13:00

CMPT231

Lecture 10: ch22

Graph Algorithms

Some material from [Sedgewick + Wayne, "Algorithms"](http://algs4.cs.princeton.edu/)

13:00

Romans 10:13-15 (NIV)

“Everyone who **calls** on the name of the Lord will be saved.”

How, then, can they call on
the one they have not **believed** in?
And how can they believe in
the one of whom they have not **heard**?
And how can they hear
without someone **preaching** to them?
And how can anyone preach unless they are **sent**?

As it is written:

13:00

Outline for today

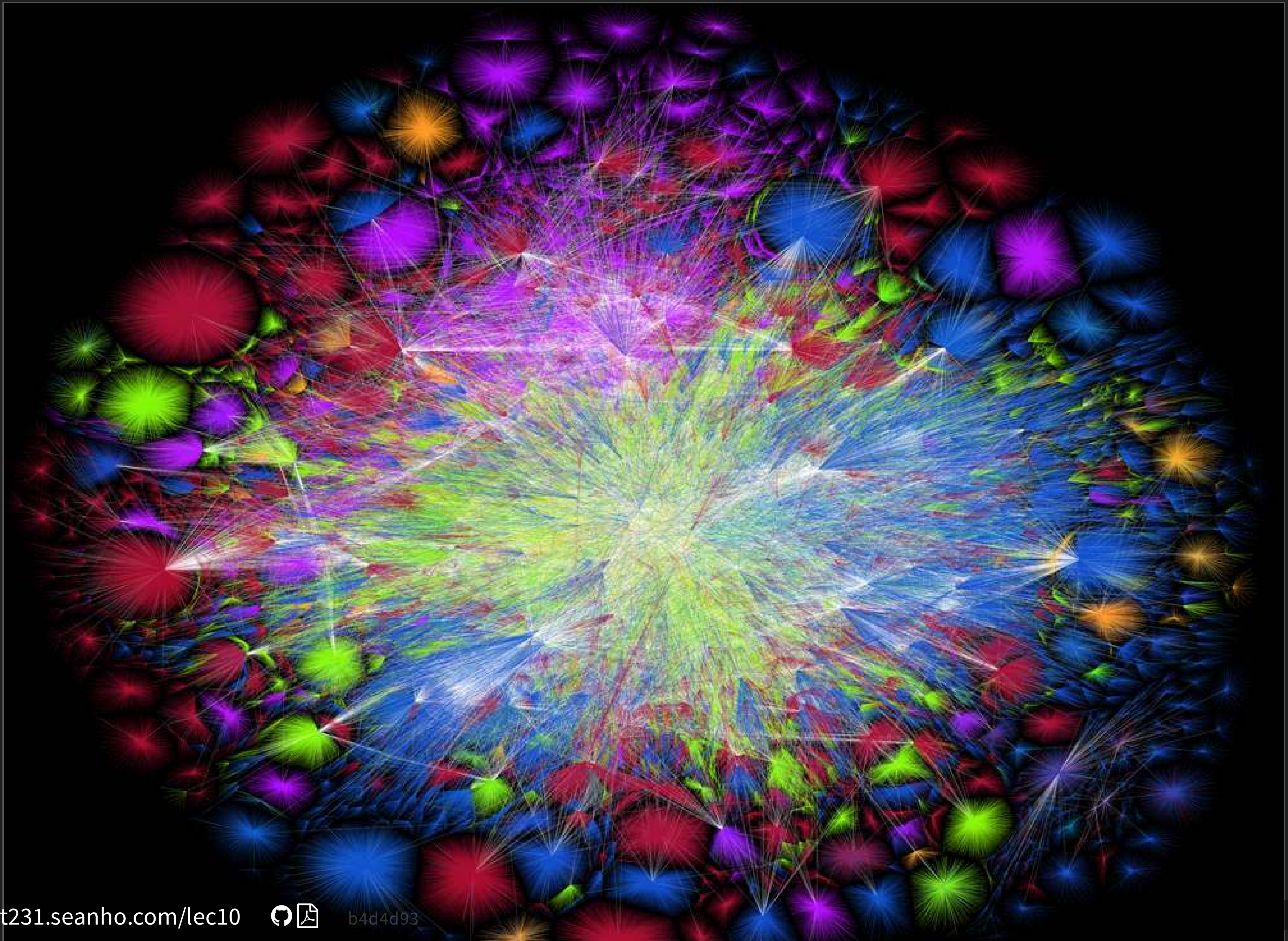
- Intro to **graph** algorithms
  - Applications and typical problems
  - Edge list, adjacency list, adjacency matrix
- **Breadth-first** graph traversal
- **Depth-first** graph traversal
  - **Parenthesis** structure
  - Edge **classification**
  - Topological **sort**
  - Finding **strongly-connected** components

Intro to graph algorithms

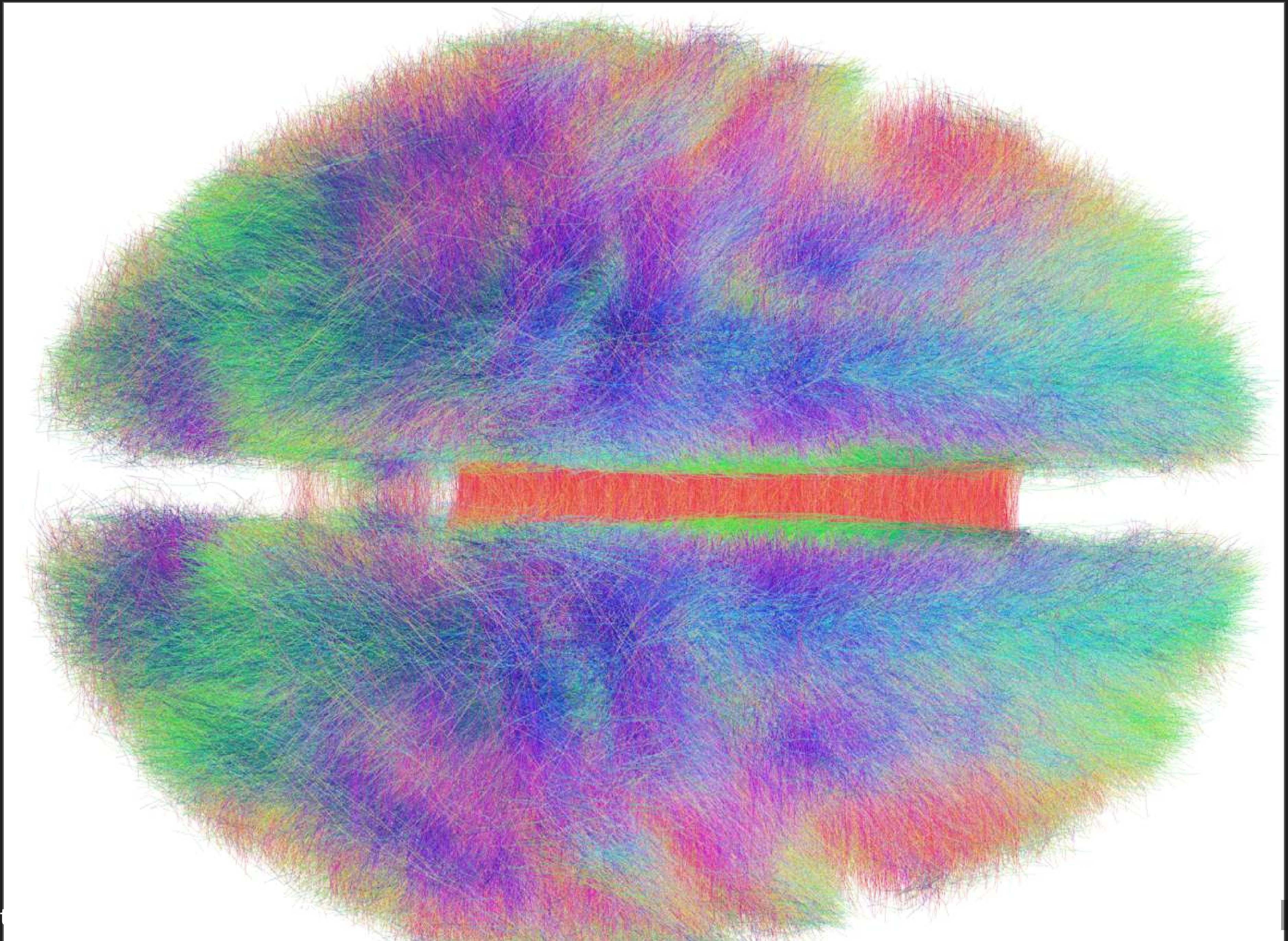
- Representing **graphs**: $G = (V, E)$
- V: **vertices** / nodes
  - storage: array, linked-list, etc.
- E: **edges** connecting vertices
  - directed or undirected
  - storage: edge list, adjacency matrix, etc.
- Some corner cases:
  - Self-loop: edge from vertex to itself
  - Parallel edges: multiple edges with same start/end
- **Complexity** of graph algorithms in terms of $|V|$ and $|E|$

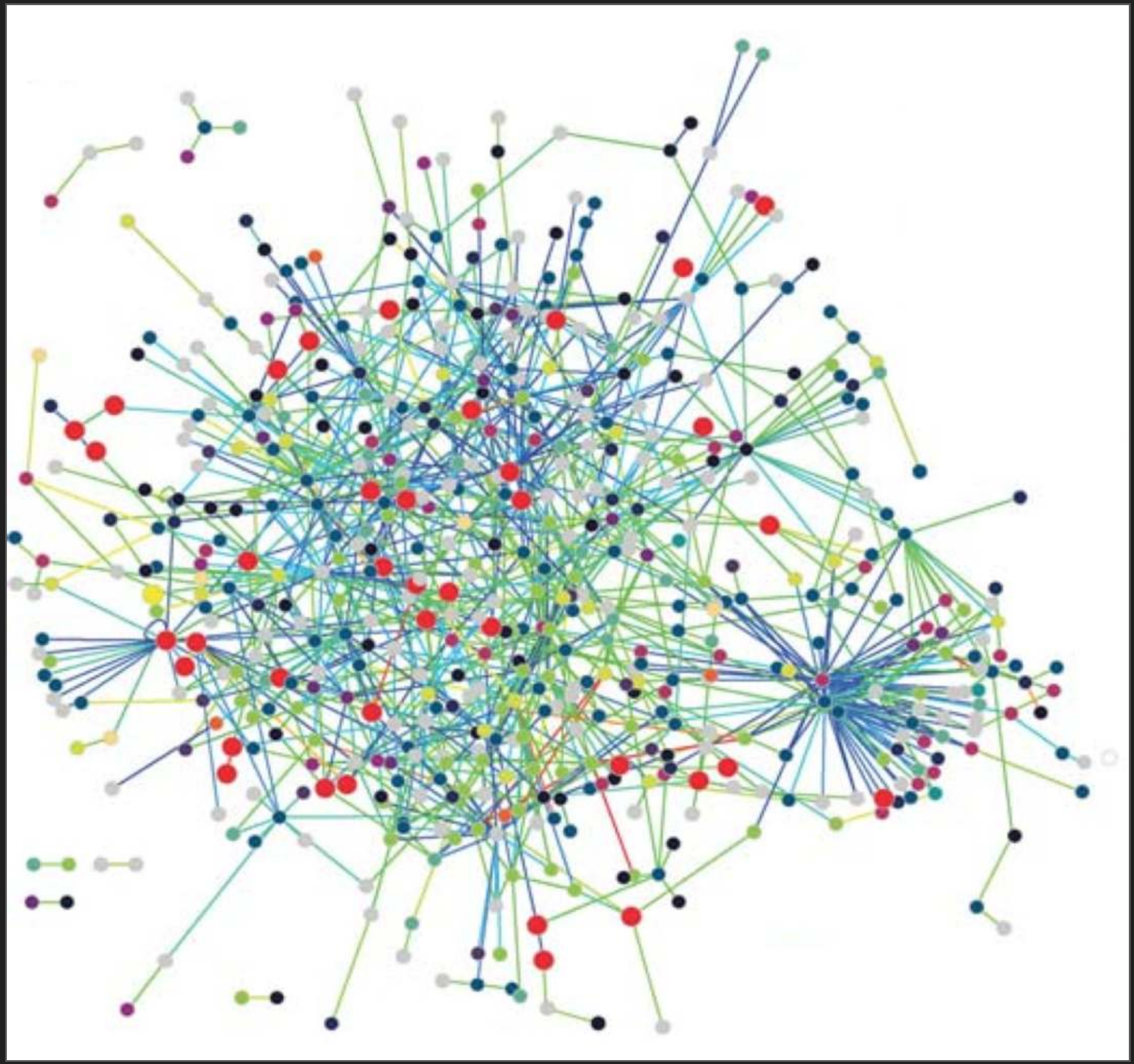
Applications of graphs

graph	vertex	edge
air **transport**	airport	flight path
social	person	friendship/relationship
internet	computer	network connection
finance	stock/asset	transaction
neural net	neuron	synapse
protein net	protein	protein-protein interaction



24hrs of flights in/out of Europe: [422South for NATS](http://422south.com/work/euro-24-air-traffic-visualization-for-nats)





Problems in graph theory

- **Path finding**: is there a path from u to v?
- **Shortest path**: find the shortest path from u to v
- **Cycle**: does the graph have any cycles?
- **Euler cycle**: traverse each edge exactly once
- **Hamilton cycle**: touch each vertex exactly once
- **Connectivity**: are all the vertices connected?
- **Bi-connectivity**: can you disconnect the graph by removing one vertex?
- **Planarity**: draw graph in 2D w/o crossing edges?
- **Isomorphism**: are two graphs equivalent?

Outline for today

- Intro to graph algorithms
  - Applications and typical problems
  - **Edge list, adjacency list, adjacency matrix**
- **Breadth-first graph traversal**
- Depth-first graph traversal
  - Parenthesis structure
  - Edge classification
  - Topological sort
  - Finding strongly-connected components

Representing edges

- **Edge list**: array/list of (u,v) pairs of nodes
  - [(1,2), (1,3), (2,4)]
  - How to find neighbours of a vertex u?
- **Adjacency list**: indexed by start node
  - [{1: [2, 3]}, {2: [1, 4]}, {3: [1]}, {4: [2]}]
  - How to find the (out)-degree of each vertex?
- **Adjacency matrix**: boolean |V| x |V| matrix
  - A[i,j] = 1 iff (i,j) is an edge: $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$
  - What about directed graphs? Weighted graphs?

Graph traversal: breadth-first

- **Traversal**: visits each node exactly once
- BFS: overlay a **breadth-first tree**
  - Choose a start (root) node
  - Path in tree = shortest path from root
    - Only nodes reachable from start node
  - BFS tree not necessarily unique
- Graph could have **loops**:
  - Need to **track** which nodes we've seen
- Assign **colours** to nodes as we traverse graph:
  - White: unvisited
  - Grey: on border (some unvisited neighbours)
  - Black: no unvisited neighbours

BFS algorithm

- In: **vertex** list, **adjacency** (linked) list, **start** node
- Out: **modify** vertex list, adding parent pointers

```
def BFS( V, E, start ):
  init V all white and NULL parer
  start.colour = grey
  init FIFO: Q.push( start )
  while Q.notempty():
    u = Q.pop()
    for v in E.adj[ u ]:
      if v.colour == white:
v.colour = grey
v.parent = u
Q.push( v )
    u.colour = black
```

![BFS](static/img/Breadth-
First-Search-Algorithm.gif)

Complexity?

BFS properties

- BFS examines nodes in order of **distance** from source
  - Queue first holds all nodes of distance k,
  - Then all nodes of distance k+1, etc.
- **Levels** of BFS tree = nodes of same distance from source
- $\Rightarrow$ BFS computes **shortest paths** from source to all other reachable nodes in time $O(|V| + |E|)$
  - e.g., Kevin Bacon number:
    - vertices = actors, edges = shared movies

Outline for today

- Intro to graph algorithms
  - Applications and typical problems
  - Edge list, adjacency list, adjacency matrix
- Breadth-first graph traversal
- **Depth-first graph traversal**
  - Parenthesis structure
  - Edge classification
  - Topological sort
  - Finding strongly-connected components

Trémaux maze solving

- Graph representation of a **maze**:
  - Vertex = **intersection**, edge = **passage**
- **Theseus** slaying the Minotaur in the Labyrinth
  - Ariadne gave him a tool: **ball of string**:
- **Unwind** string as you go
  - **Track** each visited intersection + passage
  - **Retrace** steps when

`![Theseus in the Labyrinth](static/img/Theseus-Labyrinth.jpg)`

Depth-first search

- First explore as **deep** as we can
  - **Backtrack** to explore other paths
  - **Recursive** algorithm (ball of string = call stack)
- **Colouring**: white = undiscovered, grey = discovered, black = finished (visited all neighbours)
- Add **timestamps** on discover and finish
- Overlay a **forest** on the graph
  - **Subtree** at a node = all nodes visited between this node's discovery and finish

DFS algorithm

```
def DFS( V, E ):
  init V all white and NULL parent
  time = 0
  for u in V:     # why loop over ALL vert
    if u.colour == white:
      DFS-Visit( V, E, u )
```

```
def DFS-Visit( V, E, u ):
  time++
  u.discovered = time
  u.colour = gray
  for v in E.adj[ u ]:
    if v.colour == white:
      v.parent = u
      DFS-Visit( V, E, v )
  u.colour = black
  time++
  u.finished = time
```

`![DFS anim](static/img/Depth-First-Search.gif) ![DFS](static/img/DFS.svg)`

13:00

Outline for today

- Intro to graph algorithms
  - Applications and typical problems
  - Edge list, adjacency list, adjacency matrix
- Breadth-first graph traversal
- Depth-first graph traversal
  - **Parenthesis structure**
  - **Edge classification**
  - Topological sort
  - Finding strongly-connected components

DFS: parenthesis structure

- Each node's **subtree** is visited between its discovery and finish times
- **Print** a $(._u$ when we discover node u
  - Print a $)_u$ when we finish it
- Output is a valid **parenthesisation**:
  - e.g., $\left(\cdot_u \left(\cdot_v \left(\cdot_w\right)_w\right)_v \left(\cdot_x \left(\cdot_y\right)_y\right)_x\right)_u \left(\cdot_z\right)_z$
  - But not $(\cdot_u (\cdot_v)_u)_v$
- The (discover, finish) intervals for any two vertices are
  - Either completely **disjoint**
  - Or one **contained** in the other

DFS: white-path theorem

- The (d, f) interval for v is **contained** in u ⇔ v is a **descendant** of u in the DFS
  - i.e., u.d < v.d < v.f < u.f
- **White-path** theorem:
  - v is a **descendant** of u in the DFS ⇔
  - When u is discovered, there is a **path** u → v all of white vertices

![DFS]
(static/img/DFS.svg)

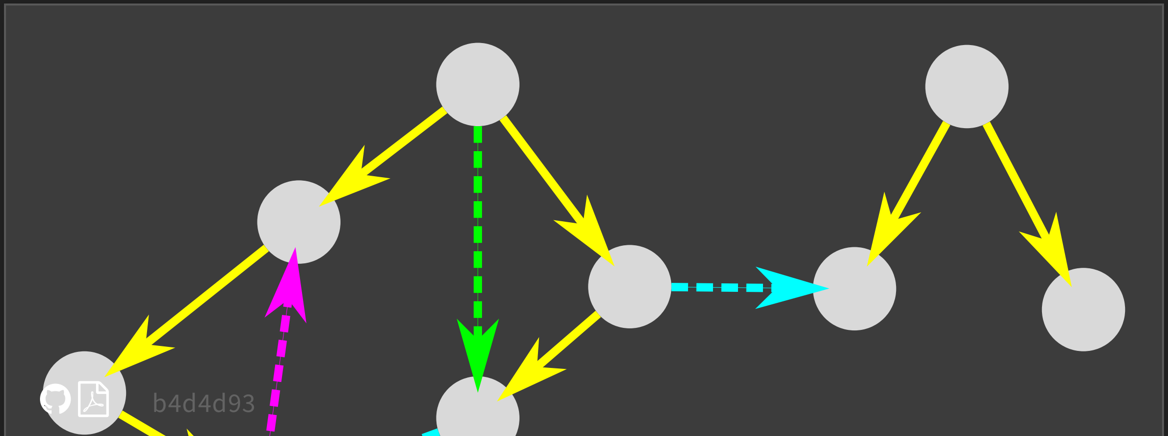
DFS: flood-fill

- **Vertex**: pixel
- **Edge**: adjacent pixels of similar colour
- **Blob**: all pixels connected to given pixel

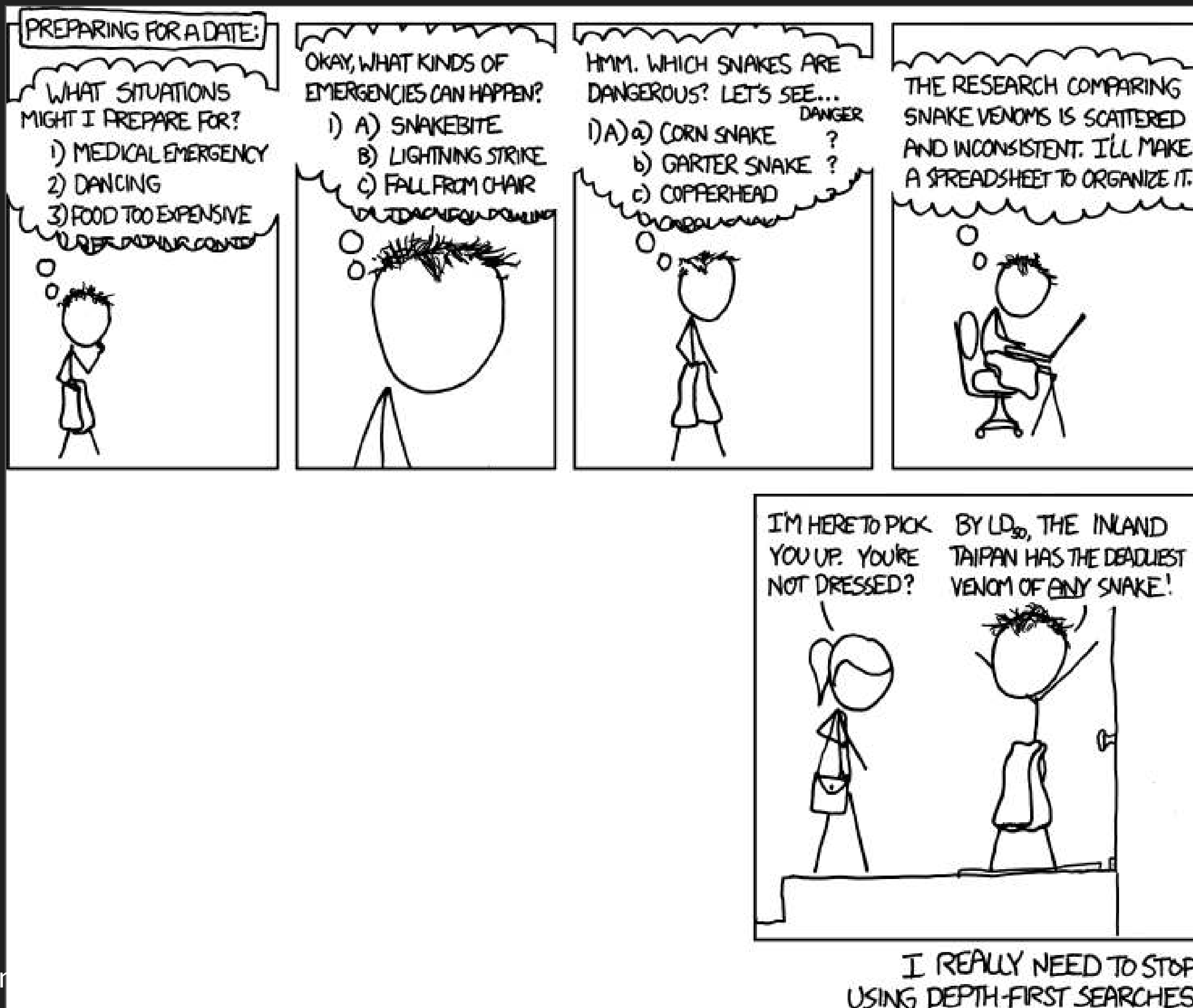
`![Manhattan map](static/img/Manhattan.svg)`
`![Australia grid](static/img/Australia_grid2.png)`

DFS: edge classification

- All **edges** in a graph are either
  - **Tree** edges: in the DFS forest
  - **Back** edges: up to ancestor in same DFS tree (incl self-loop)
  - **Forward** edges: down to descendant
  - **Cross** edges: different subtrees or DFS trees
- For directed graphs: **acyclic** ⇔ no **back** edges



DFS: preparing for a date (XKCD)



13:00

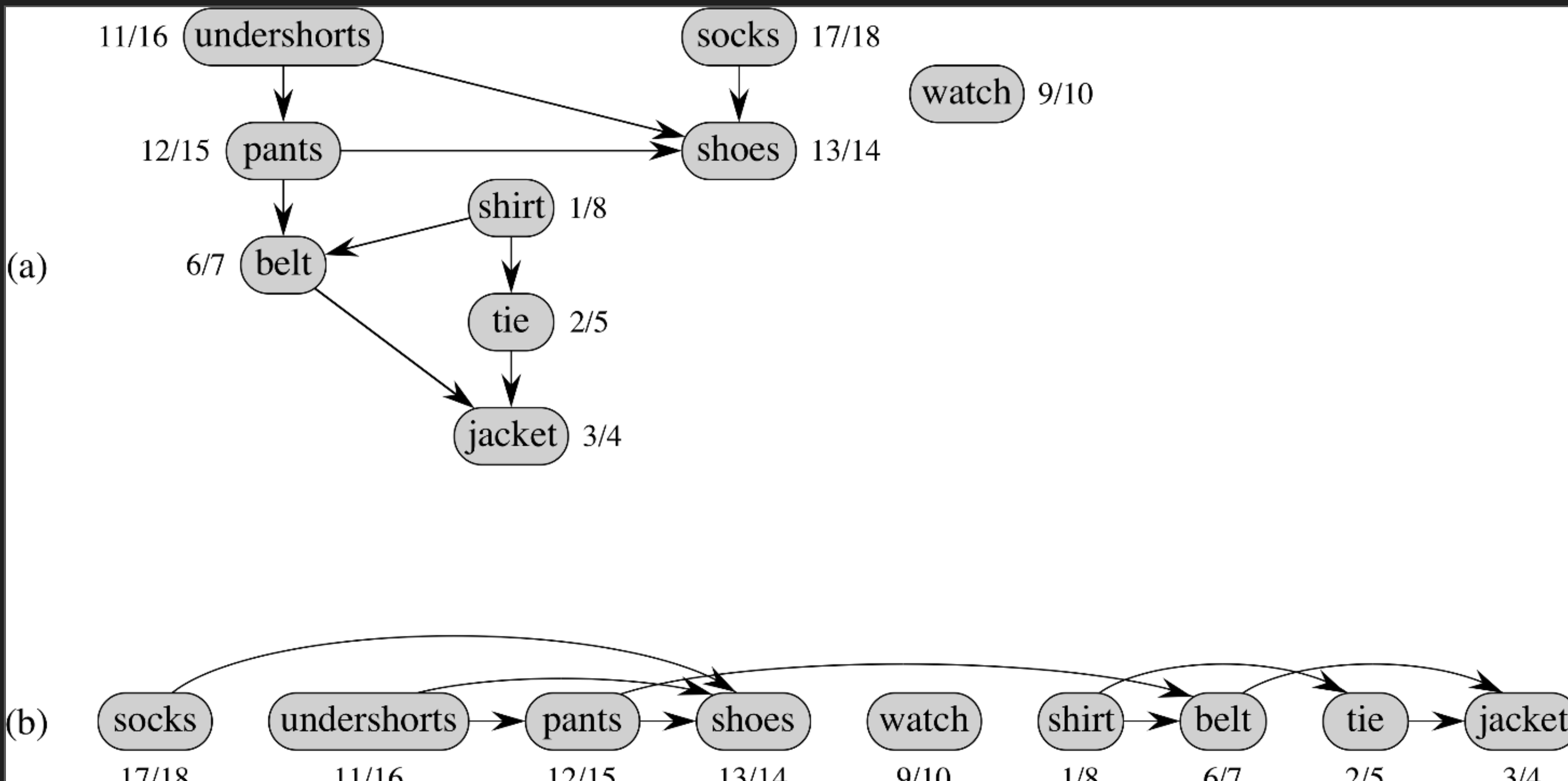
Outline for today

- Intro to graph algorithms
  - Applications and typical problems
  - Edge list, adjacency list, adjacency matrix
- Breadth-first graph traversal
- Depth-first graph traversal
  - Parenthesis structure
  - Edge classification
  - **Topological sort**
  - **Finding strongly-connected components**

DFS: topological sort

- Linear **ordering** of vertices such that:
  - for every edge $u \to v$, u comes **before** v in the sort
  - Assumes **no cycles**! (i.e., DAG: directed acyclic)
- **Applications**: dependency resolution, compiling files, task planning / Gantt chart
- Use **DFS** to sort in **decreasing** order of finish time
  - As each vertex finishes, insert at head of a linked list
- DFS might not be **unique**, so topological sort might not be unique

Topological sort: example

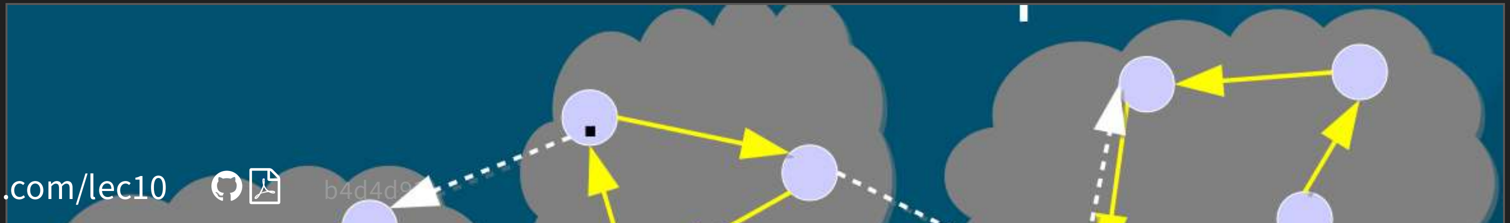


Topological sort: proof

- Recall DFS **colouring**: white = undiscovered
  - grey = discovered, black = finished
- Proof of **correctness**: $(u, v) \in E \Rightarrow v.f < u.f$
- When DFS explores (u,v), what **colour** is v?
  - if **gray**: then v is an **ancestor** of u
    - So (u,v) is a **back** edge
    - So graph has a **loop** (disallowed)
  - if **white**: then v becomes a **child** of u:
    - u.d < v.d < v.f < u.f
  - if **black**: then v is **done**, but not u yet:
    - v.f < u.f

DFS: connected components

- Largest **completely-connected** set of vertices:
  - Every vertex has a **path** to every other vertex in the component
- Algorithm:
  - Compute DFS to find **finishing** times
  - **Transpose** the graph: reverse all edges
  - Compute DFS on transposed graph
    - Start at vertex that finished **last** in orig DFS
  - Each **tree** in final DFS is a separate **component**



Connected components

- (a) **Original** graph:
  - DFS trees shaded
  - DFS starts at c
- (b) **Transpose** graph:
  - All edges reversed
  - DFS trees shaded
  - DFS starts at

![Fig 22-9: components]
(static/img/Fig-22-9.svg)

Problems in graph theory

- **Path finding**: is there a path from u to v?
- **Shortest path**: find the shortest path from u to v
- **Cycle**: does the graph have any cycles?
- **Euler cycle**: traverse each edge exactly once
- **Hamilton cycle**: touch each vertex exactly once
- **Connectivity**: are all the vertices connected?
- **Bi-connectivity**: can you disconnect the graph by removing one vertex?
- **Planarity**: draw graph in 2D w/o crossing edges?
- **Isomorphism**: are two graphs equivalent?

13:00

Outline for today

- Intro to **graph** algorithms
  - Applications and typical problems
  - Edge list, adjacency list, adjacency matrix
- **Breadth-first** graph traversal
- **Depth-first** graph traversal
  - **Parenthesis** structure
  - Edge **classification**
  - Topological **sort**
  - Finding **strongly-connected** components

Online demos

- **Breadth-first** search:
  - U San Fran (generate random graphs)
  - VisuAlgo (draw your own graph; step through code)
- **Depth-first** search:
  - U San Fran (only one tree of the DFS forest)
  - VisuAlgo (edge classification, only one tree)
- **Topological sort**:
  - U San Fran, VisuAlgo
- **Connected** components:
  - U San Fran
  - VisuAlgo (SCC: Kosaraju's algorithm)


13:00