ComS 311
Recitation 3, 2:00 Monday
Homework 6

Sean Gordon

December 7, 2019

1) Recurrence:
combo = null if remainder < 0
combo = [ ] if remainder == 0
combo = min(from(i = 0 $\rightarrow$ n) combo(set, remaining - set[i], list))

Algorithm 1 Find non-negative integers w1, ..., wn.

1:

2) Recurrence:
func = true: spaceLeft == 0 && remainingSum == 0
func = false:(spaceLeft == 0 && remainingSum != 0) || index == U.length
func = func(U, spaceLeft, remainingSum, index+1) ||
func(U, spaceLeft-1, remainingSum - U[index], index+1);

Algorithm 2 Test for subset of U of size k that adds to T.

```
1:  boolean iterFunc(U, T, k){
2:
3:  n = U.length;
4:
5:  //Make a 2d array to map subsets
6:  matrix[ ][ ] = new boolean[n][T+1];      //n, 0→T (not 1→T)
7:
8:  //Make hashmap to track the lengths of each subset that adds to a sum
9:  //key = current sum, value = array of subset lengths
10: Hashmap legths = new HashMap<Integer, List<Integer>>();
11:
12: //Set column 0 to true, as all sums == 0 use empty set
13: for int i = 0; i <= n; i++ do
14:     matrix[i][0] = true;
15: end for
16:
17: //For each number in the set
18: for int i = 0; i < n; i++ do
19:     int number = U[i];
20:
21:     //From 1→T
22:     for int sum = 1; sum <= T; sum++ do
23:         //If this number is too big, grab the val above
24:         if number > sum then
25:             matrix[i][sum] = matrix[i-1][sum];
26:             continue;
27:         end if
28:
```

```
29:
30:         //Decide if # can be added to a prev subset to fit current sum
31:         //Use typical subset-sum lookback
32:         result1 = matrix[i - 1][sum - number];
33:
34:         if result then
35:             //We are adding this to the subset
36:             //Ex: if sum = 14, number = 9, and lengths@5 = [1, 3, 4],
37:             //lengths@14 will now = [2, 4, 5]
38:             lengths@sum = lengths@(sum - number)++;
39:         end if
40:
41:         //If this number won't fit, we don't add it to the subsets, but
42:         //this sum may still be possible, so the matrix should reflect that
43:         result2 = result || matrix[i-1][sum];
44:
45:         matrix[i][j] = result2;
46:     end for
47: end for
48:
49:
50: if ! (matrix[n-1][T]) then
51:     return false;
52: end if
53:
54: //If there is a possible subset, check that there is one of length k
55: list = lengths@T
56:
57: for int i = 0; i < list.length; i++ do
58:     if(list[i]== k) return true;
59: end for
60: return false;
61: }
```

3) Recurrence:

traverse = score if x==M && y==N

traverse = max(traverse(maze, M, N, x, y+1, score-2),
traverse(maze, M, N, x+1, y, score-2),
traverse(maze, M, N, x+1, y+1, score-3))

Algorithm 3 Maximize score in M x N maze

```
1: int iterTraverse(maze, M, N){
2:
3: //Go from left→right, top→bottom,                    [X][X]
4: //looking at the max of cells to left, top left, and top [X][O]
5:
6: for int x = 1; x <= M; x++ do
7:     for int y = 1; y <= N; y++ do
8:         //Find largest score for transitioning to this cell
9:         int score = max(maze[x-1][y] -2, maze[x-1][y-1] -3, maze[x][y-1] -2)
10:
11:        //Replace maze slot with score + maybe diamond,
12:        // as we no longer need it
13:        maze[x][y] += score
14:    end for
15: end for
16:
17: return maze[M][N]
18: }
```