

## CS106X Practice Solution

---

### Solution 1: Acronyms

```
void ReadIntoMap(istream& in, Map<Vector<string> >& map)
{
    Scanner s;
    s.setSpaceOption(Scanner::IgnoreSpaces);

    while (true) {
        string line;
        getline(in, line);
        if (in.fail()) break;
        s.setInput(line);
        string acronym = "";
        while (s.hasMoreTokens())
            acronym += s.nextToken()[0];
        map[acronym].add(line); // [] creates empty vector if needed
    }
}

double PercentConfusing(Map<Vector<string> >& map)
{
    Map<Vector<string> >::Iterator itr = map.iterator();
    int numConfusing = 0;

    while (itr.hasNext()) {
        string key = itr.next();
        if (map[key].size() > 1)
            numConfusing++;
    }

    return double(numConfusing)/map.size();
}
```

### Solution 2: Pascal's Travels

```
int NumPaths(Grid<int>& board)
{
    return NumPaths(board, 0, 0);
}

int NumPaths(Grid<int>& board, int row, int col)
{
    if ((row == board.numRows() - 1) &&
        (col == board.numCols() - 1)) return 1;

    if (!OnBoard(board, row, col)) return 0;
    if (board[row][col] == 0) return 0;

    int hop = board[row][col];
    board[row][col] = 0;
    int count = NumPaths(board, row + hop, col) +
                NumPaths(board, row - hop, col) +
                NumPaths(board, row, col + hop) +
                NumPaths(board, row, col - hop);
}
```

```

    board[row][col] = hop;
    return count;
}

```

### Solution 3: Sanitizing Strings

```

string Sanitize(string workingString, Vector<string>& substrings)
{
    string shortest = workingString;
    for (int i = 0; i < substrings.size(); i++) {
        string substring = substrings[i];
        int found = 0;
        while (true) {
            found = workingString.find(substring, found);
            if (found == string::npos) break;
            string reduction = Sanitize(workingString.substr(0, found) +
                                       workingString.substr(found + substring.size()),
                                       substrings);
            if (reduction.size() < shortest.size()) shortest = reduction;
            found = found + 1;
        }
    }
    return shortest;
}

```

### Solution 4: Stretching Lists

```

void Stretch(node *list)
{
    while (list != NULL) {
        int value = list->value;
        for (int i = 0; i < value - 1; i++) {
            node *repeat = new node;
            repeat->value = value;
            repeat->next = list->next;
            list->next = repeat;
            list = repeat; // advance one
        }
        list = list->next;
    }
}

```