Homework 9

Brian Knotten, Brett Schreiber, Brian Falkenstein

September 17, 2018

4

This problem is essentially the minimum edit distance problem, but with varying weights on operations. Define input to be 2 strings, A and B, of lengths n and m respectively.

```
D = [n + 1, m + 1]
def MED(A, B):
      for i in range(0, n + 1):                      #initialization
            D[0, i] = 3*i
            D[i, 0] = 4*i
       for i in range(1, n + 1):
            for j in range(1, m + 1):
                  D[i, j] = min(
                           if A[i] == B[i]:   D[i-1, j-1]
                           else:  D[i-1, j-1] + 5
                           D[i-1, j] + 3          #deletion
                           D[i, j-1] + 4          #insertion
                           )
```

17

If it is possible to divide the gems into groups P and Q, then there must exist some subcollection (either P or Q) such that the sum of the gems add up to $L/2$, and which is made up of exactly half the emeralds ($|E|/2$) and half the rubys ($|R|/2$).

```
# Recursive solution
function possible(valueToReach, i, emeraldQuota, rubyQuota):
    # Base cases: leftover value or not enough of each quota.
    if valueToReach == 0:
        return emeraldQuota == 0 && rubyQuota == 0
    else if (emeraldQuota == 0 && rubyQuota == 0) || i == |E| + |R|:
        return false

    if possible(valueToReach, i + 1, emeraldQuota, rubyQuota):
        return true

    else let gem = gems[i]
    if gem is an emerald:
        return possible(valueToReach - value(gem), i + 1, emeraldQuota - 1, rubyQuota)
    else gem is a ruby:
        return possible(valueToReach - value(gem), i + 1, emeraldQuota, rubyQuota - 1)


# Call the function. Is is possible to reach half the sum of the gem values with half the gems?
possible(L/2, 0, |E|/2, |R|/2)
```

```
# Iterative solution
possible[L/2 + 1][|E| + |R|][|E|/2][|R|/2];
for (int i = 0; i <= L/2; i++) {
    for(int j = 0; j < |E| + |R|; j++) {
        for(int k = 0; k <= |E|/2; k++) {
            for(int l = 0; l <= |R|/2; l++) {
                if(i == 0) possible[i][j][k][l] = k == 0 && l == 0;
                else if ((k == 0 && l == 0) || j == |E| + |R|) possible[i][j][k][l] = false;
            }
        }
    }
}


for (int i = L/2; i >= 0; i--) {
    for(int j = |E| + |R|; j >= 0; j++) {
        for(int k = 0; k < |E|/2; k++) {
            for(int l = 0; l < |R|/2; l++ {
                if(possible[i][j + 1][k][l])
                    possible[i][j][k][l] = true;
                else let gem = gems[j];
                if(gem is an emerald)
                    possible[i][j][k][l] = possible[i - value(gem)][j + 1][k - 1][l]
                else(gem is a ruby)
                    possible[i][j][k][l] = possible[i - value(gem)][j + 1][k][l - 1]
            }
        }
    }
}

return possible[L/2][|E| + |R|][|E|/2][|R|/2];
```

18

```
int totalNewlines = n;

int MinimumPenalty[n + 1][totalNewlines + 1];
int max = 0;
for(int i = 0; i <= n; i++) {
    MinimumPenalty[i][numberOfNewlines] = MinimumPenalty[i - 1][numberOfNewlines] - w[i];
    MinimumPenalty[i][numberOfNewlines + 1] = L - w[i];

    if(minumumPenalty[i][numberOfNewlines + 1] < MinimumPenalty[i][numberOfNewlines]) {
        max = max(max, MinumumPenalty[i][numberOfNewlines + 1])
        numberOfNewlines++;
    } else {
        max = max(max, MinimumPenalty[i][numberOfNewlines]);
    }
}

return max;
```