CS 401 - HW4 - Subset Sum Problem
Gregory N. Schmit

1. Source code:

```
/*
 * Gregory N. Schmit
 * CS401, Spring 2017
 * HW4 - Subset Sum Problem
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

struct element {
    char name[11];
    unsigned int value;
};

struct possible {
    short int is_evaluated;
    short int is_possible_incl;
    short int is_possible_excl;
    unsigned long int subsets_incl;
    unsigned long int subsets_excl;
    struct possible *next_incl;
    struct possible *next_excl;
    struct possible *n_parent_incl;
    struct possible *n_parent_excl;
};

void print_matrix(size_t N, size_t T, struct possible sm[][T]) {
    int i, j;
    printf("possible matrix:\n\n T        N
(p_incl/p_excl/ss_incl/ss_excl/n_parent_incl/n_parent_excl)\n\n");
    for (i=T-1; i>=0; i--) {
        printf("%2i    ", i);
        for (j=0; j<N; j++) {
            printf("%i/%i/%lu/%lu/%i/%i   ", sm[j][i].is_possible_incl,
                sm[j][i].is_possible_excl,
                sm[j][i].subsets_incl,
                sm[j][i].subsets_excl,
                !!sm[j][i].n_parent_incl,
                !!sm[j][i].n_parent_excl);
        }
        printf("\n\n\n");
```

```
    }
    // printf("evaluated matrix:\n\nT            N\n\n");
    // for (i=T-1; i>=0; i--) {
    //     printf("%2i    ", i);
    //     for (j=0; j<N; j++) {
    //         printf("%i    ", sm[j][i].is_evaluated);
    //     }
    //     printf("\n\n");
    // }
}

int ssum_recursive(struct element *set, size_t cur_n, unsigned int t) {
    // Check if there exists a subset of s(0..n-1) that sums to t.
    if (!set) { return 0; }
    if (!t) { return 1; }
    if (cur_n == 0) { return (t == set[cur_n].value); }
    if (set[cur_n].value <= t && ssum_recursive(set, cur_n-1, t-(set[cur_n].value))) {
return 1; }
    if (ssum_recursive(set, cur_n-1, t)) { return 1; }
    return 0;
}

int ssum_nparent_recursive(struct possible *node) {
    if (node->next_incl) {
        node->next_incl->n_parent_incl = node;
        ssum_nparent_recursive(node->next_incl);
    }
    if (node->next_excl) {
        node->next_excl->n_parent_excl = node;
        ssum_nparent_recursive(node->next_excl);
    }
    return 1;
}

int ssum_complete_matrix(size_t N, size_t T, struct possible sm[][T], struct element
*set) {
    if (!sm || !set) { return 0; }
    // base case:
    int i, j;
    for (i=0; i<N; i++) { // target=0
        sm[i][0].is_possible_excl = 1;
        if (i) { sm[i][0].next_excl = &sm[i-1][0]; }
        sm[i][0].subsets_excl = 1;
        if (!set[i].value) {
            sm[i][0].is_possible_incl = 1;
            if (i) { sm[i][0].next_incl = &sm[i-1][0]; }
            sm[i][0].subsets_incl = 1;
        }
```

```
        sm[i][0].is_evaluated = 1;
    }
    for (i=1; i<T; i++) { // singleton
        if (set[0].value == i) {
            sm[0][i].is_possible_incl = 1;
            sm[0][i].subsets_incl = 1;
        }
        sm[0][i].is_evaluated = 1;
    }
    // now for the real work:
    for (i=1; i<N; i++) {
        for (j=1; j<T; j++) {
            if (sm[i-1][j].is_possible_incl || sm[i-1][j].is_possible_excl) {
                sm[i][j].is_possible_excl = 1;
                sm[i][j].next_excl = &sm[i-1][j];
                sm[i][j].subsets_excl+=(sm[i][j].next_excl->subsets_incl +
sm[i][j].next_excl->subsets_excl);
            }
            if (j-(int)set[i].value >= 0 && (sm[i-1][j-set[i].value].is_possible_incl
|| sm[i-1][j-set[i].value].is_possible_excl)) {
                sm[i][j].is_possible_incl = 1;
                sm[i][j].next_incl = &sm[i-1][j-set[i].value];
                sm[i][j].subsets_incl+=(sm[i][j].next_incl->subsets_incl +
sm[i][j].next_incl->subsets_excl);
            }
            sm[i][j].is_evaluated = 1;
        }
    }
    return 1;
}

int main(int argc, char **argv) {
    // sanity
    if (argc != 3) {
        printf("Sanity Failed: use the following format: `sset <N_MAX> <TARGET>`\n");
        return 0;
    }
    int N = strtol(argv[1], 0, 10);
    if (N < 0 || N > 4000000) {
        printf("Sanity Failed: N_MAX must be greater than or equal to 0, and less than
4000000.\n");
        return 0;
    }

    // Read in problem set
    struct element *set;
    set = malloc(N*sizeof(*set));
    int i, j;
```

```
    int T;
    int value;
    char name[11];
    i = 0;
    while (i < N && scanf("%i %s", &value, name)) {
        // project precondition is that there exist no name duplicates, so if
        //  there is one between adjacent elements, I can assume end of file.
        if (i > 0 && !strcmp(name, set[i-1].name)) { break; }
        set[i].value = value;
        strcpy(set[i].name, name);
        i++;
    }
    N = i;
    T = strtol(argv[2], 0, 10) + 1;
    // for (i=0; i<N; i++) {
    //     printf("%i - %s\n", set[i].value, set[i].name);
    // }

    // Build table of possibles
    struct possible (*sm)[T];
    sm = calloc(N*T, sizeof(**sm));
    ssum_complete_matrix(N, T, sm, set);
    //print_matrix(N, T, sm);

    ssum_nparent_recursive(&sm[N-1][T-1]);

    struct possible *tmp = 0;
    int tmp_n;
    for (i=0; i<N; i++) {
        for (j=T-1; j>=0; j--) {
            if (sm[i][j].n_parent_incl || sm[i][j].n_parent_excl) {
                tmp = &sm[i][j];
                tmp_n = i;
                goto getout; // https://xkcd.com/292/
            }
        }
    }
getout:;

    printf("Target sum of %i is ", T-1);
    if (sm[N-1][T-1].is_possible_incl || sm[N-1][T-1].is_possible_excl) {
        printf("FEASIBLE!\n\n");
    } else {
        printf("NOT FEASIBLE!\n\n");
    }
    printf("Number of distinct solutions:   %lu\n", sm[N-1][T-1].subsets_incl +
sm[N-1][T-1].subsets_excl);
```

```
    if (tmp) {
        int subset[N];
        subset[0] = (tmp - &sm[0][0]) / T;
        int sl = 1;
        while(tmp && (tmp->n_parent_incl || tmp->n_parent_excl)) {
            if (tmp->n_parent_incl) {
                tmp = tmp->n_parent_incl;
                subset[sl] = (tmp - &sm[0][0]) / T;
                sl++;
            } else {
                tmp = tmp->n_parent_excl;
            }
        }
        printf("first subset: { ");
        for (i=0; i<sl; i++) {
            printf("%s", set[subset[i]].name);
            if (i < sl-1) { printf(", "); }
        }
        printf(" }\n");
    }

    free(set);
    free(sm);
}
```

2. Explanation (number of subsets):

I built some meta data into the subset matrix that tracks how many subsets exist including that particular element and how many exist excluding that particular element. The end result is that for any target value T for a set of size N, you can go to sm[T][N-1] and pull the structure elements .subsets_incl and .subsets_excl and add them to find the total number of subsets that sum to the target value.

3. Explanation (min subset)

I could not find a reasonable way to find the minimum size of the subsets outside of hacky things like tracking the size of each subset (a lot of space was consumed attempting this method) or iterating through the subsets (a lot of time was consumed attempting this method). I didn't implement it, but if I had more time, I think I would try tracking the minimum size subset when I build the matrix. I could also track how many of those exist. I just couldn't figure out the details to make it work.

4. Explanation (Lex. first min-sized subset):

I did not get to this part. I was able to figure out how to get the first lexicographically first subset and I printed that. I did that by recursively riding the tree downward (from the sm[T][N-1] element and going to possible/feasible subsets) and memoizing the parent of those elements. I then found the first subset that had a parent (meaning it would end up summing to T) and traversing the tree upwards from there, favoring including the elements riding upwards. I ended up basically doing a depth traversal but only down the last branch.

5. Data (electoral.txt target=269):

I had to comment out my recursive solution that finds the lexicographically first subset:

```
$ ./ssum 100 269 < electoral.txt
Target sum of 269 is FEASIBLE!

Number of distinct solutions:   16976480564070
```

6. Data (purple.txt target=220):

```
$ ./ssum 100 220 < purple.txt
Target sum of 220 is FEASIBLE!

Number of distinct solutions:   9958625
first subset: { AZ, AR, CO, DC, DE, FL, GA, IN, KY, IA, LA, ME, MI, MN, MS,
MO, NE, NV, NH, NJ, NC, OR, WV, WI }
```

7. Data (purple.txt target=121):

```
$ ./ssum 100 121 < purple.txt
Target sum of 121 is FEASIBLE!

Number of distinct solutions:   9958625
first subset: { AZ, AR, CO, DC, DE, FL, GA, IN, KY, IA, LA, ME, NE, NH }
```