Tomasz Hulka and Frank Errichiello

CS 401: Algorithms
Program 1: Fun with Sumset Sum

1. Code (key sections)

```
struct ssum_info {

  unsigned long long int total; // total solutions
  unsigned long long int smallest; // size of smallest subset
  unsigned long long int numSmallest; // count of solutions with smallest size
  std::vector<bool> opt; //lexigraphically first shortest subset

};
```

```
 std::vector<std::vector<std::pair<bool, unsigned long long int>>> feasible;
 // - feasible[i][x].first = TRUE if there is a subset
 //  of a[0..i] adding up to x; FALSE otherwise
 //  (once instacne has been solved)
 // - feasible[i][x].second stores the smallest size of the vector adding up to x
```

```
std::vector<std::vector<unsigned long long int>> SUMS(n, std::vector<unsigned long long int>(tgt + 1, 0));

//  SUMS[i][x] = the count of distinct subsets
//  of a[0..i] adding up to x;


std::vector<std::vector<std::vector<bool>>> PRE(n, std::vector<std::vector<bool>>(tgt + 1, empty));


// PRE[i][x] = the lexicographically first subset of a[0..i] that achieves a target of x
// which is stored as a vector V of booleans of length n where V[0...n] = true if it is part of the
// lexicographically first subset



std::vector<std::vector<unsigned long long int>> MIN(n, std::vector<unsigned long long int>(tgt + 1, 0));

// MIN[i][x] stores the number of min-sized subsets from a[0..i] that result in a target of x
```

```
for(i=0; i<n; i++) {
    feasible[i][0].first = true;
    feasible[i][0].second = 0;
    SUMS[i][0] = 0;
    PRE[i][0] = std::vector<bool>(n, false);
    MIN[i][0] = 0;
}


for(x=1; x<=target; x++) {
    if(elems[0].x == x) {
        feasible[0][x].first = true;
        feasible[0][x].second = 1;
        SUMS[0][x]++;
        PRE[0][x][0] = true;
        MIN[0][x] = 1; //singleton
    }

}
```

```
for(i=1; i<n; i++) {

    for(x=1; x<=tgt; x++) {

        bool case1 = feasible[i-1][x].first; //exclude
        bool case2 = x >= elems[i].x && feasible[i-1][x-elems[i].x].first; //include

        if (elems[i].x == x) {
            feasible[i][x].first = true;
            feasible[i][x].second = 1;
            SUMS[i][x] = 1 + SUMS[i-1][x];

            if (case1 && feasible[i-1][x].second == 1) { // if we also have a singleton
              PRE[i][x] = PRE[i-1][x]; // then exclude case is always first
              MIN[i][x] = MIN[i-1][x] + 1; // if we have a singleton before then just + 1

            } else {
              PRE[i][x][i] = true;
              MIN[i][x] = MIN[i-1][x - elems[i].x] + 1;
            }

            // otherwise its a new unique value


        }
```

```
        else {


          if(case1 && case2) {
            feasible[i][x].first = true;
            feasible[i][x].second = std::min(feasible[i-1][x].second, feasible[i-1][x-elems[i].x].second + 1);
            SUMS[i][x] = SUMS[i][x] + SUMS[i-1][x - elems[i].x] + SUMS[i-1][x];

            if (feasible[i-1][x].second == (feasible[i-1][x-elems[i].x].second + 1)) {

              if (PRE[i-1][x - elems[i].x] > PRE[i-1][x]) {
                PRE[i][x] = PRE[i-1][x - elems[i].x];
                PRE[i][x][i] = true;

              } else {

                PRE[i][x] = PRE[i-1][x];
              }

              MIN[i][x] = MIN[i][x] + MIN[i-1][x - elems[i].x] + MIN[i-1][x];


            }

            // inclusion case smaller
            else if ((feasible[i-1][x-elems[i].x].second + 1) < feasible[i-1][x].second) {
              PRE[i][x] = PRE[i-1][x - elems[i].x];
              PRE[i][x][i] = true;
              MIN[i][x] = MIN[i-1][x - elems[i].x] + MIN[i][x];
            }
```

```
            else {
              PRE[i][x] = PRE[i-1][x];
              MIN[i][x] = MIN[i-1][x];

            }
          }

          else if (case1) {
            feasible[i][x].first = true;
            feasible[i][x].second = feasible[i-1][x].second;
            SUMS[i][x] += SUMS[i-1][x];
            PRE[i][x] = PRE[i-1][x];
            MIN[i][x] = MIN[i-1][x];

          }

          else if (case2) {
            feasible[i][x].first = true;
            feasible[i][x].second = feasible[i-1][x-elems[i].x].second + 1;
            SUMS[i][x] += SUMS[i-1][x - elems[i].x];
            PRE[i][x] = PRE[i-1][x - elems[i].x];
            PRE[i][x][i] = true;
            // the min ways for the include case is the min ways for the 'leftover' and min ways of getting x
            MIN[i][x] = MIN[i-1][x - elems[i].x] + MIN[i][x];

          }
        }
      }

      // otherwise, feasible[0][x] remains false
    }
    done = true;
```

The runtime of the program is O(nT) , where n is the problem size/ set length and T is the target. There are a few linear time operations and then one loop from i = 1….n-1, and for each iteration we loop through x = 1….T targets and do *only* O(1) operations using a dynamic programming approach.

2. Calculating distinct subsets

Our logic of keeping track of our distinct sums includes a 2D array of integers called SUMS that keep track of each element and the number of distinct ways to reach each possible target. See second image above.

First we initialize the entries for a target of 0 in the for i = [0...n-1], then the for t = [1...tgt] loop checks if there is a single element that adds up to the target and that is updated.

The sums are calculated when calculating through the feasible table.

Our if first state sees if it is a single element as well but this time our I value is shifting so we add one to the sums of our past element.

In the else part of this statement we are calculating for the include and exclude case and the case they are both true.

It makes more sense if the include and exclude cases are explained first.

Exclude: Our sum at our X for the current I value SUMS[i][x] is added to the sum of our past value at x SUMS[i-1][x] because we are excluding our current I. Thus, looking like SUMS[i][x] + SUMS[i-1][x]

Include: Our sum at our X for the current I value SUMS[i][x] is added to the sum of our past value at of the current target – value include SUMS[i-1][x-elems[i].x] because we are including our past I. Thus, looking like SUMS[i][x] + SUMS[i-1][x-elems[i].x]

Include and Exclude: We implement both additions to the current I value of and our target. Which looks like SUMS[i][x] + SUMS[i-1][x]+ SUMS[i-1][x-elems[i].x]

The value stored at sums[n-1][tgt] equals the number of distinct solutions

3. Determining the minimum sized subset

For determining the minimum sized subset, we changed feasible to store a pair which holds a bool and an int. The integer is what stores the size of the minimum size subset for the current target x for elems [0 to i]. Thus, our final value at Feasible[n-1][tgt].second stores our shortest subset size for the true target. This is done above in our loop when calculating feasible as well.

For the first two loops

The first two loops for feasible.second are used to set for if our target is 0 all our shortest path is equal to 0. The second loop checks if there is a single element that adds up to the target and that path is then increased by 1.

In the else part of this statement we are calculating for the include and exclude case and the case they are both true.

Include and Exclude: In this case we see find the smaller between the include which has the plus one as its included and excluded and set feasible[i][x].second equal to the minimum.

Exclude: Our shortest path at our X for the current I value which is feasible[i][x].second is set to the value of our past value or i-1 which is feasible[i-1][x].second

Include: Our shortest path at our X for the current I value feasible[i][x].second is now set equal to the past element and because we are including our past I(The + 1 is included here because the length is then updated). Thus, feasible[i-1][x-elems[i].x].second + 1

4. Capturing the lexicographically first min-sized subset

The logic behind calculating our min sized subset uses a 2d array called PRE

```
std::vector<std::vector<std::vector<bool>>> PRE(n, std::vector<std::vector<bool>>(tgt + 1, empty));


// PRE[i][x] = the lexicographically first subset of a[0..i] that achieves a target of x
// which is stored as a vector V of booleans of length n where V[0...n] = true if it is part of the
// lexicographically first subset
```

Which calculates the lexicographically first subset for all subsets yielding T of the first i elements.

Example: pre[2][3] is the first lexicographical subset considering elements 1,2,3 for indexes 0-2 that sum up to 3. Since its a vector of bool, if any value is true, include that index.

The calculations for pre can be found above, the main piece that matters is the case that we have a tie breaker which always ends up being the exclude case.

5. Data for 269 < electoral.txt

```
franke@DESKTOP-39M9MQL:/mnt/c/Users/Frank Errichiello/Desktop/School/UIC/Junior Year/1st Semester/CS 401/Programs/Program 1$ ./ssum 269 < electoral.txt

    Target sum of 269 is FEASIBLE!

    Number of distinct solutions:16976480564070
    Size of smallest satisfying subset:      11
    Number of min-sized satisfying subsets:   1
    Lexicographically first min-sized solution:

    {CA, FL, GA, IL, MI, NY, NC, OH, PA, TX, VA}
```

6. Data for 220 < purple.txt

```
franke@DESKTOP-39M9MQL:/mnt/c/Users/Frank Errichiello/Desktop/School/UIC/Junior Year/1st Semester/CS 401/Programs/Program 1T$ ./ssum 220 < purple.txt

    Target sum of 220 is FEASIBLE!

    Number of distinct solutions:       9958625
    Size of smallest satisfying subset:      13
    Number of min-sized satisfying subsets:  57
    Lexicographically first min-sized solution:

    {AZ, CO, FL, GA, IN, MI, MN, NJ, NC, OH, PA, TX, VA}
```

7. Data for 121 < purple.txt

```
franke@DESKTOP-39M9MQL:/mnt/c/Users/Frank Errichiello/Desktop/School/UIC/Junior Year/1st Semester/CS 401/Programs/Program 1T$ ./ssum 121 < purple.txt

    Target sum of 121 is FEASIBLE!

    Number of distinct solutions:       9958625
    Size of smallest satisfying subset:       5
    Number of min-sized satisfying subsets:   2
    Lexicographically first min-sized solution:

    {FL, GA, OH, PA, TX}
```