University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

March 4, 2020

Homework 8, Problem 4

Problem 4 (20 points) Programming: Electoral College Ties:

Determine how many different ways the Electoral College can result in a 269-269 tie in the 2020 US Presidential election.

How the electoral college works: Each US state plus the District of Columbia has a given number of delegates based on its population. An election is held in each state and the winner of that election receives all of the delegates for that state. The person receiving the largest number of delegates is then the president of the US. (This method has the possibility that the person elected president is not necessarily the person winning the most votes nationally.)

For this problem, you are given a list of the number of votes each state has in the electoral college, and you are asked to compute the number of ways that these votes can be allocated to reach a 269-269 tie. We are assuming that there are only two candidates, and that states allocate all of their votes to one candidate or the other.

Obviously, use dynamic programming. There are lots of different ways of reaching a tie - so many that you will need to use 64-bit integers (e.g., long ints).

The data is available here so you can copy the arrays into your program.

a.) How many ways are there for the electoral college to result in a 269-269 tie.

b.) Find a group of states that can reach exactly 269 votes.

c.) Provide your algorithmic code.

d.) What is the runtime of your algorithm (as a function of the number of states, and of the number of electoral votes).

e.) Provide a justification for items a, b, and d.

Answer:

a: There are 8488240282035 in total ways.

b:

- Minnesota:10
- Michigan:16
- Massachusetts:11

- Maryland:10
- Maine:4
- Louisiana:8
- Kentucky:8
- Kansas:6
- Iowa:6
- Indiana:11
- Illinois:20
- Idaho:4
- Hawaii:4
- Georgia:16
- Florida:29
- District of Columbia:3
- Delaware:3
- Connecticut:7
- Colorado:9
- California:55
- Arkansas:6
- Arizona:11
- Alaska:3
- Alabama:9

c:

```
import java.util.ArrayList;
import java.util.Arrays;


public class electoralCollege  {
    static String[] states = {"Alabama","Alaska","Arizona","Arkansas","Californ
    "Georgia","Hawaii","Idaho","Illinois","Indiana","Iowa","Kansas","Kentucky"
    "Michigan","Minnesota","Mississippi","Missouri","Montana","Nebraska","Nevac
    "New York","North Carolina","North Dakota","Ohio","Oklahoma","Oregon","Pen
```

```
"South Dakota","Tennessee","Texas","Utah","Vermont","Virginia","Washington"
static long[] electoralVotes = { 9, 3, 11, 6, 55, 9, 7, 3, 3, 29, 16, 4, 4,
        6, 10, 3, 5, 6, 4, 14, 5, 29, 15, 3, 18, 7, 7, 20, 4, 9, 3, 11, 38,

static long solutionCount(long[] electoralVotes){
    int K = 269;
    long[] Opt = new long[K + 1];
    Opt[0] = 1;
    for(long num : electoralVotes){
        for(int k = K; k >= 0; k--){
            if(k >= num){
                Opt[k] += Opt[(int) (k - num)];
            }
        }
    }
    return Opt[K]/2;
}


static ArrayList<String> getASolution(long[] electoralVotes){
    int K = 269;
    int n = electoralVotes.length;
    Boolean[][] Opt = new Boolean[n + 1][K + 1];
    for (int i = 0; i <= n; i++)
    {
        Arrays.fill(Opt[i], false);
    }

    Opt[0][0] = true;

    for (int j = 1; j <= n; j++){
        for (int k = 0; k <= K; k ++){
            Opt[j][k] = Opt[j-1][k];
            if (k - electoralVotes[j-1] >= 0){
                int residue = (int) (k - electoralVotes[j-1]);
                Opt[j][k] = (Opt[j-1][k] || Opt[j-1][residue]);
            }
        }
    }

    ArrayList<String> subSet = new ArrayList<String>();
    int verifySum = 0;
    int k = K;
    for (int j = n; j >= 1; j--){
        if(!Opt[j-1][k]){
```

```
                    subSet.add(states[j-1]);
                    verifySum += electoralVotes[j-1];
                    System.out.printf("\item %s:%d\n",
                    states[j-1], electoralVotes[j-1]);
                    k -= electoralVotes[j-1];
                }
            }
            assert verifySum == 269;
            return subSet;
        }
        public static void main(String[] args) {
            System.out.println(solutionCount(electoralVotes));
            System.out.println(getASolution(electoralVotes));
            }
        }
```

d:
define:
number of states = n
number of electoral votes = k

For question a, the runtime is $O(n * k)$
For question b, the runtime is $O(n * k)$

e:
proof for a:
base case:
By defination, when there is no value, the sum from 0 to K can not attain by no value, therefore the Opt value all equal to zero.

induction hypothesis:
For each value s_i in set S, assume the value $s_0 \dots s_{i-1}$ has been correctly determined :
Then when k equal to each value from 0 to K: Opt[k] = Opt[k] + Opt[k - s]

The function apply to each inductive step since for each value k, there are only two possibilities:
(1) value k can be divide into s and k - s, s and k - s can be calculated from value $s_0 \dots s_{i-1}$

(2) value k can be divide into s and k - s, s and k - s cannot be calculated from value $s_0 \dots s_{i-1}$
In either case, the value of Opt[k] (number of solutions to get k from $s_0 \dots s_i$) rely on the correctly precaclulated value Opt[k] and Opt[k - s], regardless k can be attaibed by values from value $s_0 \dots s_{i-1}$ or not, if not, the Opt[k] and Opt[k - s] are all zero, if can attaibed, the value built upon the prior result from $s_0 \dots s_{i-1}$.
Since the algorithm calculate Opt$[0 \dots k]$ from s_0 cummulative to s_n, and for each adding s_i, the algorithm attaines result built opon from correctly precomputed value, then algorithm is garantees to find the number of solutions add up to K.
proof for b:
For each value s_j in set S, assume the value $s_0 \dots s_{j-1}$ has been correctly determined :
Then when k equal to each value from 0 to K: Opt[j][k] = Opt[j-1][k] or Opt[j-1][k-S[j-1]]

The function apply to each inductive step since for each value k, there are only two possibilities: (1) value k can be attaibed with the first j_th value add a precaclulated sum, therefore it Opt[j-1][k-S[j-1]] is must true or value k can already be attained before j_{th} value, Opt[j][k] inherent the true value either from Opt[j-1][k] or Opt[j-1][k-S[j-1]].
(2) value k can not be attaibed with the first j_{th} value add a precaclulated sum, therefore it inherents the impossibility from the result of first j-1 th value to get k; In either case, the value of Opt[k] rely on the correctly precaclulated value Opt[k] and Opt[k - s], regardless k can be attaibed by values from value $s_0 \dots s_{j-1}$, since or operant can automatically inherent the true if it can be attained, if either one of Opt[j-1][k] and Opt[j-1][k-S[j-1]] is true, otherwise, it inherent the false value.
Since the algorithm calculate Opt$[0 \dots k]$ from s_0 cummulative to s_n, and for each adding s_j, the algorithm attaines result built opon from correctly precomputed value, then algorithm is garantees to find the at first j value, wether it can get a sum of k. And the subproblems built up on top of prior value all the way to nth value.

complexity:
Since on each subproblem of subset $s_0 \dots s_i$, the algorithm only traces the result from at most K result, so each subproblem will has K opearations. And there are n subproblem, therefore the time complexity is $O(nK)$.