CS106X
Winter 2017

Chris Gregg

Section Handout #3 Solutions

If you have any questions about the solutions to the problems in this handout, feel free to reach out to your section leader, Aaron, or Chris for more information.

1. Partitionable

```
bool partitionableHelper(Vector<int> &rest, int sum1, int sum2) {
  if (rest.isEmpty()) {
    return sum1 == sum2;
  } else {
    int n = rest[0];
    rest.remove(0); // choose
    // explore putting n in either partition
    bool answer = partitionableHelper(rest, sum1 + n, sum2)
               || partitionableHelper(rest, sum1, sum2 + n);
    rest.insert(0, n); // un-choose
    return answer;
  }
}

bool partitionable(Vector<int>& v) {
  return partitionableHelper(v, 0, 0);
}
```

2. Make Change

```
void makeChangeHelper(int amount, Vector<int> &coins, Vector<int> &chosen) {
  if (coins.isEmpty()) {
    if (amount == 0) {
      cout << chosen << endl;
    }
  } else {
    int coin = coins[0];
    coins.remove(0);                                // choose a coin
    for (int i = 0; i <= (amount / coin); i++) {    // explore all coins of this coin
      chosen += i;
      makeChangeHelper(amount - (i * coin), coins, chosen);
      chosen.remove(chosen.size() - 1);
    }
    coins.insert(0, coin);                          // un-choose a coin
  }
}

void makeChange(int amount, Vector<int> &coins) {
  Vector<int> chosen;
  makeChangeHelper(amount, coins, chosen);
}
```

3. Print Squares

```
void printSquaresHelper(int n, int min, Set<int> &chosen) {
  if (n == 0) {
    cout << chosen << endl;
  } else {
    int max = (int) sqrt(n); // valid choices go up to sqrt(n)
    for (int i = min; i <= max; i++) {
      chosen.add(i);                                  // choose
      printSquaresHelper(n - (i * i), i + 1, chosen);   // explore
      chosen.remove(i);                               // un-choose
    }
  }
}

void printSquares(int n) {
  Set<int> chosen;
  printSquaresHelper(n, 1, chosen);
}
```

4. Longest Common Subsequence

```
string longestCommonSubsequence(string &s1, string &s2) {
  if (s1.length() == 0 || s2.length() == 0) {
    return "";
  } else if (s1[0] == s2[0]) {
    return s1[0] + longestCommonSubsequence(s1.substr(1), s2.substr(1));
  } else {
    string choice1 = longestCommonSubsequence(s1, s2.substr(1));
    string choice2 = longestCommonSubsequence(s1.substr(1), s2);
    if (choice1.length() >= choice2.length()) {
      return choice1;
    } else {
      return choice2;
    }
  }
}
```

5. Ways to Climb

```
void waysToClimbHelper(int stairs, Stack<int> &chosen) {
  if (stairs <= 0) {
    cout << chosen << endl;
  } else {
    chosen.push(1);                           // choose 1
    waysToClimbHelper(stairs - 1, chosen); // explore
    chosen.pop();                             // un-choose
    if (stairs > 1) {
      chosen.push(2);                           // choose 2
      waysToClimbHelper(stairs - 2, chosen); // explore
      chosen.pop();                             // un-choose
    }
  }
}

void waysToClimb(int stairs) {
  Stack<int> chosen;
  waysToClimbHelper(stairs, chosen);
}
```

6. Letter Tiles and Words

```
void gatherWordsHelper(string &prefix, Vector<string> &rects,
                       Lexicon &english, Set<string> &words) {
  if (!english.containsPrefix(prefix)) {
    return;
  } else if (english.contains(prefix)) {
    words.add(prefix);
  } else {
    for (int i = 0; i < rects.size(); i++) {
      string rect = rects[i];
      rects.remove(i); // choose
      // explore both orientations of the tile
      gatherWordsHelper(prefix + rect[0] + rect[1], rects, english, words);
      gatherWordsHelper(prefix + rect[1] + rect[0], rects, english, words);
      rects.insert(i, rect); // un-choose
    }
  }
}

void gatherWords(Vector<string>& rects, Lexicon &english, Set<string> &words) {
  Vector<string> copy = rects;
  gatherWordsHelper("", copy, english, words);
}
```

7. Domino Chaining

```
static bool chainExistsHelper(Vector<Vector<int>> &dominoes, int start, int end) {
  if (start == end) {
    return true;
  } else if (dominoes.isEmpty()) {
    return false; // technically optional! know why?
  } else {
    for (int i = 0; i < dominoes.size(); i++) {
      Vector<int> domino = dominoes[i];
      dominoes.remove(i);  // choose this domino

      // explore both possible orientations of the domino
      if ((domino[0] == start && chainExistsHelper(dominoes, domino[1], end)) ||
           domino[1] == start && chainExistsHelper(dominoes, domino[0], end))) {
          return true;
      }
      dominoes.insert(i, domino); // un-choose this domino
    }
    return false;
  }
}

static bool chainExists(Vector<Vector<int>> &dominoes, int start, int end) {
  Vector<Vector<int>> copy = dominoes; // we need our own copy so we can modify it return
  chainExistsHelper(copy, start, end);
}
```