Stamati Morellas
Recitation - Tuesdays 1:10pm
Due 12/8/19

COM S 311 Homework 6

This HW is on dynamic programming.

For each problem, give:

1. The **recurrence**
2. An **iterative algorithm** based on the recurrence.

Your algorithm must **not be recursive or use memoization**. **State the run time**. *Part of the grade depends on the **efficiency***. Please **provide pseudo-code**, your pseudo code need not handle array index out of bounds exceptions. *Each problem is worth **40 points***.

Solutions on the following pages...

[Problem 1]

Given $\{1, x_2, x_3, \ldots x_n\}$ and an integer W:

Find a set $\{w_1, \ldots, w_n\}$ such that $\sum_{i=1}^{n} w_i x_i = W$ and $\sum_{j=1}^{n} w_j$ is minimized

Recurrence:

```
W = target integer
x = {x1, x2, ... ,xn}
w = {w1, w2, ... ,wm}

Initialize all wj = 0
Sort x in order of descending values

// Recurrence relation
opt(wi, xi, W) {
    if (W <= 0) { // if the target integer is zero
      return;
    }
    else if (xi < W) { // if the given x value is 0 < x <= W
      wi = W / xi;
      W = W % xi;
      opt(wi+1, xi+1, W);
    }
    else {
      opt(wi+1, xi+1, W);
    }
}
```

Iterative Algorithm:

```
W = target integer
n = {x1, x2, ... ,xn}
w = {w1, w2, ... ,wn}

Initialize all wj = 0
Sort n in order of descending values

// Iterative algorithm
Subset-Multiply(w, n, W) {
    Array M[n, w]; // construct a matrix with the given info
    For each wj in w[]:
```

```
      Initialize M[0, w_j] = 0
    For each x_i in n[]:
      For each w_i in w[]:
        If (W < w_i):
          opt(w_i, x_i, W) = opt(w_i-1, x_i-1, W)
        Else:
          opt(w_i, x_i, W) = MAX(opt(w_i-1, x_i-1, W), w_i + opt(w_i-1, n_i-1, (W - w_i)))
      Endfor
    Endfor
    Return M[n, w]
}

// Runtime: O(nW)
```

[Problem 2]

Let $U = \{x_1, \dots, x_n\}$. Given some integers T and k, determine if there exists a subset $S[]$ of U such that $S = \{s_0, \dots, s_k\}$ and $\sum_{i=1}^{k} s_i = T$.

Recurrence:

```
T = target integer
U = {x_1, … ,x_n}
S = {s_1, … ,s_k}

Initialize all s_i = 0
Integer j = 1

// Recurrence relation
opt(x_i, T) {
    if (T <= 0) { // if the target integer is 0
      return true;
    }
    else if (x_i < T && j <= k) { // if the given x value is 0 < x_i < T
      s_j = x_i;
      T = T - x_i;
      j++;
      opt(x_i+1, T);
    }
    else {
```

```
      j = 1;
      opt(x_{i+1}, T);
    }
    return false;
}
```

Iterative Algorithm:

```
T = target integer
U = {x_1, … ,x_n}
S = {s_1, … ,s_k}

Initialize all s_i = 0
Initialize boolean result = false

Subset-Sum(U, k, T) {
    For each i in U:
      For each w in S:
        If (T < w) {
          opt(i, T) = opt(i-1, T)
        }
        Else {
          opt(i, T) = MAX(opt(i-1, T), w + opt(i-1, T-w))
        }
      Endfor
    Endfor
}

// Runtime: O(nT)
```

[Problem 3]

You are in a rectangular maze organized in the form of M by N cells/locations. You are starting at the upper left corner (grid location: (1; 1)) and you want to go to the lower right corner (grid location: (M;N)). From any location, you can move either to the right or to the bottom, or go diagonal. I.e., from (i; j) you can move to (i; j + 1) or (i + 1; j) or to (i+1; j +1). Cost of moving right or down is 2, while the cost of moving diagonally is 3. The grid has several cells that contain diamonds of whose value lies between 1 and 10. I.e, if you land in such cells you gain an amount that is equal to the value of the diamond in the cell. Once you reach the destination your earnings is the difference between earnings and costs. **Give an algorithm that will output the maximum amount that you can earn**. *Your algorithm need not give the path.*

Recurrence:

```
Matrix X[M][N] // cost grid representation
Matrix D[M][N] // diamonds grid representation

opt(m, n) {
    if (m == 1 && n == 1) {
      X[M][N] = 0;
    }
    else {
      X[M][N] = MIN(2 + opt(m, n-1), 2 + opt(m-1, n), 3 + opt(m-1, n-1)) - D[m][n]
    }
    return X[M][N]
}
```

Iterative Algorithm:

```
Matrix X[M][N] // cost (distance) of each space on the grid
Matrix D[M][N] // value (diamonds) of each space on the grid

Initialize G[1][1] = 0 - D[1][1]

// Initialize the rest of the vertices to be null
For each int m in M: // from 2 to M (rows)
  For each int n in N: // from 2 to N (columns)
    G[m][n] = NULL;
  Endfor
Endfor

// Iterate through grid
```

```
For each int m in M:
  For each int n in N:
    // check space below
    If G[m+1][n] == NULL:
      G[m+1][n] = G[m][n] + 2 - D[m+1][n];
    Else:
      G[m+1][n] = MIN(G[m][n] + 2, G[m+1][n-1] + 2, G[m][n-1] + 3) - D[m+1][n])

    // check space right
    If G[m][n+1] == NULL:
      G[m][n+1] = G[m][n] + 2 - D[m][n+1];
    Else:
      G[m][n+1] = MIN(G[m][n] + 2, G[m-1][n+1] + 2, G[m-1][n] + 3) - D[m][n+1]

    // check space diagonal
    If G[m+1][n+1] == NULL:
      G[m+1][n+1] = G[m][n] + 3 - D[m+1][n+1]
    Else
      G[m+1][n+1] = MIN(G[m][n+1] + 2, G[m+1][n] + 2, G[m][n] + 3) - D[m+1][n+1]
  Endfor
Endfor

Return G[M][N]

// Runtime: O(N*M)
```