ACM ICPC 2015 国内予選　問題 D 500 円玉預金

【考察】

- 動的計画法で解く。
- i 番目のお店を出てきた時点で、手持ちの小銭（100 円玉以下のコイン）の総額が m、取得した 500 円玉の枚数を num、これまでのコスト（購入した商品の合計金額）を cost とする。今後、残りのお店を回った時に最終的に得られる 500 円玉の枚数やそれにかかる総コストは、ここまでのお店での状況には左右されず、また、手持ちの小銭の内訳にも影響されない。
- m を index とする配列に、num, cost を格納。
- 商品を購入しない場合、m, num, cost は変化しない。
- 商品を購入する場合（価格 p とする）、千円札で支払う場合のお釣りを change とすると、change = (1000 – p % 1000) % 1000 となる。
  - change >= 500 の時は千円札で支払えば良い。（∵手持ちの小銭を使って 500 円玉を得ても、500 円玉の枚数とコストは千円札で支払った場合と同じであり、千円札で支払った方が手持ちの小銭が増えるので、今後それを用いて 500 円玉を得られる可能性が高くなる。）このとき
    - 500 円玉の枚数は 1 増加
    - コストは p 増加
    - 手持ちの小銭は、(change – 500)円増加
  - change < 500 の時、千円札で支払うと
    - 500 円玉の枚数は変化しない
    - コストは p 増加
    - 手持ちの小銭は change 円増加
  - change < 500 の時、手持ちの小銭が（500 – change）円以上なら、手持ちの小銭から（500 – change）円だけ余分に払うことで 500 円玉を得ることが出来る。このとき
    - 500 円玉の枚数は 1 増加
    - コストは p 増加
    - 手持ちの小銭は、(500 – change）円減少
- 上記の式に基づき、手持ち小銭総額の部分の状態が改善されるようであれば改善していく。

【プログラム例 pd.c】

```
// ACM-ICPC 2015 国内予選 Problem D 500 円貯金
// http://icpc.iisf.or.jp/past-icpc/domestic2015/contest/all_ja.html#section_D
//   Filename:      pd.c
//   Compile:       gcc pd.c
//   Execution:     ./a.out < D0 > D0.result
//   Check:         diff D0.ans D0.result
//   Algorithm:     動的計画法による。各店において、いくら支払うかに応じて
//                  手持ちの 500 円玉未満の小銭の総計、その時点で得られている
//                  500 円玉の総枚数、商品の購入金額の総計を求めていく。
//                  全ての店を回った時点で、500 円玉の最大枚数と、
//                  その枚数を取得する為に支払った金額の最小値をプリントする。
//   （注）コメント内の「小銭」は 100 円玉以下のコインを意味する

#include <stdio.h>

#define NSTORE 100                      // 店数の最大値
#define NSTATE (NSTORE * 499)           // 手持ちの小銭の最大値

typedef struct {
  int num;                              // 500 円玉の枚数
  int cost;                             // 支払い金額
} data_t;

int n;                                  // 店の数
int max_change;                         // 手持ち小銭の最大値
data_t states[NSTATE+1];  // 手持ちの 100 円玉未満の小銭の総和毎に、
                                        // その時点で得ている 500 円玉の最大枚数と
                                        // 最大枚数取得にかかったコストの最小値を
                                        // 格納する配列
data_t newstates[NSTATE+1];             // 動的計画法で次の状態を表す配列
int price[NSTORE];                      // 各お店の商品の価格

int max_num;                            // 取得している 500 円玉の最大枚数
int min_cost;                           // それを取得するのに
                                        // 購入した商品価格合計の最小値

// 100 円玉以下の小銭の手持ち合計金額が m の時に、
// 500 円玉の枚数 num とその為の cost が、
// これまでに求まっているものよりも良ければ、状態を更新する
void update_sub(int m, int num, int cost)
{
  // まず手持ち小銭の合計金額が m の場合のデータを必要に応じて更新
  if(num > newstates[m].num){           // 500 円玉の枚数が増えるので
    newstates[m].num = num;             // 500 円玉の枚数と
    newstates[m].cost = cost;           // それを取得するコストを更新
```

```
  }
  else if(num == newstates[m].num){ // 500 円玉の個数は増えないが
    if(cost < newstates[m].cost)        // コストが下がるので
      newstates[m].cost = cost;                  // コストのみ更新
  }
  // 500 円玉の最大枚数のその時の最小コストを必要に応じて更新
  if(num > max_num){                  // より多くの 500 円玉を取得したので
    max_num = num;                             // 500 円玉の最大枚数と
    min_cost = cost;                           // その時の最小枚数を更新
  }
  else if(num == max_num){                     // 500 円玉の枚数は同じだが
    if(cost < min_cost)                        // コストが下がるので
      min_cost = cost;                         // 最小コストを更新
  }
}

// 動的計画法による状態の更新
// update が呼び出された時点では、newstates と states は同じ内容である。
// このことにより、商品を買わない場合の状態変化もカバーできている。
void update(int p) // p = 商品の価格
{
  int m;              // 100 円玉以下の小銭の手持ち合計
  int next_m;         // 商品を購入してお釣りをもらった結果の手持ち小銭の合計
  data_t next_data;           // その時の 500 円玉の枚数と総コスト
  int change;                 // 千円札で支払った場合のお釣りの合計
  for(m=0; m<=max_change; m++){               // m = 現在の手持ち小銭合計金額
    // この時点で手持ち小銭合計金額が m になるのは不可能な場合は何もしない
    if(states[m].num<0) continue;
    // 千円札で支払う場合のお釣り
    change = (1000 - p % 1000)%1000;
    if(change >= 500){        // 500 円以上お釣りがあるので
      next_data.num = states[m].num+1;        // 500 円玉 1 個増える
      next_data.cost = states[m].cost + p;        // コストも p 円増える
      next_m = m + (change - 500);   // 手持ちの小銭はお釣り－500 円増える
      update_sub(next_m, next_data.num, next_data.cost);          // 状態を更新
    }
    else {                        // 千円札で支払った場合のお釣りは 500 円未満
      next_data.num = states[m].num;          // 500 円玉は増えない
      next_data.cost = states[m].cost + p;        // コストは p 円増える
      next_m = m + change;            // 手持ちの小銭はお釣りの分だけ増える
      update_sub(next_m, next_data.num, next_data.cost);          // 状態を更新
      if(m >= (500-change)){          // 小銭を余分に払ってお釣りを 500 円にする
        next_data.num = states[m].num + 1;  // 500 円玉 1 個増える
        next_data.cost = states[m].cost + p;    // コストも p 円増える
        next_m = m - (500-change);  // 余分に払った分手持ちの小銭が減る
        update_sub(next_m, next_data.num, next_data.cost);     // 状態を更新
```

```
        }
      }
    }
    for(m=0; m<=max_change; m++)                    // 求まった次状態を現状態にする
      states[m] = newstates[m];
}

int main(void)
{
    int i;
    int p;
    for(;;){
      scanf("%d", &n);                              // 店の数 n を入力
      if(n==0) break;                               // n が 0 なら終了
      max_change = n * 499;                 // 手にすることが可能な小銭総額
      for(i=0; i<n; i++)                            // 各商品の価格を読み込む
        scanf("%d", &price[i]);
      for(i=0; i<= max_change; i++)       // とりあえず各状態は到達不能とする
        states[i].num = states[i].cost = -1;
      max_num = 0;                          // 取得した 500 円玉の最大枚数は 0
      min_cost = 0;                         // それにかかる費用も 0
      states[0].num = 0;                    // 手持ち小銭=0, 500 円玉 0 枚、コスト 0 円 よ
り開始
      states[0].cost = 0;
      for(i=0; i<=max_change; i++)        // 現状態 states を newstates にコピー
        newstates[i] = states[i];
      for(i=0; i<n; i++){                   // 各商品に対して
        p = price[i];               // p =  商品の価格
        update(p);
      }
      printf("%d %d¥n", max_num, min_cost);
    }
}
```