[Problem D]　ぐらぐら

一般に、n 個の物体があり、i 番目の物体の重心の x 座標を x_i, 重さを w_i とすると、全体の重心の x 座標と重さ w は $x = (\sum_{i=1}^{n} x_i \cdot w_i) / w, \quad w = \sum_{i=1}^{n} w_i$ となる。

良さそうな方法は思いつかなかった。アイデア募集中！！！
ので、少し強引に解いている。

入力データの読み込みは、w と h を scanf で読み込み、getchar でその行の行末コードを読み込み、ピースのデータは fgets で char 型の 2 次元配列に読み込んでいる。

不要かもしれないが、各ピースにユニークなピース番号をふっておいた方が後の処理での混乱が少なくなると思われるので、別途整数型の 2 次元配列を用意し、そこにユニークなピース番号をふっている。

各ピースごとに、
1)　そのピースを構成する 4 つのブロックの座標
2)　自分の下側に接しているピースの番号
3)　下側に接している面の最左 x 座標と最右 x 座標
4)　上側に接しているピース（本ピースが直接支えているピース）の個数（最大で 4）
5)　上側に接しているピースの番号
6)　自分自身を含めて支えているピース全体の重量と重心の x 座標
を求めている。

各ピースの支えあう状態は、地面に接しているピースを根とする木構造で表現できる（と問題に書かれている）ので、各ピースに対して 1)～5)を求めることにより木構造を表現する。次に、地面に接しているピース（木構造の根）から post order の深さ優先探索で 6)を求めていく。各ピースにおいて 6)が求まった時点で、重心の x 座標が 3)で求めた最左 x 座標と最右 x 座標の間に入っているかをチェックし、入っていなければ「UNSTABLE」と判定する。途中で「UNSTABLE」と判定されることなく根節点に辿り着き、根節点も「UNSTABLE」でなければ「STABLE」と判定する。

プログラム例

```
// ACM-ICPC 2010 Japan Online Contest Problem D
// http://icpc2010.honiden.nii.ac.jp/domestic-contest/problems#section_D
//      ファイル名: pd.c
//      コンパイル方法: cc pd.c
//      実行方法: ./a.out < D0 > D0.result  など
//      チェック方法: diff D0.ans D0.result  など
```

```
//
// アルゴリズム概要
// 1. 各ブロックにユニークなピース番号をふる
// 2. 各ピースに属しているブロックとそれらに隣接しているブロックを調べ
//        下側のピース番号と下側ピースに接している最左、最右 x 座標
//        上側のピース（本ピースが支えているピース）の個数とそれらのピース番号
//     を求める
// 3. 地面に接しているピースから post order で、各ピースが支える全重量、重心位置を
//     を求め、安定性をチェックする。不安定なものが見つかれば「不安定」、
//     全てが安定なら「安定」と判定

#include <stdio.h>

#define WMAX 10             // 幅 w の最大値
#define HMAX 60             // 高さ h の最大値
#define PMAX (WMAX*HMAX/4)  // ピースの個数の最大値

struct piece_t {
  int xpos[4];      // ピースを構成するブロックの x 座標
  int ypos[4];      // ピースを構成するブロックの x 座標
  int down;         // 下側のピース番号
  int xl; // 下側ピースに接している面の最左 x 座標
  int xr; // 下側ピースに接している面の最右 x 座標
  int nup;          // 上側のピース(本ピースが支えているピース）の個数
  int up[4];        // 上側のピース(本ピースが支えているピース）番号
  int weight;       // 自分自身を含め支えているピース全体の重量
  double xcg;       // 自分自身を含め支えているピース全体の重心の x 座標
};
int w,h;
char map[HMAX][WMAX+2];         // 入力データを文字列で記憶する配列
int pnum[HMAX][WMAX];           // ピース番号を覚える配列
struct piece_t pdata[PMAX];     // 各ピースのデータを格納する配列
int np;                         // ピースの個数
// 隣接位置のオフセット（右、上、左、下）
int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};

// 深さ優先探索により安定性をチェックし、安定なら 0 、不安定なら 1 を返す
int dfs(int pn)
{
  int i;
  int tw;
  double cg;
  tw = 0;
  cg = 0;
  for(i=0; i< pdata[pn-1].nup; i++){          // 自分が支えている各ピースに対し
    if(dfs(pdata[pn-1].up[i])) return 1;      // それが不安定なら「不安定」と判定
    // 安定な場合は、その重みと重心計算のためのモーメントを足し込む
    tw += pdata[pdata[pn-1].up[i]-1].weight;
    cg += pdata[pdata[pn-1].up[i]-1].weight * pdata[pdata[pn-1].up[i]-1].xcg;
  }
  for(i=0; i<4; i++)            // このピースを構成する各ブロックに対して
    cg += pdata[pn-1].xpos[i]+0.5;  // ブロックのモーメントを足し込む
  tw += 4;                      // ブロックは 4 個なので、重みとして 4 を足す
  pdata[pn-1].weight = tw; // このピースが支えている重み（自分自身も含む）
```

```
    pdata[pn-1].xcg = cg/tw;  // このピースが支えている重みの重心の x 座標
    // 重心位置が下側面の最左位置以下あるいは最右位置以上ならば不安定
    if(pdata[pn-1].xcg <= pdata[pn-1].xl || pdata[pn-1].xcg >= pdata[pn-1].xr){
      return 1;                    // 不安定
    }
    return 0;                      // 安定
}

// ピースに関する情報を作成
void gen_pdata()
{
    int i,j,k;
    int x, y;
    for(i=0; i<np; i++){                    // 各ピースに対して情報をゲット
      pdata[i].xl = w+1;                    // 下のピースに接する最左 x 座標
      pdata[i].xr = -1;                     // 下のピースに接する最右 x 座標
      pdata[i].nup = 0;                     // 上のピースの個数
      for(j=0; j<4; j++){                   // このピースの各ブロックについて調べる
        x = pdata[i].xpos[j];  // ブロックの x 座標
        y = pdata[i].ypos[j];  // ブロックの y 座標
        if(y>0){                            // 地面には接していない場合
          if(pnum[y-1][x] !=0 && pnum[y-1][x] != i+1){
            // 自分と異なるピースのブロックが下にあるケース
            if(x < pdata[i].xl) pdata[i].xl = x;          // より左にあれば最左 x 座標を更新
            if(x+1 > pdata[i].xr) pdata[i].xr = x+1;      // より右にあれば最右 x 座標を更新
            pdata[i].down = pnum[y-1][x];                 // 下にあるピースの番号をセット
          }
        }
        else {          // y == 0  すなわち地面に接している
          if(x < pdata[i].xl) pdata[i].xl = x;            // より左にあれば最左 x 座標を更新
          if(x+1 > pdata[i].xr) pdata[i].xr = x+1;        // より右にあれば最右 x 座標を更新
          pdata[i].down = 0;            // 下にあるピースの番号として 0（地面）をセット
        }
        if(y < h-1){ // 最も上の場所ではない（この上に他のピースがある可能性有り）
          if(pnum[y+1][x] >= 1 && pnum[y+1][x] != i+1){
            // 自分と異なるピースのブロックが上にあるケース
            int match = 0;
            for(k=0; k<pdata[i].nup; k++){      // そのピースが登録されているかチェック
              if(pdata[i].up[k] == pnum[y+1][x]){         // 既に登録されている
                match = 1; // 既登録であることを示すフラグ
                break;
              }
            }
            if(match == 0){ // まだ登録されていなかったので
              pdata[i].up[pdata[i].nup++] = pnum[y+1][x];         // 登録する
            }
          }
        }
      }
    }
}

// ピースを見つけユニークなピース番号を振る
void find_pieces()
{
```

```
  int x, y;
  char myx;
  int qf, qr;
  np = 0;                       // ピースの個数を 0 に初期化
  for(y=0; y<h; y++)
    for(x=0; x<w; x++)
      pnum[y][x] = 0;           // ピース番号を格納する配列を初期化
  for(y=0; y<h; y++){
    for(x=0; x<w; x++){
      myx = map[y][x];
      if(myx == '.') continue;  // (x,y)にブロックは無い
      if(pnum[y][x] > 0) continue;  // (x,y)のブロックにはピース番号が振られている
      np++;                     // 新しいピースが見つかったのでピースの個数を 1 個増やし
      pnum[y][x] = np;          // そのブロックのピース番号とする
      pdata[np-1].xpos[0] = x;  // そのピースの 0 番目のブロックの x 座標をセット
      pdata[np-1].ypos[0] = y;  // そのピースの 0 番目のブロックの y 座標をセット
      qf=0; qr=1;
      while(qr<4){              // そのピースを構成する残り 3 つのブロックを見つける
        int xx, yy, nx, ny, d;
        xx = pdata[np-1].xpos[qf];  // qf 番目のブロック
        yy = pdata[np-1].ypos[qf];
        qf++;
        for(d=0; d<4; d++){     // そのブロックに隣接する 4 つのブロックを調べる
          nx = xx + dx[d];      // 隣接位置の座標
          ny = yy + dy[d];
          if(map[ny][nx]=='.') continue;    // その場所にブロックは無い
          if(pnum[ny][nx] > 0) continue;    // 既にピース番号が決まっている
          if(map[ny][nx] != myx) continue;  // 今のピースとは異なる
          pdata[np-1].xpos[qr] = nx;        // このブロックはこのピースに属する
          pdata[np-1].ypos[qr] = ny;
          pnum[ny][nx] = np;
          qr++;
        }
      }
    }
  }
}

int main()
{
  char line[WMAX+1];    // ブロックデータを読み込むための配列
  int x, y, val;
  while(1){
    scanf("%d %d", &w, &h);     // w と h の入力
    if(w==0 && h==0) break;     // 両方 0 なら終了
    getchar();                  // w と h の行の行末の '¥n' を読みとばす
    for(y=h-1; y>=0; y--){
      fgets(map[y],WMAX+2,stdin);       // 1 行分のデータを文字列として読み込む
    }
    find_pieces();          // ピースを見つけユニークなピース番号を振る
    gen_pdata();            // ピース間の上下関係より木構造を構築
    if(dfs(1))              // post order の深さ優先探索で安定性を判定
      printf("UNSTABLE¥n");     // dfs が 1 を返せば不安定
    else
      printf("STABLE¥n");       // dfs が 0 を返せば安定
```

```
    }
}
```