Algorithms II

Graphs

Definition
A **graph** $G = (V, E)$ where $E \subseteq V \times V$

- In an **undirected graph**, edges are unordered pairs.
- A **weighted graph** has a **weighting function** $E \to \mathbb{R}$ which could be positive, zero or negative.
- A graph is **complete** if $E = V \times V$

Representations of a Graph

Adjacency matrix	Adjacency list
A $\lvert V\rvert \times \lvert V\rvert$ matrix $\Theta(\lvert V\rvert^2)$ in size. • If unweighted, each cell stores a 0 or 1 • If weighted, stores the weight • If undirected, it is symmetric, so only half of the matrix will need to be stored.	List of holding a linked list of adjacent vertices.
$O(1)$ to check $(u, v) \in E$.	$O(\lvert V\rvert)$ to check $(u, v) \in E$.
$O(\lvert V\rvert)$ to list neighbours.	$O(\text{neighbour count})$ to list neighbours.
$O(\lvert V\rvert^2)$ to iterate over edges.	$O(\lvert E\rvert)$ to iterate over edges.
More compact for dense graphs (1 bit per edge)	More compact for sparse graphs

- The **transpose of a graph** $G^T = (V, E^T)$ represents a **reverse index**.
- The **square of a graph** $G^2 = (V, E^2)$ where $(u, v) \in E^2$ if there is a path from u to v consisting of at most 2 edges.

Two vertices are adjacent if they share a vertex.

Definition
An **induced subgraph** $G' = (V', E')$ where $V' \subseteq V$ is where

$$\forall u, v \in V' : (u, v) \in E \iff (u, v) \in E'$$

A **clique** in a graph is any induced subgraph that is **complete**.

Colouring problem	Description
Vertex colouring	Assign colours to $v \in V$ so no adjacent vertices have same colour.
Edge colouring	... no adjacent edges have the same colour.
Face colouring	... no two adjacent faces on a **planar graph** have the same colour.

Definitions
- A **planar graph** can be drawn on a plane so no two lines intersect.
- A **face** is a region bounded by edges.

Breadth First Search

To work on cyclic graphs, mark vertices we have visited to prevent us from visiting twice.

```
for v in G.V:
  v.marked = false

Q = new Queue
Enqueue(Q, s)

while !QueueEmpty(Q):
  u = Dequeue(Q)
  if (!u.marked):
    u.marked = true
    for v in G.E.adj[u]:
      Enqueue(Q, v)
```

This algorithm is inefficient because it may add the same vertex multiple times to the queue, to fix this add a **pending** flag for the element.

The flags are be replaced by any data structures where membership can be tested.

Two Vertex Colourability

Input: connected, undirected graph

1. Run BFS to colour the first level as red, 2nd as black, etc $O(|V|)$
2. Check if every adjacent vertices are of different colour $O(|E|)$

Total cost: $O(|V| + |E|)$, for a complete graph, it is $O(|E|)$

> **Note**
> - The algorithm doesn't matter wherever you run it from.
> - If the graph is not connected, that part will not be explored.

Single-source All-destination Shortest Path

- Input: unweighted graph and a starting node s
- Output: distance and shortest path to all nodes

Run BFS with:

- Source distance $= 0$, path $= [\ ]$
- The output of any unreachable vertices have distance ∞

If average path length is $O(|V|)$, then output is $O(|V|^2)$
