Report

CS 401 Subset Sum Programming Assignment

Authors: Tirth Patel, Lavithra Mysore

1) **Computing distinct subsets:**
   - ➔ The feasible array is modified and populated such that each cell stores the number of subsets that can be formed that sum up to the respective column value.

```
feasible[n][target];
```

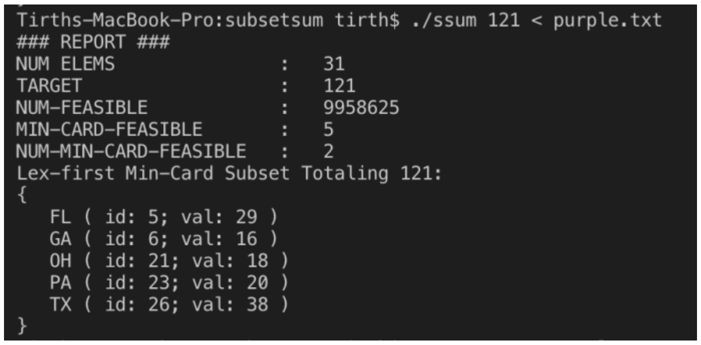
stores the total subsets available summing up target. Returning number of subsets will be O(nT). Line 110-129

2) **determining the minimum sized subset:**
   - ➔ To computer the minimum sized subset, we have a map of vector that keeps that stores indices of valid subsets. From that we return size of vector in a map that has minimum size. Line 187-195

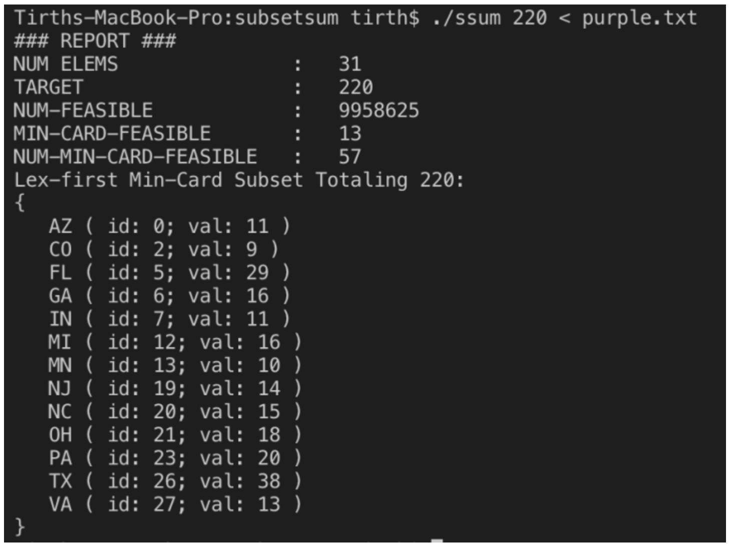
3) **lexicographically first min-sized subset totaling the target:**
   - ➔ We have used a map to store the lexicographical min-sized subset as map outputs in sorted order. First, we go over the map of vector and find the one which has the least size. As soon as we find one, we add the indices from that vector to lexicographical map as key and the value corresponding to that index from the elems vector and break. To print the name of state, we directly print the name from the elems vector corresponding to the index. Line number 198-208.

More in comments in the source code. // ssum.cpp file

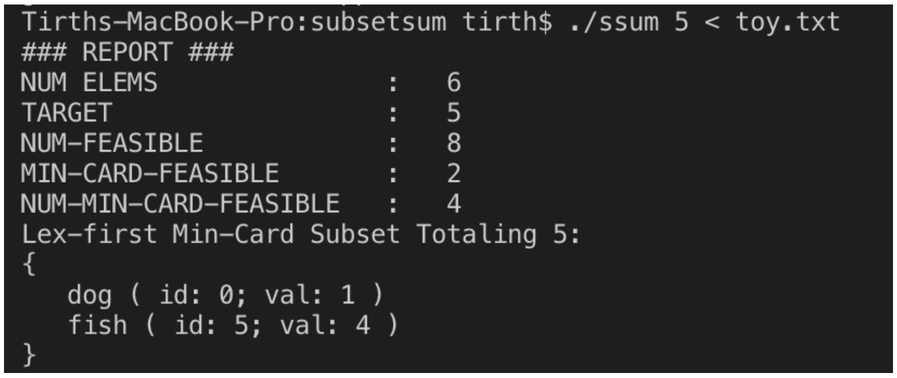
Screenshot of the results



```
Tirths-MacBook-Pro:subsetsum tirth$ ./ssum 121 < purple.txt
### REPORT ###
NUM ELEMS               :   31
TARGET                  :   121
NUM-FEASIBLE            :   9958625
MIN-CARD-FEASIBLE       :   5
NUM-MIN-CARD-FEASIBLE   :   2
Lex-first Min-Card Subset Totaling 121:
{
   FL ( id: 5; val: 29 )
   GA ( id: 6; val: 16 )
   OH ( id: 21; val: 18 )
   PA ( id: 23; val: 20 )
   TX ( id: 26; val: 38 )
}
```



```
Tirths-MacBook-Pro:subsetsum tirth$ ./ssum 220 < purple.txt
### REPORT ###
NUM ELEMS               :   31
TARGET                  :   220
NUM-FEASIBLE            :   9958625
MIN-CARD-FEASIBLE       :   13
NUM-MIN-CARD-FEASIBLE   :   57
Lex-first Min-Card Subset Totaling 220:
{
   AZ ( id: 0; val: 11 )
   CO ( id: 2; val: 9 )
   FL ( id: 5; val: 29 )
   GA ( id: 6; val: 16 )
   IN ( id: 7; val: 11 )
   MI ( id: 12; val: 16 )
   MN ( id: 13; val: 10 )
   NJ ( id: 19; val: 14 )
   NC ( id: 20; val: 15 )
   OH ( id: 21; val: 18 )
   PA ( id: 23; val: 20 )
   TX ( id: 26; val: 38 )
   VA ( id: 27; val: 13 )
}
```



```
Tirths-MacBook-Pro:subsetsum tirth$ ./ssum 5 < toy.txt
### REPORT ###
NUM ELEMS               :   6
TARGET                  :   5
NUM-FEASIBLE            :   8
MIN-CARD-FEASIBLE       :   2
NUM-MIN-CARD-FEASIBLE   :   4
Lex-first Min-Card Subset Totaling 5:
{
   dog ( id: 0; val: 1 )
   fish ( id: 5; val: 4 )
}
```

```
//Source code
#include <stdio.h>


#include <stdlib.h>
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <utility>
#include <queue>
#include <map>
#include<unordered_map>

using namespace std;
struct ssum_elem {
  unsigned int x;
  std::string name;
};

class Triplet {
public:
  int row ;
  int col ;
  string idxPath;   // for keeping track of the indices on the path to target sum

   Triplet(int row, int col , string idxPath)
  {
    this->row=row;
    this->col=col;
    this->idxPath=idxPath;
  }
};

class ssum_instance {
   int total_nodes;
    unsigned int target=0;

    std::vector<std::vector<int>> feasible;
      // vector<vector<int>> _feasible;
    // feasible[i][x] = TRUE if there is a subset
    //  of a[0..i] adding up to x; FALSE otherwise
    //  (once instacne has been solved)
    //
    int done=false;    // flag indicating if dp has been run or
                       //   not on this instance

    public:
    std::vector<ssum_elem> elems;
    // Function:  read_elems
    // Description:  reads elements from standard-input;
    //   Format:  sequence of <number> <name> pairs where
    //      <number> is non-negative int and
    //      <name> is a string associated with element
    void read_elems( std::istream &stream) {
        ssum_elem e;

        elems.clear();
        // while(std::cin >> e.x && std::cin >> e.name) {
        while(stream>> e.x && stream >> e.name) {
            elems.push_back(e);
        }
        done = false;
    }
    // Function:  solve
    // Desc:  populates dynamic programming table of
    //    calling object for specified target sum.
    //    Returns true/false depending on whether the
    //    target sum is achievalble or not.
    //    Table remains intact after call.
    //    Object can be reused for alternative target sums.
    int solve(unsigned int tgt,unsigned long long int  &distinctSubSets,int &minCardFeasible, int
&numMinCardFeasible,map<int,int> &lexicographical) {
        unsigned int n = elems.size();
        unsigned int i, x;
        vector<int> allValidSubsets;   // to store the size of all valid subsets (subsets that add up to target sum )
        unordered_map<int, vector<int>> mapOfVector;      // storing the indices of valid subsets
        if(target == tgt && done)
        {
            return feasible[n-1][tgt];
        }
        target = tgt;
        feasible = std::vector<std::vector<int>>(n+1, std::vector<int>(target+1, 0));

        // for(int i = 0 ; i < feasible.size(); i++)
        // {
        //   for(int j =0 ; j < feasible[0].size(); j++)
        //   {
        //     if(j == 0)
        //     {
        //       feasible[i][j] = 1;
        //     }
        //     else if (i ==0)
        //     {
        //       feasible[i][j] = 0;
        //     }
        //     else{
        //       if(feasible[i-1][j]){
        //         feasible[i][j] = 1;
        //       }
        //       else if ( j >= elems[i-1].x && feasible[i-1][j-elems[i-1].x])
```

```
    //       {
    //           feasible[i][j] = 1;
    //       }
    //     }
    //   }
    // }

// The modified approach is to have a 2d vector of intgers rather than boolean.
// It is similar to previous feasible matrix with the chanfge that each cell stores the number of subsets that sum to
//corresponding column value.. This gives O(nT) to retrieve the number of possible subsets.
 feasible[0][0] = 1;
for (int i = 1; i <= tgt; i++)
    feasible[0][i] = 0;
for (int i = 1; i <= n; i++)
    feasible[i][0] = 1;

for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= tgt; j++)
    {
        if (elems[i - 1].x > j)  // this is the exclude case
            feasible[i][j] = feasible[i - 1][j];
        else   // exclude case
        {
          // feasible [i][j] stores the include value + exclude value
          //  feasible[i - 1][j - elems[i - 1].x] is the include case
            feasible[i][j] = feasible[i - 1][j] + feasible[i - 1][j - elems[i - 1].x];
        }
    }
}
    done = true;
    distinctSubSets= feasible[n][target];  // get the distinct Subsets value

    // return if target sum is not possible
    if(feasible[n][target] == 0)
      return false;
```

```
         Triplet t =  Triplet(n,target,"");    // creating the instance of class Triplet storin total subsets totalling to target
value
          queue<Triplet> q ;

          q.push(t);          // pushing that triplet in a queue
         //  int smallestSubset = INT_MAX;
           int k =0;
         vector<int> sample;    // this vector stores the indices of valid subsets

        while(q.size() > 0 )
        {
          Triplet t_instance = q.front();    // get the first triplet
          q.pop();                          // remove the first triplet

          if(t_instance.row ==0 || t_instance.col==0)    // we add when row or col is 0, that is path is valid
          {
           //  distinctSubSets++;
           stringstream iss( t_instance.idxPath );
           int number;
           // convert the string of path to vector
           // example path of "5 6 21 23 26" will be adddded to sample vector sample = {5,6,21,23,26}
           while ( iss >> number ){
             sample.push_back( number );
           }

           mapOfVector[k] = sample;   // sample is a vector containing the indices of the elems
                                     // storing each valid subset's indices in a  map(key,value) with value as a vector
           k++;
           sample.clear();
          }
          else
          {
             if(t_instance.col >= elems[t_instance.row-1].x) // we include
            {
              int inc = feasible[t_instance.row-1][t_instance.col - elems[t_instance.row-1].x];   // include case
              if(inc != 0)  // add the triplet with include case to queue
              {
                string path = to_string(t_instance.row-1)+ " " + t_instance.idxPath;
                q.push(Triplet(t_instance.row-1, t_instance.col - elems[t_instance.row-1].x, path));  //test-- 5 6 21 23 26 ->
path

              }
            }
            int exc = feasible[t_instance.row-1][t_instance.col];  // exclude case
            if(exc != 0)  // if exclude case stores a number greater than 0, we add to queue
            {
              q.push(Triplet(t_instance.row-1,t_instance.col,t_instance.idxPath)); // add the triplet with exclude case to
queue
            }
          }
        } // end while q.size

             // storing the size of all subsets in a vector
           for ( const auto &it : mapOfVector ) {
                   allValidSubsets.push_back(it.second.size());
             }
         // min size of the subset totaling target sum
         // finding the mimum size subset from allValidSubsets vector
         minCardFeasible = *min_element(allValidSubsets.begin(), allValidSubsets.end());

        // count the total number of subsets with minimum size
         numMinCardFeasible = count(allValidSubsets.begin(),allValidSubsets.end(),minCardFeasible);

         // Here, looping through map and storing the index and values in lexicographical map.
           for(const auto &it: mapOfVector){
              if(it.second.size() == minCardFeasible)
                 {
                   for(auto i : it.second)
                   {
                     lexicographical[i] = elems[i].x;
                   }
                   break;
                   cout <<endl;
                 }
        }
    // cout << feasible[n][target];
      return feasible[n][target];    // total subsets
  }



};  // end class

/**
* usage:  ssum  <target> < <input-file>
*
*
* input file format:
*
*     sequence of non-negative-int, string pairs
*
*     example:

  12 alice
  9  bob
  22 cathy
  12 doug

* such a file specifies a collection of 4 integers: 12, 9, 22, 12
* "named" alice, bob, cathy and doug.
*/
int main(int argc, char *argv[]) {
 unsigned int target;
 ssum_instance ssi;
 unsigned long long int distinctSubSets =0;
 int minCardFeasible = 0 ;
 int numMinCardFeasible = 0 ;
 map<int,int> lexicographical;  // using map to store values in lexicographical manner as elements are ordered in
map

 if(argc != 2) {
  fprintf(stderr, "one cmd-line arg expected: target sum\n");
```

```
   return 0;
  }
  if(sscanf(argv[1], "%u", &target) != 1) {
   fprintf(stderr, "bad argument '%s'\n", argv[1]);
   fprintf(stderr, "   Expected unsigned integer\n");
   return 0;
  }

  ssi.read_elems(std::cin);

//  unsigned long long int total =
ssi.solve(target,distinctSubSets,minCardFeasible,numMinCardFeasible,lexicographical);
//  cout << total;
  // cout<<total;

  if(ssi.solve(target,distinctSubSets,minCardFeasible,numMinCardFeasible,lexicographical) > 0 ) {
   // std::cout << "HOORAY!  Apparently, the target sum of " <<
   //   target << " is achievable\n";
   // std::cout << "  How you ask?  Sorry, we just know it is possible...\n";
   cout<<"### " <<"REPORT" << " ###"<<endl;
   cout<<"NUM ELEMS                :   "<< ssi.elems.size()<<endl;
   cout<<"TARGET                   :   " << target<<endl;
   cout<<"NUM-FEASIBLE             :   "<< distinctSubSets << "\n";
   cout<<"MIN-CARD-FEASIBLE        :   "<< minCardFeasible<< "\n";
   cout<<"NUM-MIN-CARD-FEASIBLE   :   " << numMinCardFeasible<< "\n";
   cout<<"Lex-first Min-Card Subset Totaling "<<target<< ":"<<endl;
   cout<<"{"  <<endl;           ;   // print the lexicogrphical map

  for(auto it = lexicographical.begin(); it != lexicographical.end(); ++it){
   cout<< "   "<< ssi.elems[it->first].name<< " ( "<<"id: "<<it->first <<  "; "<<"val: "<<it->second << " )"<<endl;
  }

  cout<< "}"<<endl;


  }
  else {
   std::cout << "SORRY!  Apparently, the target sum of " <<
```

```
      target << " is NOT achievable\n";
  }


}
```