1. Depth-First Search

For the above graphs, write the paths a depth-first search would find from vertex A to all other vertices in the graphs. If you can't get to a given vertex from vertex A, write "unreachable".

2. Breadth-First Search

For the above graphs, write the paths a depth-first search would find from vertex A to all other vertices in the graphs. If you can't get to a given vertex from vertex A, write "unreachable". Which paths are shorter than the ones found with DFS?

3. isBST

Write a member function named isBST that could be added to a BinaryTree class. Your function should return whether or not a binary tree is arranged in valid binary search tree (BST) order. The empty tree is a BST by definition.

```cpp
bool isBST(BinaryTreeNode* node){...}
```

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.
4. swapChildrenAtLevel

Write a function named `swapChildrenAtLevel` that manipulates a binary tree. Your function should accept two parameters: a reference to a pointer to the root of a tree, and an integer \( k \), and should swap the left and right children of all nodes at level \( k \). In other words, after your function is run, any node at level \( (k+1) \) that used to be its parent's left child should now be its parent's right child and vice versa. For this problem, the overall root of a tree is defined to be at level 1, its children are at level 2, etc.

```cpp
void swapChildrenAtLevel(BinaryTreeNode* node, int k){...}
```

5. isReachable

Write a function named `isReachable` that accepts three parameters: a reference to a `BasicGraph`, and two strings representing names of vertexes \( v1 \) and \( v2 \). Your function should return `true` if a path can be made from the vertex \( v1 \) to the vertex \( v2 \), or `false` if not. If the two vertexes are the same, return `true`. You may assume that the parameter values passed are valid.

```cpp
bool isReachable(BasicGraph& graph, string v1, string v2){...}
```

6. kth Level Friends

Imagine a graph of Facebook friends, where users are vertexes and friendships are edges. Write a function named `kthLevelFriends` that accepts three parameters: a reference to a `BasicGraph`, the string name of a vertex to start from, and an integer \( K \). Your function should return a set of strings representing the set of people whose shortest path is exactly \( K \) hops away from the given vertex (and not fewer). For example, if \( K = 1 \), those are the person's direct friends; if \( K = 2 \), they are the person's friends-of-friends.

```cpp
Set<string> kthLevelFriends(BasicGraph& graph, string v, int k){...}
```

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.