vertex v, run DFS from v.
- Time. O(n + m).

Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs

Breadth-First Search

- Breadth first search from s.
  - Unmark all vertices and initialize queue Q.
  - Mark s and Q.ENQUEUE(s).
  - While Q is not empty:
    - v = Q.DEQUEUE().
    - For each unmarked neighbor u of v
      - Mark u.
      - Q.ENQUEUE(u).

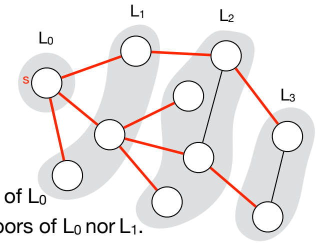


- Intuition.
  - Explore, starting from s, in all directions - in increasing distance from s.

- Shortest paths from s.
  - Distance to s in BFS tree = shortest distance to s in the original graph.

Shortest Paths

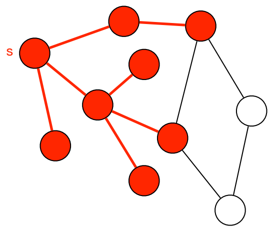
- Lemma. BFS finds the length of the shortest path from s to all other vertices.
- Intuition.
  - BFS assigns vertices to layers. Layer i contains all vertices of distance i to s.



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

Breadth-First Search

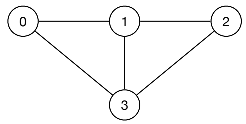
```
BFS(s)
   mark s
   s.d = 0
   Q.ENQUEUE(s)
   repeat until Q.ISEMPTY()
      v = Q.DEQUEUE()
      for each unmarked neighbor u
         mark u
         u.d = v.d + 1
         u.π = v
         Q.ENQUEUE(u)
```



- Time. (on 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.

Breadth-First Search

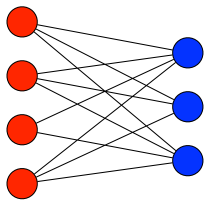
```
from collections import deque
q = deque()
visited = [False for i in range(n)]
distance = [-1 for i in range(n)]


visited[0] = True
distance[0] = 0
q.append(0)
while q:
    s = q.popleft()
#    print(s)
    for u in adj[s]:
        if (visited[u]):
            continue
        visited[u] = True
        distance[u] = distance[s]+1
        q.append(u)
```



0
1
3
2

Bipartite Graphs

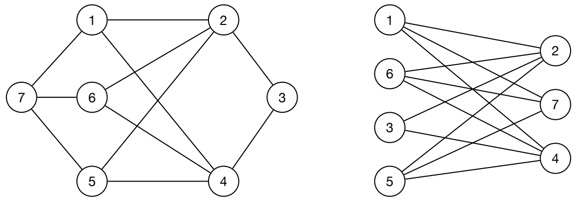
- Definition. A graph is bipartite if and only if all vertices can be colored red and blue such that every edge has exactly one red endpoint and one blue endpoint.
- Equivalent definition. A graph is bipartite if and only if its vertices can be partitioned into two sets V_1 and V_2 such that all edges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



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

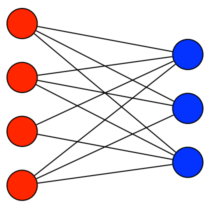
Bipartite Graphs

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



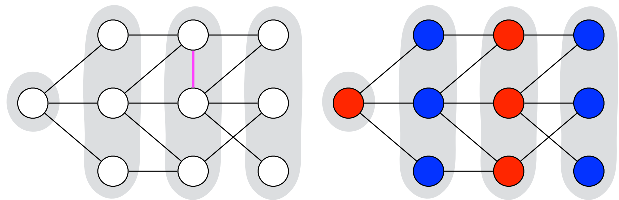
Bipartite Graphs

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



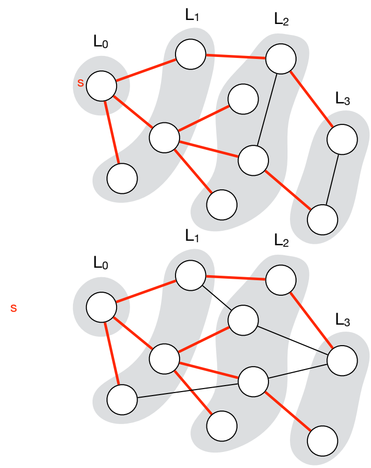
Bipartite Graphs

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



Bipartite Graphs

- Algorithm.
  - Run BFS on G.
  - For each edge in G, check if it's endpoints are in 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 on the adjacency list representation.

Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs