CoinChange.java

```
1   package DynamicProgramming;
2
3   import java.util.*;
4
5   public class CoinChange {
6       //DP Opt:
7       public static int coinChange(int[] coins, int amount) {
8   1       int max = amount + 1;
9
10          int[] dp = new int[max];
11  1       Arrays.fill(dp, max);
12
13          dp[0] = 0;
14  2       for (int i = 1; i <= amount; i++) {
15              for (int x : coins) {
16  2               if (i >= x) {
17  2                   dp[i] = Math.min(dp[i], dp[i - x] + 1);
18                  }
19              }
20          }
21
22  3       return dp[amount] > amount ? -1 : dp[amount];
23      }
24  }
25
26  // //Memory DFS:
27  // class Solution1 {
28  //     public static int coinChange1(int[] coins, int amount) {
29  //         if (amount <= 0) {
30  //             return 0;
31  //         }
32  //
33  //         return coinChangehelper(coins, amount, new int[amount]);
34  //     }
35  //
36  //     public static int coinChangehelper(int[] coins, int rem, int[] count) {
37  //         if (rem < 0) {
38  //             return -1;
39  //         }
40  //
41  //         if (rem == 0) {
42  //             return 0;
43  //         }
44  //
45  //         if (count[rem - 1] != 0) {
46  //             return count[rem - 1];
47  //         }
48  //
49  //         int min = Integer.MAX_VALUE;
50  //         for (int coin : coins) {
51  //             int res = coinChangehelper(coins, rem - coin, count);
52  //             if (res >= 0 && res < min) {
53  //                 min = 1 + res;
54  //             }
55  //         }
56  //         count[rem - 1] = (min == Integer.MAX_VALUE) ? -1 : min;
```

```
57  //         return count[rem - 1];
58  //     }
59  // }
60  // //BFS Opt:
61
62  // class Solution2 {
63  //     public static int coinChange2(int[] coins, int amount) {
64  //         if (amount <= 0) {
65  //             return 0;
66  //         }
67
68  //         Arrays.sort(coins);
69
70  //         Queue<Integer> queue = new LinkedList<>();
71  //         queue.offer(amount);
72
73  //         boolean[] visited = new boolean[amount + 1];
74  //         visited[amount] = true;
75
76  //         int step = 1;
77  //         while (!queue.isEmpty()) {
78  //             int size = queue.size();
79  //             for (int i = 0; i < size; i++) {
80  //                 Integer cur = queue.poll();
81  //                 for (int x : coins) {
82  //                     int target = cur - x;
83  //                     if (target == 0) {
84  //                         return step;
85  //                     }
86  //                     if (target < 0) {
87  //                         break;
88  //                     }
89  //                     if (!visited[target]) {
90  //                         visited[target] = true;
91  //                         queue.offer(target);
92  //                     }
93  //                 }
94  //             }
95
96  //             step++;
97  //         }
98
99  //         return -1;
100 //     }
101 // }
```

Mutations

8	1. Replaced integer addition with subtraction → KILLED
11	1. removed call to java/util/Arrays::fill → KILLED
14	1. negated conditional → KILLED 2. changed conditional boundary → KILLED
16	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
17	1. Replaced integer addition with subtraction → KILLED 2. Replaced integer subtraction with addition → KILLED
22	1. changed conditional boundary → KILLED 2. replaced int return with 0 for DynamicProgramming/CoinChange::coinChange → KILLED 3. negated conditional → KILLED

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- DynamicProgramming.CoinChangeTest.testDP(DynamicProgramming.CoinChangeTest) (0 ms)

Report generated by PIT 1.15.0