ACM ICPC2016 国内予選　問題 E　3D プリント

http://icpc.iisf.or.jp/past-icpc/domestic2016/problems/all_ja.html#section_E

- 「ある立方体が 2 個の立方体と重なるとき、その 2 個の立方体は重ならない」ので 3 個以上の立方体が重なることはない（図 1）。したがって、重なりによる表面積の減少は、2 つの立方体の重なりによる表面積の減少を積算していけば良い（図 2）。

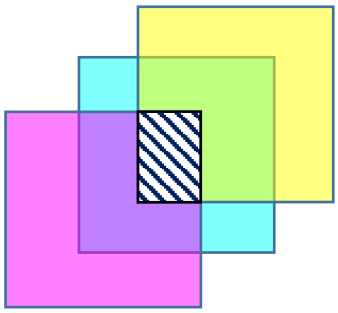

図 2：3 個の立方体の重なり（網掛け部分）は無い

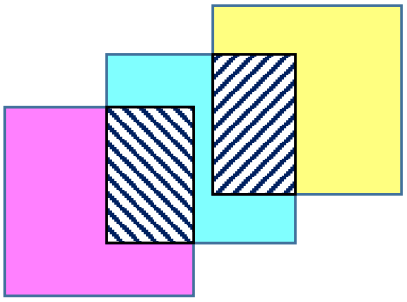

図 1：2 個の立方体の重なり（網掛け部分）を積算していけば良い

- 例えば 2 個の立方体（頂点の最小座標値を$(x,y,z),(x',y',z')$とする）の重なりにより、上から見た部分の表面積は $(s-|x'-x|)\times(s-|y'-y|)$ だけ減少する（図 3）。

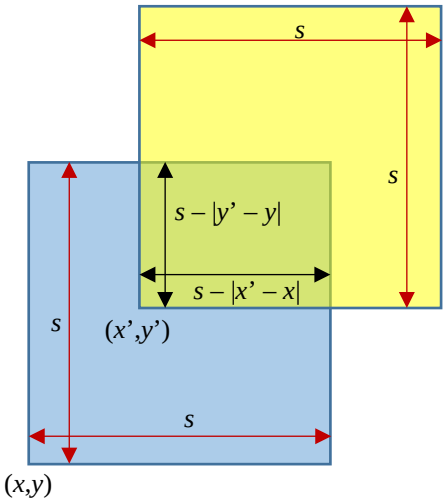

図 3：重なりで減少する表面積

- したがって、2 個の立方体の重なりにより減少する表面積は、2 個の立方体が重なっている部分の直方体の表面積となる（図 4）。 2 つの立方体の頂点の最小座標を各々$(x,y,z),(x',y',z')$、立方体の一辺の長さを s とすると、重なり部分の直方体の表面積は、

$$2\times\{(s-|x'-x|)\times(s-|y'-y|)+(s-|y'-y|)\times(s-|z'-z|)+(s-|z'-z|)\times(s-|x'-x|)\}$$ である。

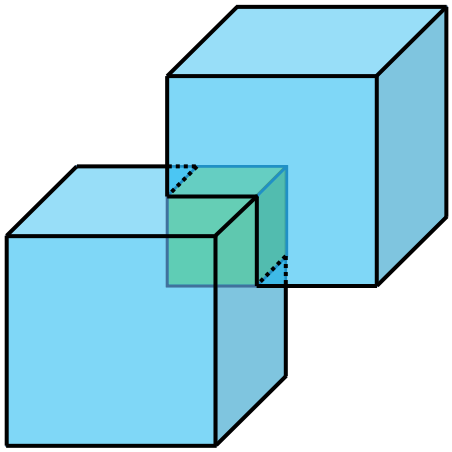

図 4：立方体の重なり

- 立方体を頂点、立方体同士の重なりを辺、辺の重みを重なり部分の直方体の表面積とするグラフを考える。「各立方体は 0 個, 1 個 または 2 個の立方体と重なることがあるが, 3 個以上とは重ならない」ので、このグラフの頂点の位数は高々2 である。したがって、このグラフの連結成分は
  - 1 個の頂点
  - 直線状に連結する分岐やループの無いグラフ
  - 単純ループ

  となる。
- k 個以上の節点からなる連結成分に対して、連続する k 個の頂点を結ぶ辺の重みの合計（但し、k 個頂点からなるループの場合は終点と始点を結ぶ辺の重みも加算する）が最大（重なりにより減少する表面積が最大→構成される多面体の表面積が最小）のものが求める解を与える。

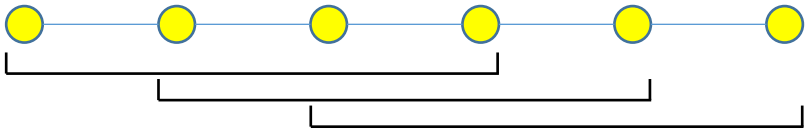

図 5：直線状に連結する連結成分で k=4 の場合（黒実線の範囲の最大値）

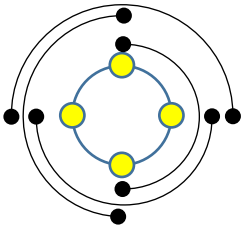

図 7：ループの連結成分で k がループ長よりも短い場合（黒実線の範囲の最大値）

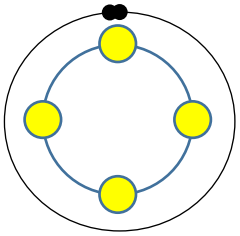

図 6：k がループ長と等しい場合

【pe1.c】

```
/*************************************************************
 * ACM ICPC2016 国内予選 問題 E 3D プリント
 *   Filename: pe1.c
 *   Compile:   cc -Wall pe1.c
 *   Check:      ./a.out < E0 > E0.result; diff E0.result E0.ans
 *   Algorithm:
 *        ・連結立方体の表面積は、各立方体の表面積の和から連結部分の
 *          直方体の表面積を引いたものとなる
 *        ・各立方体は高々２個の立方体としか重ならないので、
 *          立方体を頂点、重なりを辺とするグラフの連結成分は
 *          直線状（分岐なしツリー）か単純なループとなる
 *        ・各連結成分に対して、k 個連結している部分グラフの連結部分の
 *          表面積が最大になるものを求めれば良い
 *        ・求める答えは k 個の立方体の表面積から
 *          連結部分の表面積を引いたものとなる
 *************************************************************/

#include <stdio.h>
#define MAXN 2000                                   // 立方体の候補地総数の最大値
#define ABS(x) (((x)>=0)?(x):(-(x)))      // 絶対値を求めるマクロ
// #define DEBUG_CONNECTION       // 立方体の重複状況のデバッグ用
// #define DEBUG_LINE                // 連結が直線状になる場合のデバッグ用
// #define DEBUG_LOOP                // 連結がループになる場合のデバッグ用

// 立方体に関する情報を格納する構造体
typedef struct {
    int x;                  // 頂点の x 座標の最小値
    int y;                  // 頂点の y 座標の最小値
    int z;                  // 頂点の z 座標の最小値
    int nc;                 // 連結数
    int cc[2];              // 連結している立方体の番号
    int sa[2];              // 連結部分の表面積
    int checked;            // チェック済みフラグ
} cube_t;
int n;                      // 立方体の候補地総数
int k;                      // 求める多面体を構成する立方体の個数
int s;                      // 立方体の一辺の長さ
cube_t cube[MAXN];          // 立方体の情報を入れる配列
```

```
int connected[MAXN+1];   // 連結部分の表面積のリスト

#ifdef DEBUG_CONNECTION
void print_cubes(void); // 立方体の情報をプリント（デバッグ用）する関数
#endif

// 2個の立方体の連結部分の表面積を求める（未連結の時は0）
int sarea(int i, int j)
{
    int dx, dy, dz;       // 連結部分の直方体の大きさ
    dx = s - ABS(cube[i].x - cube[j].x);          // x軸方向の重なり幅
    dy = s - ABS(cube[i].y - cube[j].y);          // y軸方向の重なり幅
    dz = s - ABS(cube[i].z - cube[j].z);          // z軸方向の重なり幅
    if(dx > 0 && dy > 0 && dz > 0){               // 連結しているなら
        return 2*(dx*dy + dy*dz + dz*dx);         // 連結部分の表面積を返す
    }
    return 0;                 // 連結していない場合は0を返す
}

// start から始まる直線状の連結立方体に対して、
// k個の立方体による 多面体の最小表面積を返す
// 立方体の個数がk個未満のときは－1を返す
int proc_line(int start)
{
    int len;                          // 連結立方体の長さ（連結部分の個数）
    int index;                        // 次の立方体番号を格納している場所(0 or 1)
    int next, previous;               // 次の立方体と手前の立方体
    int kcsa, kcsamax;                // k個の立方体の連結部分の表面積とその最大値
    int first, last;                  // k連結立方体の先頭と最後
    len = 0;
    next = cube[start].cc[0];     // 隣の立方体番号
    previous = start;                 // 処理中の立方体番号
    cube[start].checked = 1;      // この立方体をチェック済みにする
    #ifdef DEBUG_LINE
    printf("line: %d (%d) ", start, cube[start].sa[0]);
    #endif
    connected[len++] = cube[start].sa[0];         // 連結部分の表面積を格納
    // 次の立方体の連結数が2の間繰り返す（直線を可能な限り順に伸ばす）
    while(cube[next].nc == 2){
```

```
        cube[next].checked = 1;                    // この立方体をチェック済みにする
        if(cube[next].cc[0] == previous)           // 0 番目の連結立方体から来たので
            index = 1;                             // 次の立方体は 1 番目の連結立方体となる
        else                                       // 1 番目の連結立方体から来たので
            index = 0;                             // 次の立方体は 0 番目の連結立方体となる
        #ifdef DEBUG_LINE
        printf("%d (%d) ", next, cube[next].sa[index]);
        #endif
        connected[len++] = cube[next].sa[index];   // 次の連結部分の表面積を格納
        previous = next;                           // この立方体の番号を保存
        next = cube[next].cc[index];               // 次の立方体にセット
    }
    cube[next].checked = 1;          // 線分の最後の立方体をチェック済みとする
    #ifdef DEBUG_LINE
    printf("%d¥n", next);
    #endif
    // 直線状の連結立方体からなる多面体が求まったので、
    // この中から k 個の立方体で構成される部分多面体の
    // 連結部分の表面積の最大値を求める
    if(len < k-1) return -1;      // k 個連結できない場合は − 1 を返す
    kcsa = 0;
    // まず先頭から k 個の立方体の連結部分(k-1 個)の表面積の和を求める
    for(first=0; first < k-1; first++){
        kcsa += connected[first];
    }
    kcsamax = kcsa;
    // k 連結立方体を後ろへ一つずつずらしながら連結部分の表面積を求め
    // 必要に応じて最大値を更新する
    first=0; last = k-1;
    while(last<len){                  // 直線の最後に達するまで繰り返す
        kcsa += connected[last] - connected[first];       // 一つ後ろの表面積
        if(kcsa > kcsamax) kcsamax = kcsa;                // 最大値の更新
        first++; last++;              // 一つ後ろへずらす
    }
    return kcsamax;
}

// start から始まるループ状の連結立方体に対して
// k 個の立方体による 多面体の最小表面積を返す
```

```
// 立方体の個数が k 個未満のときは－ 1 を返す
int proc_loop(int start)
{
    int len;                  // ループの長さ
    int index;                // 次の立方体番号を格納している場所(0 or 1)
    int next, previous; // 次の立方体と手前の立方体
    int kcsa, kcsamax;   // k 個連結立方体の連結部分の表面積とその最大値
    int first, last;      // k 個連結立方体の先頭と最後の立方体
    len = 0;
    next = cube[start].cc[0];    // 隣の立方体番号
    previous = start;             // 処理中の立方体番号
    cube[start].checked = 1;     // この立方体をチェック済みにする
    #ifdef DEBUG_LOOP
    printf("loop: %d (%d) ", start, cube[start].sa[0]);
    #endif
    connected[len++] = cube[start].sa[0];          // 連結部分の表面積を格納
    while(cube[next].checked == 0){                 // 未チェックの間繰り返す
        cube[next].checked = 1;                     // チェック済みとする
        if(cube[next].cc[0] == previous)           // 0 番目の隣接立方体から来た
            index = 1;              // 1 番目の隣接立方体が次の立方体となる
        else                  // 1 番目の隣接立方体から来たので
            index = 0;              // 次の立方体は 0 番目の立方体となる
        #ifdef DEBUG_LOOP
        printf("%d (%d) ", next, cube[next].sa[index]);
        #endif
        connected[len++] = cube[next].sa[index];   // 次の連結部分の表面積を格納
        previous = next;                           // この立方体の番号を保存
        next = cube[next].cc[index];                // 次の立方体にセット
    }
    #ifdef DEBUG_LOOP
    printf("¥n");
    #endif
    // ループを抽出出来た
    if(len < k) return -1;              // k 個立方体が含まれていない
    kcsa = kcsamax = 0;                 // 連結部分の表面積を 0 に初期化
    if(len == k){                       // ループが丁度 k 個の立方体からなる時は
        for(first=0; first<len; first++){    // 全連結部分が含まれる
            kcsa += connected[first];
        }
```

```
        kcsamax = kcsa;
    }
    else {                      // ループ長が k より長い場合,
        // 最初の k 個の立方体の連結部分の表面積を求める
        for(first=0; first<k-1; first++){
            kcsa += connected[first];
        }
        kcsamax = kcsa;
        // ループ上で先頭を一つずつ後ろへずらしながら
        // 連結部分の表面積を求めていく
        first = 0; last = k-1;
        do {
            // 一つ後ろへずらした場合の連結部分の表面積を求める
            kcsa += (connected[last] - connected[first]);
            if(kcsa > kcsamax) kcsamax = kcsa;          // 最大値の更新
            first = (first+1)%len;   // サイクリックに一つずらす
            last = (last+1)%len;
        } while(first > 0);              // 一周するまで繰り返す
    }
    return kcsamax;
}

int main(void)
{
    int i, j, maxsa;
    int line_sa;
    int loop_sa;
    while(1){
        scanf("%d%d%d", &n, &k, &s);      // n, k, s の入力
        if(n==0 && k==0 && s==0) break; // 全て 0 なら終了
        maxsa = -1;
        for(i=0; i<n; i++){                    // 各立方体に対して
            scanf("%d%d%d", &cube[i].x, &cube[i].y, &cube[i].z); // x,y,z の入力
            cube[i].nc = 0;                    // 連結数は 0 に初期化
            cube[i].checked = 0;             // フラグは未チェックに初期化
        }
        // 各立方体相互の連結状態と連結部分の表面積を求める
        int sa;                          // 連結部分の直方体の表面積
        for(i=0; i<n-1; i++){
```

```
        for(j=i+1; j<n; j++){
            if((sa = sarea(i, j))>0){          // 立方体 i と j は連結
                cube[i].cc[cube[i].nc] = j;             // cube[i]に j を登録
                cube[i].sa[cube[i].nc++] = sa;
                cube[j].cc[cube[j].nc] = i;             // cube[j]に i を登録
                cube[j].sa[cube[j].nc++] = sa;
            }
        }
    }
    #ifdef DEBUG_CONNECTION
    print_cubes();
    #endif
    // 立方体が 1 個の場合は、その表面積が求める答え
    if(k==1){
        printf("%d¥n", 6*s*s);
        continue;
    }
    // line
    for(i=0; i<n; i++){
        // 連結数が 1 で未チェックの立方体は線分の先頭となる
        if(cube[i].nc == 1 && cube[i].checked == 0){
            line_sa = proc_line(i);
            if(line_sa == -1)          // 線分の立方体数が k 未満
                continue;
            // 全体の連結部分の表面積の最大値を更新
            if(line_sa > maxsa) maxsa = line_sa;
        }
    }
    // loop
    for(i=0; i<n; i++){
        // 連結数が 2 で未チェックの立方体を含むループを抽出する
        if(cube[i].nc == 2 && cube[i].checked == 0){
            loop_sa = proc_loop(i);
            if(loop_sa == -1) continue;       // ループに k 個立方体がない
            // 全体の連結部分の表面積の最大値を更新
            if(loop_sa > maxsa) maxsa = loop_sa;
        }
    }
    if(maxsa == -1)                    // k 個連結した立方体は無い
```

```
            printf("-1¥n");
        else      // k 個の立方体の表面積から連結部分の表面積の最大値を引けば
                  //  多面体の最小表面積になる
            printf("%d¥n", 6*s*s*k - maxsa);
    }
    return 0;
}

#ifdef DEBUG_CONNECTION
void print_cubes(void)
{
    // debug print
    int i;
    for(i=0; i<n; i++){
        printf("%d: ", i);
        printf("x=%d y=%d z=%d checked=%d nc=%d",
                cube[i].x, cube[i].y, cube[i].z, cube[i].checked, cube[i].nc);
        if(cube[i].nc > 0){
            printf(" %d(%d)", cube[i].cc[0], cube[i].sa[0]);
        }
        if(cube[i].nc == 2){
            printf(" %d(%d)", cube[i].cc[1], cube[i].sa[1]);
        }
        printf("¥n");
    }
}
#endif
```