Lecture 15, Oct 30, 2023

Graphs

- A *graph* is a collection of n nodes or vertices V and m edges E connecting two nodes
  - In an *undirected* graphs the edges are unordered pairs $(u, v) \in V$; in a *directed graph* they are ordered and directionality matters
  - Note self connections are possible
  - Note that m is roughly bounded by n^2
- Graphs can be stored in an adjacency list, an array in which each element represents a node, containing a subarray that contains all the nodes connected to that node
  - This representation takes $O(n + m)$ space; it is linear in both nodes and edges
  - For a sparse graph, this is very efficient
  - To check whether there is an edge between i and j takes $O(n)$ in the worst case (since every node can be connected to every other node)
- Another way is to use an adjacency matrix, an $n \times n$ matrix such that A_{ij} is 1 if there is an edge from i to j, or a 0 if there is no edge
  - Undirected graphs have symmetric adjacency matrices
  - This representation takes $O(n^2)$ space, so it's very inefficient for sparse graphs
  - The advantage is that we can query whether there is an edge between i and j in $O(1)$ time
- A *graph search* is a systematic exploration of a graph
  - Different graph search algorithms explore the nodes in different order
  - Graph searches can reveal structural properties of the graph, e.g. connectedness, presence of cycles, shortest paths
- Breath-first search (BFS) searches nodes in the order of their discovery
  - Newly discovered nodes are inserted into a queue to maintain FIFO order
  - The next node to explore is taken from the head of the queue
  - While the algorithm is exploring the graph it maintains 3 properties about each node:
    * $\mathit{color}[v]$, which is white if it's undiscovered, grey if discovered but unexplored (i.e. in the queue), and black if explored
    * $p[v] = u$, the parent of each node (i.e. the node v was discovered while exploring u)
    * $d[v]$, the length of the discovery path from the starting node (i.e. the number of edges in the path from the starting node to v)
  - Note $d[v] = d[u] + 1$
  - The complexity is $O(n + m)$ since we go through the adjacency list
  - The result of BFS, $p[v]$, is called the *BFS tree*; this tree contains the shortest path from the starting node to every other node