**Graphs**

* A graph G consists of a set of vertices (nodes) V and a set of edges E
  + Let V = {v1, … vn}, E = {e1, … em}
  + Each ei = (vj, vk)
* E.g. travelling salesman problem (TSP)
  + Given a list of cities (nodes) and distances between each city (weighted edges), what is the shortest possible route that visits each city exactly once and returns to the starting point? (Hamiltonian cycle)
  + Can be solved with brute force – O(n!) time
  + (We believe) there is no k such that the solution is guaranteed to be no worse than O(nk) time for n nodes
  + This is an **NP-hard problem** – solvable “in principle” but no practically
* **Trees**
  + Single node called **root**
    - Root node has no **parent**
  + Each node may have 0 or more **children**, and exactly 1 **parent**
  + There are no cycles; for any node, there is a unique path connecting to it from the root
  + A node with no children is a **leaf**
  + Non-leaves are **internal nodes**
  + **Height** of a node is the longest path from it to a leaf (look down)
  + **Depth** of a node is the length of path from the root to it (look up)
* **Binary tree**
  + Each node has at most 2 children – left & right nodes
  + **Binary search tree** – each node has a key value
    - **BST property**
      * Key of all nodes in left subtree < key of current node
      * Key of all nodes in right subtree > key of current node
    - A sorted linked list is a degenerate BST
    - Lookup: start at root
  + struct QNode {

string key;

Node\* left;

Node\* right;

}

* + typedef \*Node BST;
  + bool lookup(const BST & root, string key);
    - BST → same BST (returns whether given element was found)
    - Recursively call lookup for left & right subtrees
  + void print(const BST & root);
    - In-order traversal – print left subtree, current node, then right subtree
  + void insert(BST & root, string key);
    - BST → BST with new element added
    - Navigate recursively until at leaf, add new element as a child of leaf
  + void delete(BST & root, string key);
    - Non-empty BST → BST with element removed
    - Cases: node has no children / node has 1 child / node has 2 children
      * If node has 2 children, “replacement node” is the node w/ max value in the left subtree