ICPC2011 国内予選問題 E　輪番停電計画

概要

- グループは左上角の位置（座標）と右下角の位置（座標）で一意に特定できる。
- 需要表の高さと幅の最大値は 32 なので、左上角の位置は 10 ビット（x 座標 5 ビット、y 座標 5 ビット）、右下角の位置は 10 ビットで指定できるので、合計 20 ビットで指定することが出来る。
- したがって、異なるグループの総数は高々2^{10}≒100 万である。
- グループの消費電力が、全体の不足電力より少ない場合は、そのグループを停電させても電力不足は解消しないので、そのようなグループは採用できない。
- グループの消費電力が、全体の不足電力の 2 倍より少ないい場合は、そのグループを 2 つのグループに分割することは出来ない（∵分割すると少なくとも一方のグループの消費電力は、全体の不足電力より少なくなってしまう）。
- あるグループ G が 2 つのグループ G1 と G2 に分割できたとする。G1 をさらに最大限に細分化した時の G1 内のグループ数を ng1, その時の予備力を sp1 とする。同様に、G2 をさらに最大限に細分化した時の G2 内のグループ数を ng2, その時の予備力を sp2 とする。この時、グループ G を最大限に細分化した時のグループ数を ng, その時の予備力を sp とすると

  ng = ng1 + ng2,　sp = Min(sp1, sp2)

  が成立する。
- この関係を用いると、動的計画法や深さ優先探索により答えを求めることが出来る。
- 深さ優先探索に基づくプログラム例を以下に示す。

［プログラム例］

```
// ICPC 2011 国内予選問題 E
// http://icpc2011.ait.kyushu-u.ac.jp/icpc2011/contest/all_ja.html#section_E
//          Filename = pe1.c
//          Compile:    cc -O2 pe1.c          // 最適化コンパイル
//          Execution:  ./a.out < E0 > E0.result
//          Check:                diff E0.ans E0.result
// 深さ優先探索による解法
//
#include <stdio.h>
#include <stdlib.h>
#define MAXH 32                 // 需要表の高さの最大値
#define MAXW 32                 // 需要表の幅の最大値
#define MAX_GROUPS MAXH*MAXW*MAXH*MAXW
```

```
// 各グループは左上の座標(x0,y0)と右下の座標(x1,y1)で識別する (x0<=x1, y0<=y1)
// 各座標は5ビットで表現できるので、合計20ビットの GroupID でグループを識別する
#define GroupID(x0, y0, x1, y1)     ((x0 << 15) + (y0 << 10) + (x1 << 5) + (y1))

// グループ ID gid から、x0, y0, x1, y1 を求めるマクロ
#define X0(gid)         ((gid)>>15)
#define Y0(gid)         (((gid)>>10) & 31)
#define X1(gid)         (((gid)>>5) & 31)
#define Y1(gid)         ((gid) & 31)

// 2数の最小値を求めるマクロ
#define Min(a, b)       ((a)>(b)?(b):(a))

int h;                  // 需要表の高さ 1 <= h <= 32
int w;                  // 需要表の幅    1 <= w <= 32
int s;                  // 供給力
int fusoku;             // 全体での消費電力の不足数

int map[MAXH][MAXW];            // 需要表  各要素の値は1以上100以下

// 各グループの状態を表す構造体
typedef struct group_type {
   int state;           // -1  使用不可(予備力が負), 0  未訪問, 1  訪問済みで答えが求まっている
   int cons;            // 消費電力計
   int ng;              // このグループの可能な細分化グループ数
   int spare;           // その時の予備力
} group_t;

// 深さ優先探索で各グループの可能な細分化グループ数と予備電力を求める際に用いる構造体
typedef struct ng_spare_pair {
   int ng;              // グループ数
   int spare;           // 予備力
} gsp_t;

group_t gd[MAX_GROUPS];         // グループのデータ

// 深さ優先探索で、グループ gid の可能な細分化グループ数と予備電力を求める
gsp_t dfs(int gid)
{
   // val に答えを入れる。
   //val1 と val2 には、このブループを2分割した時の各々のグループに対する答えを入れる。
   gsp_t val, val1, val2;
   int x0, y0, x1, y1, x, y;
   int g1, g2;
   if(gd[gid].state == 1){          // このグループを訪問済みなら、その時の答えを返す
      val.ng = gd[gid].ng;
      val.spare = gd[gid].spare;
   }
   else {
```

```
    x0 = X0(gid);                    // このグループの左上角と右下角の座標を求める
    y0 = Y0(gid);
    x1 = X1(gid);
    y1 = Y1(gid);
    for(x=x0; x<x1; x++){            // x 座標で2分割する
      g1 = GroupID(x0, y0, x, y1);
      g2 = GroupID(x+1, y0, x1, y1);
      // どちらかのグループの予備力が負なら分割出来ないのでスキップ
      if(gd[g1].state == -1 || gd[g2].state == -1) continue;
      val1 = dfs(g1);                // グループ g1 に対する答えを深さ優先探索で求める
      val2 = dfs(g2);                // グループ g2 に対する答えを深さ優先探索で求める
      // この分割を採用した時の本グループの細分化グループ数と予備力を求める
      // 細分化グループ数は、g1 と g2 の細分化グループ数の和となり
      // 予備力は、g1 と g2 の予備力の小さい方となる
      val.ng = val1.ng+val2.ng;
      val.spare = Min(val1.spare, val2.spare);
      if(gd[gid].ng < val.ng){                  // これまでに求まっているグループ数より多ければ
          gd[gid].ng = val.ng;                  // グループ数を更新し
          gd[gid].spare = val.spare;            // 予備力も更新する
      }
      // これまでに求まっているグループ数と等しい時は、予備力が大きければ予備力を更新する
      else if(gd[gid].ng == val.ng && gd[gid].spare <val.spare){
          gd[gid].spare = val.spare;
      }
    }
    for(y=y0; y<y1; y++){            // y 座標で2分割する
      g1 = GroupID(x0, y0, x1, y);
      g2 = GroupID(x0, y+1, x1, y1);
      if(gd[g1].state == -1 || gd[g2].state == -1) continue;
      val1 = dfs(g1);
      val2 = dfs(g2);
      val.ng = val1.ng+val2.ng;
      val.spare = Min(val1.spare, val2.spare);
      if(gd[gid].ng < val.ng){
          gd[gid].ng = val.ng;
          gd[gid].spare = val.spare;
      }
      else if(gd[gid].ng == val.ng && gd[gid].spare <val.spare){
          gd[gid].spare = val.spare;
      }
    }
    gd[gid].state = 1;               // 本グループに対する答えが求まったので「訪問済み」にし
    val.ng = gd[gid].ng;             // 戻り値をセットする
    val.spare = gd[gid].spare;
  }
  return val;
}

int main()
{
```

```
    int i,j, gid, total;
    int y0, x0, y1, x1, y, x;
    gsp_t ret;

    while(1){
        scanf("%d %d %d", &h, &w, &s);                    // 需要表の高さ、幅、供給力の入力
        if(h==0 && w==0 && s==0) break;                   // それらが全て0なら終了
        total = 0;                                        // 消費電力の合計を求める
        for(i=0; i<h; i++)
            for(j=0; j<w; j++){
                scanf("%d", &map[i][j]);
                total += map[i][j];
            }
        fusoku = total - s;                               // 不足電力
        // 可能なグループに対して情報をセットする
        for(gid=0; gid<MAX_GROUPS; gid++){
            // gid に対する左上角と右下角の座標を求める
            x0 = X0(gid);
            y0 = Y0(gid);
            x1 = X1(gid);
            y1 = Y1(gid);
            // マップの範囲外や左上角と右下角の組になっていない場合は無視(スキップ)
            if(x0 >= w || x1 >= w || x0 > x1) continue;
            if(y0 >= h || y1 >= h || y0 > y1) continue;
            gd[gid].cons = 0;                             //   このグループの消費電力を求める
            for(y=y0; y<=y1; y++)
                for(x=x0; x<=x1; x++)
                    gd[gid].cons += map[y][x];
            gd[gid].ng = 1;                   // まだ細分化していないのでグループ数は 1 とする
            // 本グループを停電させた時の予備力を求める
            gd[gid].spare = gd[gid].cons - fusoku;
            // その予備力が負になる場合、本グループの利用は不可
            if(gd[gid].spare < 0) gd[gid].state = -1;
            // その予備力が不足電力以下の場合、このグループは細分化出来ない ->
            //     このグループに対する答えが求まっている
            else if(gd[gid].spare < fusoku) gd[gid].state = 1;
            // それ以外の場合は細分化可能なので未訪問とする(深さ優先探索で答えを見つける)
            else gd[gid].state = 0;
        }
        // 全体を一つにしたグループに対して、深さ優先探索で細分化し答えを求める
        ret = dfs(GroupID(0,0, w-1, h-1));
        printf("%d %d¥n",ret.ng, ret.spare);          // 答えの印刷
    }
}
```