

Section Solution

Discussion Section 1 Solution: Farey Series, Take II

```
void generateFareySeries(Vector<fraction>& series,
                        fraction left, fraction right, int n) {
    fraction mediant = {
        left.numerator + right.numerator,
        left.denominator + right.denominator
    };
    if (mediant.denominator > n) return;
    generateFareySeries(series, left, mediant, n);
    series.add(mediatant);
    generateFareySeries(series, mediant, right, n);
}

Vector<fraction> generateFareySeries(int n) {
    Vector<fraction> series;
    fraction zero = {0, 1};
    fraction one = {1, 1};
    generateFareySeries(series, zero, one, n);
    return series;
}
```

Discussion Problem Solution 2: NEWS Mazes

```
bool pathExists(Grid<rule>& maze, coord start, coord finish) {
    Set<coord> visited;
    return pathExists(maze, Any, start, finish, visited);
}

bool pathExists(Grid<rule>& maze, rule prevRule,
                coord curr, coord finish, Set<coord>& visited) {

    if (coordOffLimits(maze, curr, visited)) return false;
    if (curr == finish) return true;

    rule currRule = maze[curr.row][curr.col];
    Vector<rule> possibilities;
    switch (currRule) {
        case North: possibilities.add(North); break;
        case East: possibilities.add(East); break;
        case West: possibilities.add(West); break;
        case South: possibilities.add(South); break;
        case Bridge: possibilities.add(prevRule); break;
        case Any: possibilities.add(North);
                 possibilities.add(East);
                 possibilities.add(West);
                 possibilities.add(South);
                 break;
    }

    if (currRule != Bridge) visited.add(curr);
}
```

```

    for (int i = 0; i < possibilities.size(); i++) {
        if (solutionExists(maze, possibilities[i],
                           nextCoord(curr, possibilities[i]),
                           finish, visited) {
            return true;
        }
    }

    visited.remove(curr);
    return false;
}

coord nextCoord(coord c, rule r) {
    coord next = c;
    switch (r) {
        case North: next.row++; break;
        case East: next.col++; break;
        case West: next.col--; break;
        case South: next.row--; break;
    }

    return next;
}

bool coordOffLimits(Grid<rule>& maze, coord curr, Set<coord>& visited) {
    return
        curr.row < 0 || curr.row >= maze.numRows() ||
        curr.col < 0 || curr.col >= maze.numCols() ||
        maze[curr.row][curr.col] == OffLimits || visited.contains(curr);
}

```

Lab Problem Solution 1: Generating Anagrams

```

bool findAnagramWithFixedPrefix(
    string prefix, string rest, Lexicon& lex, Vector<string>& words) {
    if (!lex.containsPrefix(prefix)) return false;
    if (lex.containsWord(prefix) && prefix.length() >= 4) {
        if (rest == "" || findAnagram(rest, lex, words)) {
            words.add(prefix);
            return true;
        }
    }

    for (int i = 0; i < rest.length(); i++) {
        if (findAnagramWithFixedPrefix(prefix + rest[i],
                                       rest.substr(0, i) + rest.substr(i + 1),
                                       lex, words)) {
            return true;
        }
    }

    return false;
}

bool findAnagram(string letters, Lexicon& lex, Vector<string>& words) {
    return findAnagramWithFixedPrefix("", letters, lex, words);
}

```