BFS

BFS(v) {
    q = empty queue
    v.visited = True
    q.enqueue(v)
    while !q.empty {
        u = q.dequeue()
        for each w in u.neighbours() {
            if (!w.visited) {
                w.visited = True
                q.enqueue(w)
            }
        }
    }
}
Example: BFS with Stack (Not BFS)
Example: BFS with queue (correct)
Survey of problems on graphs

Lecture 28
COMP 250 Winter 2018
(Slides from M. Blanchette)
Outline

• Graphs model many real-world applications
• Graphs lend themselves to nice computer science problems:
  – Shortest path
  – Cycles: Eulerian, Hamiltonian
  – Cliques and independent sets
  – Coloring
  – Matching
• We will only consider undirected graphs
Shortest path problem

• Unweighted Graph Shortest Path:
  – Given an unweighted graph and two vertices u and v,
  – Find the shortest path (minimum number of edges) between u and v

• Weighted Graph Shortest Path:
  – Given an weighted graph and two vertices u and v,
  – Find the shortest path (minimum total edges weight) between u and v

• Applications:
  – Driving from one city to another
  – Routing packets through the internet
  – Solving the Rubik’s cube using the least number of moves
Algo. for unweighted graph shortest path

- Algorithm for unweighted graph:
  - Do a breadth-first search starting at $u$, until $v$ is reached
  - For each vertex visited, remember from which vertex it was reached
  - Works because vertices are visited in increasing order of distance from $u$
Algo. for weighted graph shortest path

Idea:
- Visit vertices in increasing order of distance from u
- The first time you get to v, you came to it via the shortest path.
- This can be done efficiently using a priority queue (see HW5)
Eulerian cycles

- Recall: A cycle is a path that returns to its starting vertex
- An **eulerian cycle** visits each **edge** exactly once (but vertices can be visited more than once)

Problem:
- Given a undirected graph
- Find an eulerian cycle (if one exists)

Algorithm: Sounds hard, but actually easy!
- Start at any vertex $u$ and follow any unvisited edge, as long as this does not result in a graph whose unvisited edges are unreachable
- No need to plan ahead, so algorithm is fast
Hamiltonian cycles

• A Hamiltonian cycle visits each vertex exactly once

Problem:
  – Given a undirected graph
  – Find an Hamiltonian cycle
    (if one exists)

Algorithm: Very hard!
  – Nobody knows how to do much better than trying all
    \((n-1)!\) possible vertex orderings
  – Be famous: find an algorithm that runs in
    polynomial time
Graph coloring

• Problem: Given an undirected graph
  – Find the minimum number of colors needed to paint the vertices so that no pair of adjacent vertices have the same color

• Application: Coloring maps
  – Color countries so that neighbors always have different colors
  – Draw “contact graph”
    • One vertex per country
    • Edges between touching countries

• Be famous: Find a poly. time algo. for graph coloring
Cliques

• Given an undirected graph, a **clique** is a subset of vertices where all vertices are adjacent

• Problem:
  – Given an undirected graph
  – Find the largest clique it contains

• **Be famous**: Find a poly. time algo for finding maximal cliques
Matching

• Example:
  – n people want to get married (vertices)
  – Some pairs of people are compatible (good horoscope, shown by edges), others are incompatible (no edge)
  – Question:
    Can we match everybody?

• NB: The graph contains triangles: what does that mean?

• Efficient algorithms are known but quite complicated