ACM ICPC 2018 国内予選問題 解説

平石 裕実

(京都産業大学 情報理工学部)

プログラムの大きさと必要な知識等一覧

問題	ソース	行数	実行時間(秒)	解法や知識・テクニック等
A. 所得格差	pa.c	26	0.157(0.168)	平均値、ループ、1次元配列
B. 折り紙	pb.c	75	3.306(1.615)	2次元配列、二重ループ、空間的思考能力
C. 超高層ビル「みなとハルカス」	pc.c	24	0.430(0.324)	総和計算(公式)、2次方程式の根
D. 全チームによるプレーオフ	pd.c	74	1.835(1.015)	深さ優先探索、枝刈り、post-order
E. 浮動小数点	pe.c	58	0.161(0.175)	2進数の浮動小数点加算、切捨て、整数の2進表現
F. 正三角形の柵	pf.c	148	20.14(7.944)	幾何、2直線の交点、ソーティング
G. 数式探し	pg.c	267	0.893(0.814)	BNF記法、構文解析、数式処理、尺取虫戦略
H.優秀なプログラマになるには	ph.c	165	70.61(19.98)	両端キュー、深さ優先探索、動的計画法

- 行数は、コメントや空行やデバッグ用コードを除いた行数
- 実行時間(秒): コンパイル、サンプルデータ、ジャッジデータに対する判定時間を含む（括弧内は-O2でコンパイル）
  Core i7-5650U @ 2.20GHz, 8GBメモリ, MacOS 10.13.6, Apple LLVM version 8.0.0 (clang-800.0.38)

問題A 所得格差

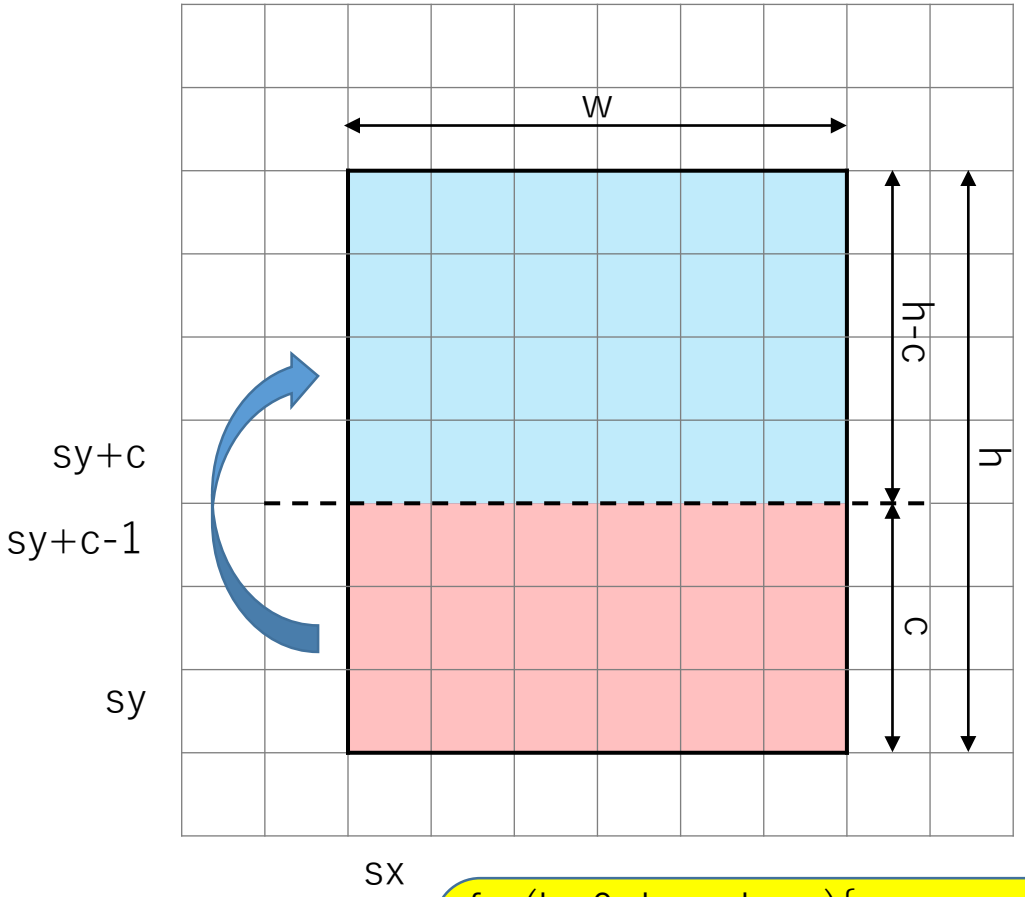
- 最初のループで、各所得a_iを読み込んで配列に格納するとともに、それらの合計sumを求める。
- 合計sumを人数nで割ると平均値が求まる。
- 次のループで、配列に格納された各所得a_iで平均値以下の個数を数えれば良い。
- 人数nは10^4以下で、各所得は10^5以下の整数なので、合計は10^9以下の整数である。したがって、合計を格納する変数sumはint型で良い。
- 各所得a_iは整数であり、平均値以下の所得を数えるので、平均値の計算（合計sum/人数n）は整数型で行えば良い（小数点以下は切り捨て）。
- 時間計算量、領域計算量はともにO(n)。

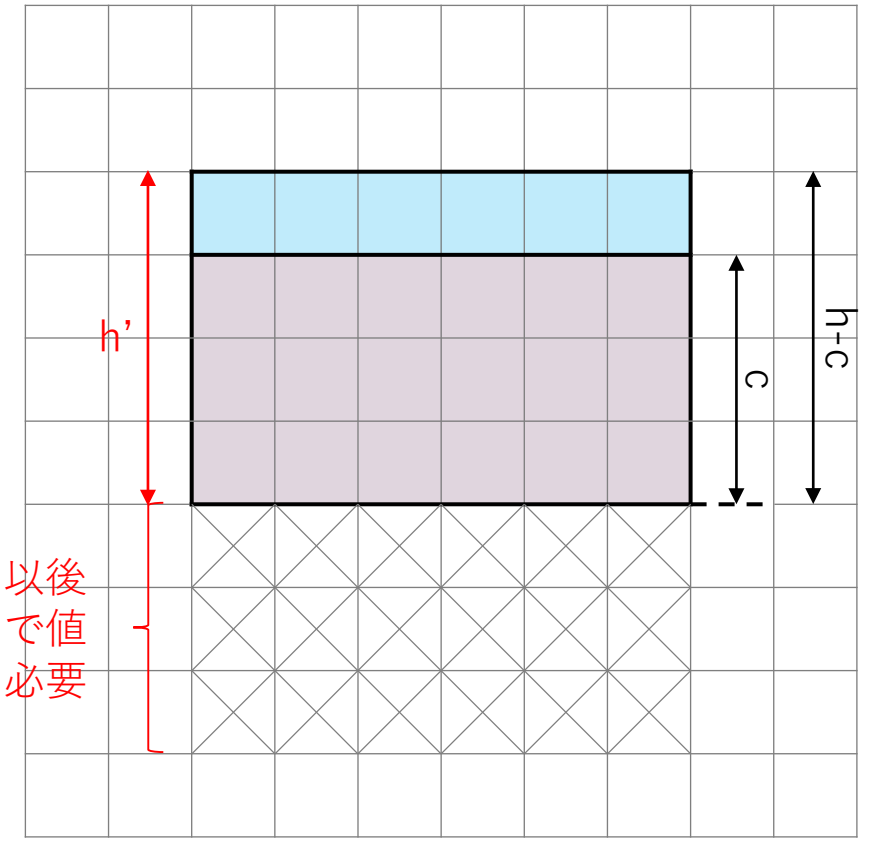
```
int income[10000];
Int n, sum, average, count;
scanf(“%d”, &n);
sum = 0;
for(i=0; i<n; i++){
    scanf(“%d”, &income[i]);
    sum += income[i];
}
average = sum/n;
count = 0;
for(i=0; i<n; i++){
    if(income[i] <= average){
        count++;
    }
}
printf(“%d¥n”, count);
```

問題B 折り紙

- 大きなマットの上に折り紙を置いて折り重ねるものとする。
- 折り重ねることにより折り紙はマット上を右または上に移動していく。
- 長方形の紙の大きさは32x32以下で、折る回数は20回以下なので、マットの一辺の大きさは672(=32+32x20)あれば十分（実際には、折り重ねるたびにその方向の長さは少なくとも１減り、折る回数が20回以下であるので、$32+\sum_{i=11}^{30} i = 442$で良い）
- マットに対応する２次元配列を用いて、各位置における折り紙の重なり枚数を記憶させる。
- 各折り重ね操作をシミュレートして、操作結果の折り紙の位置と大きさ、重なり枚数を求めていく。
- 穴をあける位置（マット上の位置）に対応する配列の要素が求める値である。
  与えられた穴をあける位置が(x,y), 折り紙の左下のマット上の位置が(sx, sy)ならば、マット上で(sx+x, sy+y)の位置の重なり枚数が求める値である。
- 672x672のint型配列に必要なメモリは約1.8M byte、2次元配列の各要素の更新回数は、高々32x32x20 = 20,480 なので、領域、計算時間共に問題ない。

問題B続：上側へ折り重ねる場合





sy'=sy+c

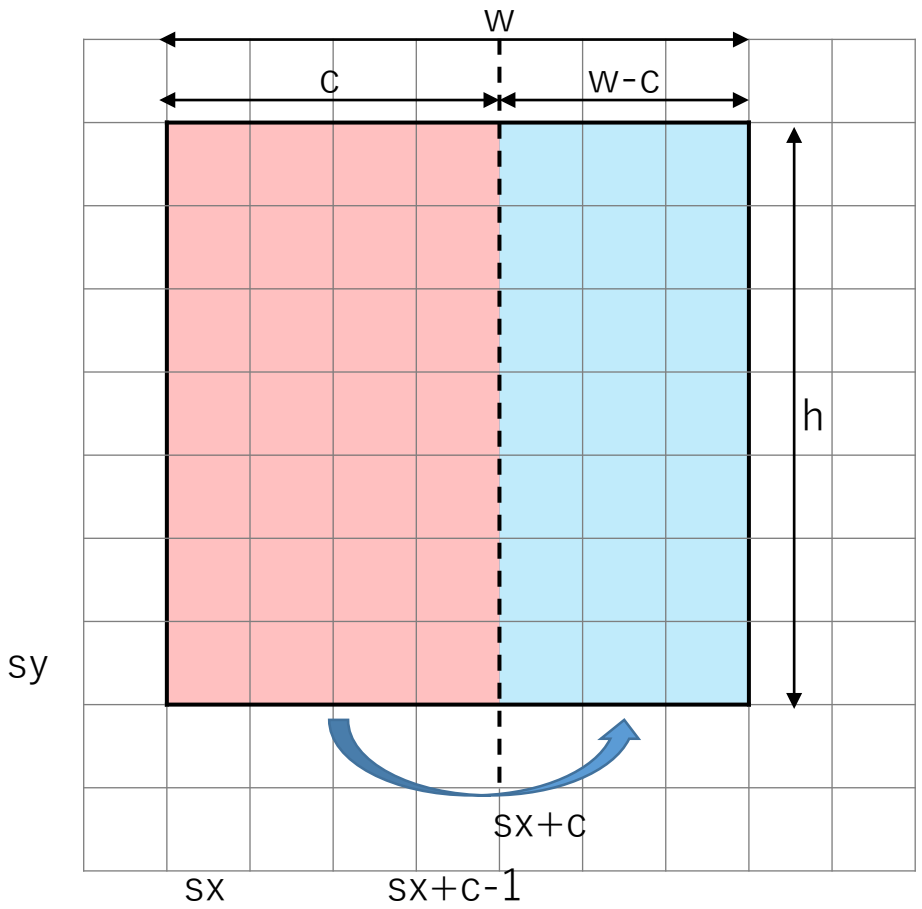
この部分は以後使わないので値を消去する必要は無い

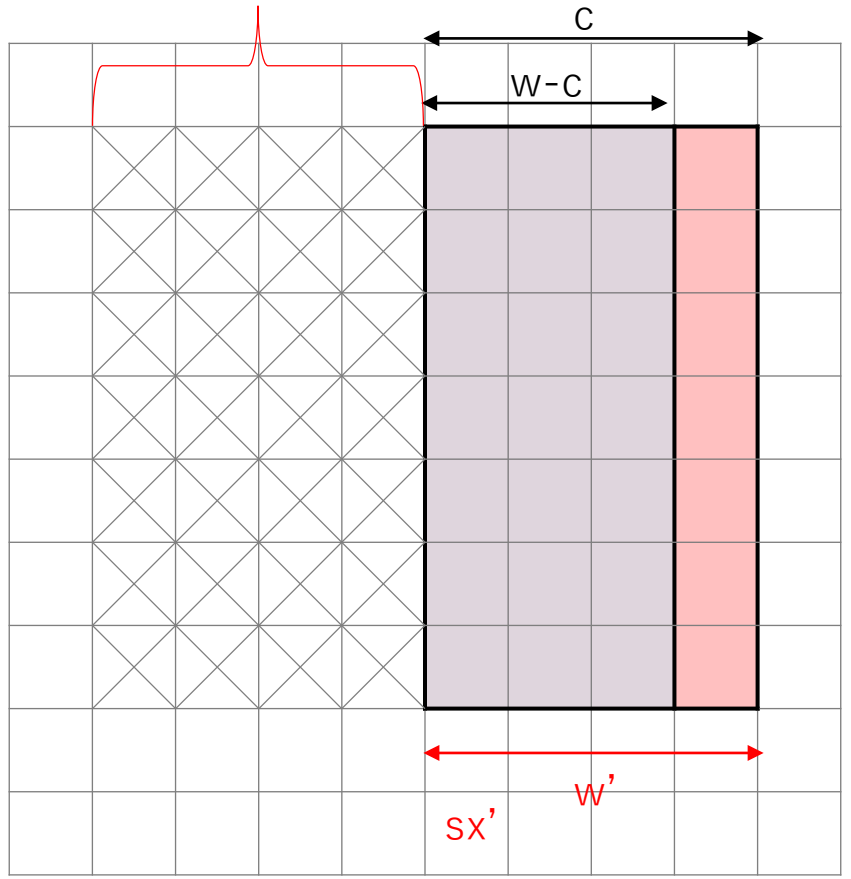
```
for(k=0; k<c; k++){
    for(x=sx; x<sx+w; x++){                    // sy'-1-k行の値をsy'+k行に加算する
        thickness[sy'+k][x] += thickness[sy'-1-k][x];
    }
}
```

h'=MAX(h-c,c)とすればcとh-cの大小関係を考慮する必要は無い

問題B続：右側へ折り重ねる場合

この部分は以後使わないので値を消去する必要は無い





sx'=sx+c

```
for(k=0; k<c; k++){
    for(y=sy; x<sy+h; y++){                    // sx'-1-k列の値をsx'+k行に加算する
        thickness[y][sx'+k] += thickness[y][sx'-1-k];
    }
}
```

w'=MAX(w-c,c)とすればcとw-cの大小関係を考慮する必要は無い

問題C 超高層ビル「みなとハルカス」

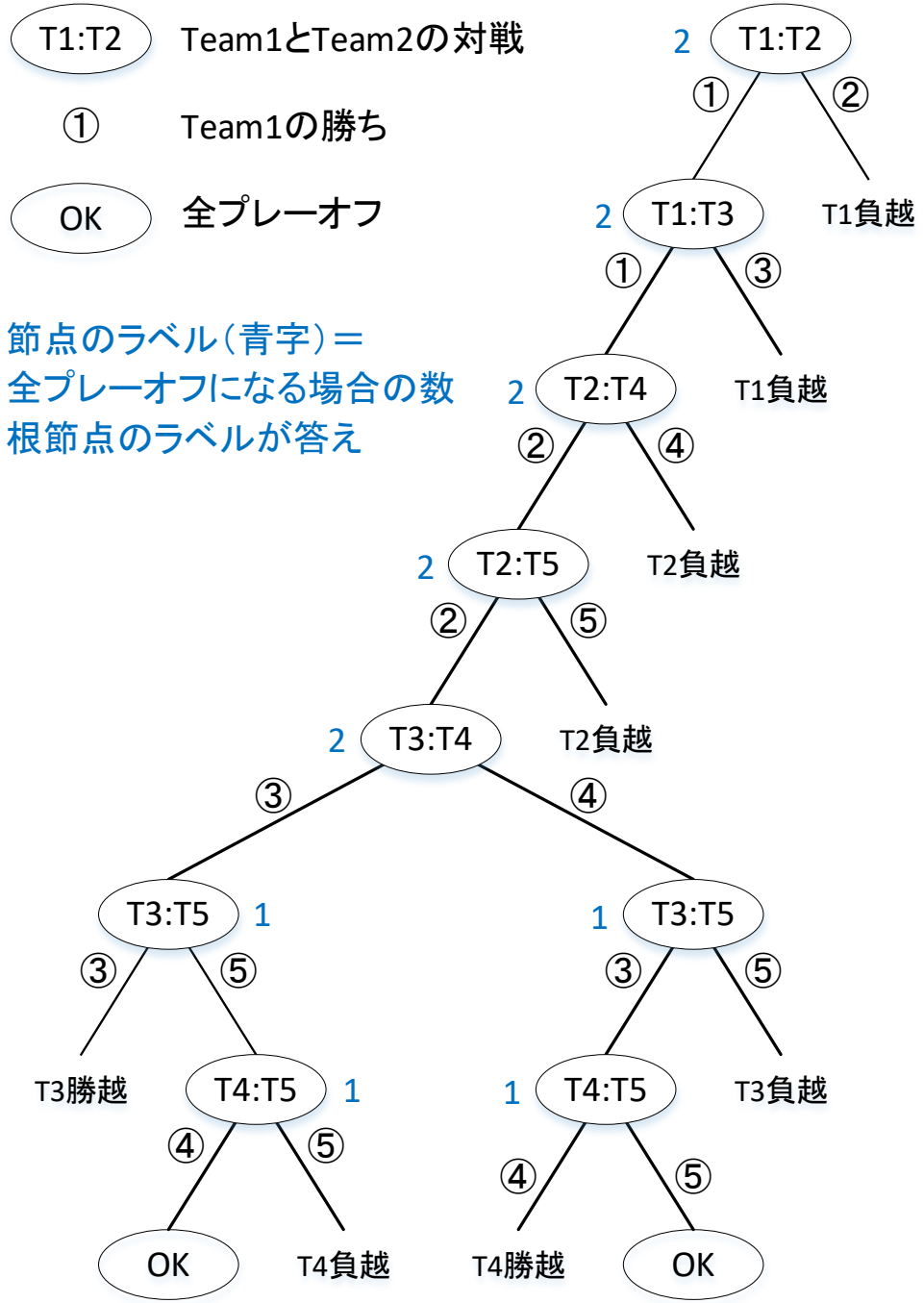
- $floor$階から上に連続したkフロアを借りる場合の賃料は、$\frac{k(k+1)}{2}+(floor-1)k$
- この賃料が予算 bと一致するとすると、$b=\frac{k(k+1)}{2}+(floor-1)k$
  これより、$floor-1=(b-\frac{k(k+1)}{2})/k$となるので、$(b-\frac{k(k+1)}{2})$は$k$の倍数
- 一方、連続したkフロアを借りる場合の賃料の最小値（1階から借りる場合）は $\frac{k(k+1)}{2}$
  なので、これがbより大きい場合は連続したkフロアを借りることはできない
  （上層階から借りる場合は1階から借りる場合より賃料が高い）
- $b=\frac{k(k+1)}{2}$をkについて解くと、kは$\frac{-1+\sqrt{1+8b}}{2}$以下である
- 浮動小数点の計算誤差を考慮して、$\frac{-1+\sqrt{1+8b}}{2}+1$以下の整数$k$で、
  $\frac{k(k+1)}{2}\leq b$かつ$(b-\frac{k(k+1)}{2})$がkの倍数となる最大のkを求めればよい
  この時、$floor=\frac{b-\frac{k(k+1)}{2}}{k}+1$となる
- 時間計算量は$O(\sqrt{b})$

問題D 全チームによるプレーオフ

- 全チームプレーオフの場合、各チームの勝利数と負け数は等しい
- 各未対戦（チームa対チームb）に対して、aが勝つ場合と負ける場合の各々に対して、全チームプレーオフになる場合の数を深さ優先探索で求める
- 探索木の節点は、未対戦ゲーム（チームa対チームb）に対応し、節点の値は全チームプレーオフになる場合の数である（チームaが勝った時に全チームプレーオフになる場合の数＋チームbが勝った時に全チームプレーオフになる場合の数）
- 節点の値の更新は post-order
- 探索中に全チームプレーオフの可能性が無くなったら（あるチームの勝利数または負け数がチーム数の半分より大きい）枝刈りする
- 探索木の根節点が求める値である

問題D続
Sample Input に対する探索木

	Team1	Team2	Team3	Team4	Team5
Team1				lost	lost
Team2			lost		
Team3		won			
Team4	won				
Team5	won				



問題E 浮動小数点

- 2進数の浮動小数点の加算では、指数部を大きい方に合わせてから加算する。この時、指数部が小さい方の数値の仮数部の値は、指数部の差の桁数分だけ小さくしてから加算する。さらに、仮数部を加算した結果が2以上になる場合は、指数部を1増やして仮数部を半分にする。

仮数部の精度が、小数点以下4桁の場合の例（この例では仮数部が表す数値（1以上2未満）で表現）

指数部	仮数部
011	1.0101
010	1.1001

指数部を大きい方にそろえる

指数部	仮数部
011	1.0101
011	0.1100

指数部を1増やしたので
仮数部を半分にする(1bit右シフト)

加算結果　011　10.0001

仮数部が1以上2未満になるように補正

指数部	仮数部
100	1.0000

指数部を1増やし仮数部を
半分にする(1bit右シフト)

- 仮数部の加算は、仮数部が表す数値の小数点以下の桁数(52bits)＋整数部分(2bits)で表現できるので、これを54ビットの符号無し整数(long型で0～2^{54}-1の範囲の整数)とみなして加算すれば良い。
- この加算結果が2^{53}以上なら、仮数部の実際の値が2以上になる。
- 上記の例では小数点以下4桁なので、6ビットの符号無し整数として加算し、32以上となるかどうかで判定できる。

問題E続

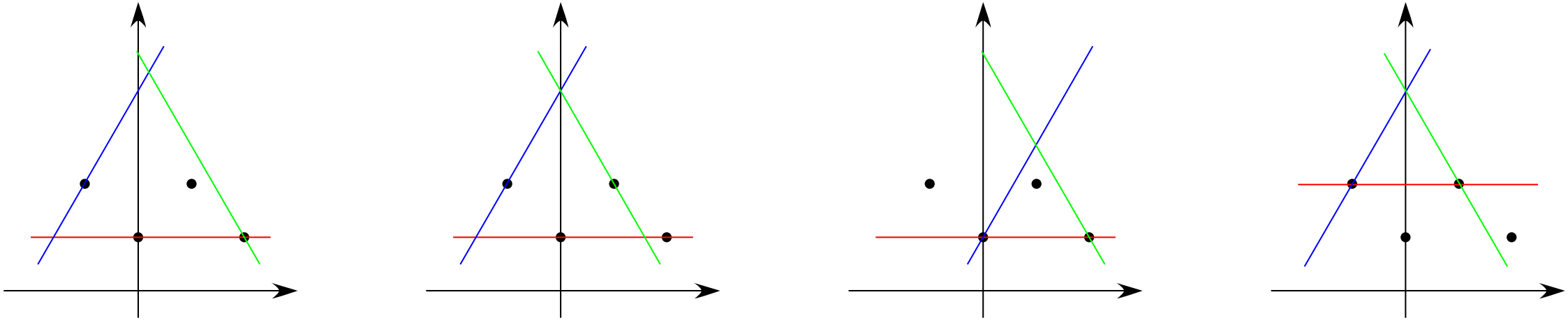
- このような加算をn回繰り返せば良いが、nの最大値は10^{18}なので非現実的。
- s := s + a を毎回行うのではなく、s が2以上になる最小のqを求めて、s := s + q*a を計算する。但し、aの総加算回数がnを超える場合は、総加算回数がnとなるようにqを小さくする。
- long型整数の2^{53}を、54ビットの符号無し整数とみなしたaで割った時の商をq', 余りをrとする。r=0ならq = q' であり、r > 0 なら q = q' + 1 である。
- s が2以上になるたびに補正後のaの値は半分になるので、s := s + q*a を53回繰返すとa=0となりそれ以降計算する必要はない。

問題F 正三角形の柵

- 柵の長さが最短の場合、三角形の底辺の柵、右斜辺の柵、左斜辺の柵には各々少なくとも1本の梨の木が接している状態である。（接していない場合は、接するまで柵を内側に移動させることで柵の長さを短く出来る。但し、接している木は柵内と考える。）
- 梨の木の位置の座標は整数なので、右斜辺と左斜辺に複数本の梨が接することは無い。（柵の形状は底辺がx軸に平行な正三角形なので、左斜辺の傾きは$\sqrt{3}$、右斜辺の傾きは$-\sqrt{3}$で共に無理数なので座標値が整数の梨の木と複数接することは無い。底辺には複数の梨の木が接する可能性がある）
- 柵の長さが最短の場合、柵外の木の本数は、ちょうど k (柵外に残しても良い本数）である（柵外の本数が k 未満の場合、左斜辺や右斜辺を内側に移動することで、柵外の本数を k にすれば柵の長さをより短くできる）
- したがって、柵外の木の本数が k となる柵の中で最短長を求めれば良い。
- アルゴリズム：
  - 底辺を1番下から一つずつ上にあげる（底辺より下にある梨の木の本数が0～kまで）
    底辺を上に上げる回数はO(k)
    - その底辺に対して、左斜辺はもっとも左の斜辺とし（左斜辺より左に梨の木は無い）、右斜辺はもっとも右の位置から柵外の本数が k になるまで左によせる
      左に寄せる回数は O(k)
    - 柵外の本数をkにしたまま左右の斜辺を右に移動しながら柵の長さを求め最小値を更新する
      移動と柵の長さを求める回数はO(k)
  - このアルゴリズムの計算量は $O(k^2)$

問題F続

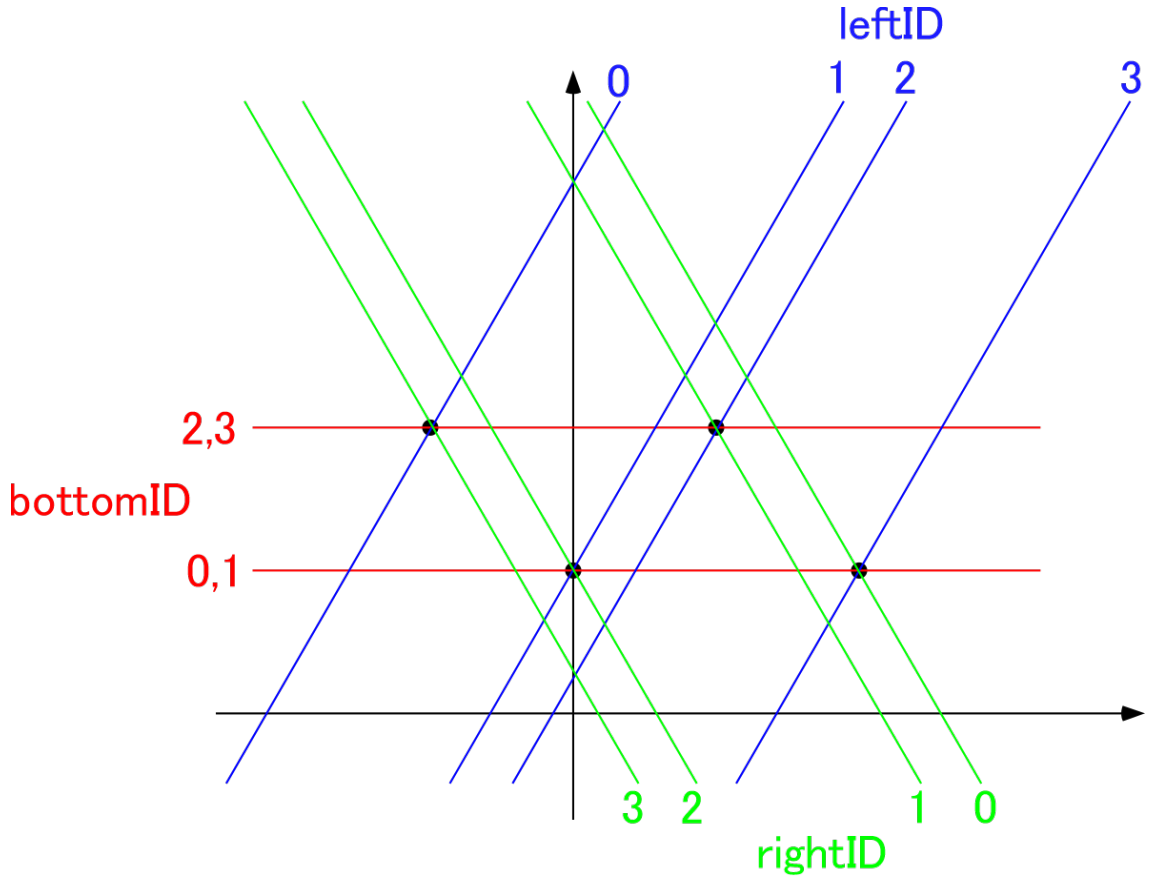
- Sample Input の最初のデータセット(柵外の木の許容本数は1)の例
  1. 最初各辺を最も外側にする(図a)。この時、柵外の木の本数は0。
  2. 柵外の木の本数が許容本数になるまで右斜辺を左に寄せる(図b)。この柵の長さを最小値にセット。
  3. 柵外の木が1本増えるまで左斜辺を右に寄せ、また、柵内の木が１本増えるまで右斜辺を右に寄せる（柵外の木の本数は許容本数のままである）(図c)。
     この柵の長さを求め最小値よりも小さければ最小値を更新。
  4. 左右の斜辺は、柵外の木の本数を保ったままこれ以上右へ移動できないので、底辺を一つ上に上げ左右の斜辺は最も外側とする(図d)。底辺の下の木の本数が許容本数を超えているので終了。



(a)　(b)　(c)　(d)

問題F続

- 木に接する形で柵を内側や外側へ移動するのを容易にするために、各木に対してleftID, rightID, bottomID の 3 つのID (整数)を割り当てる（O(n log n)）
  - leftID:　　その木に接する左斜辺のy切片の値で降順にソートした時の順位 (0～n-1)
  - rightID:　その木に接する右斜辺のy切片の値で降順にソートした時の順位 (0～n-1)
  - bottomID: その木のy座標で昇順にソートした時の順位（y座標が同じ時はx座標の昇順）(0～n-1)

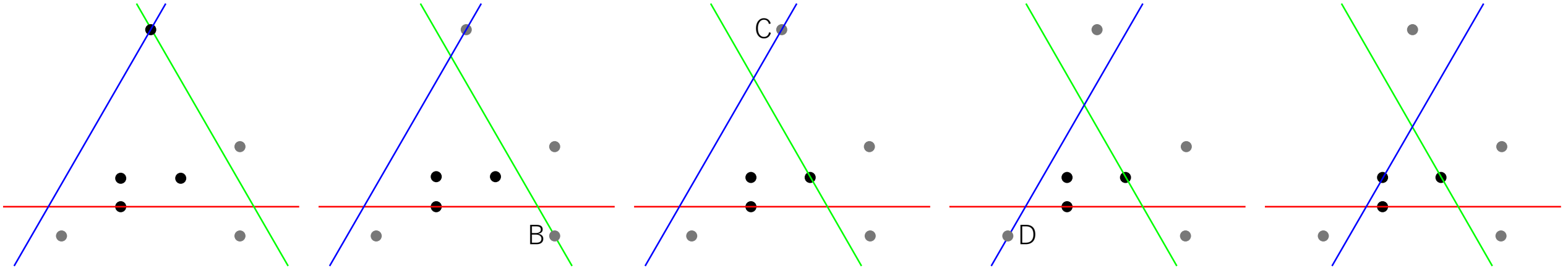


点(x, y)を通る
左斜辺のy切片 $yl = y - \sqrt{3}x$
右斜辺のy切片 $yr = y + \sqrt{3}x$

柵の移動
leftID++　　左斜辺を一つ右へ
rightID++　　右斜辺を一つ左へ
bottomID++　同じ底辺か一つ上の底辺
y座標が変化するまでbotomID++すれば一つ上の底辺となる

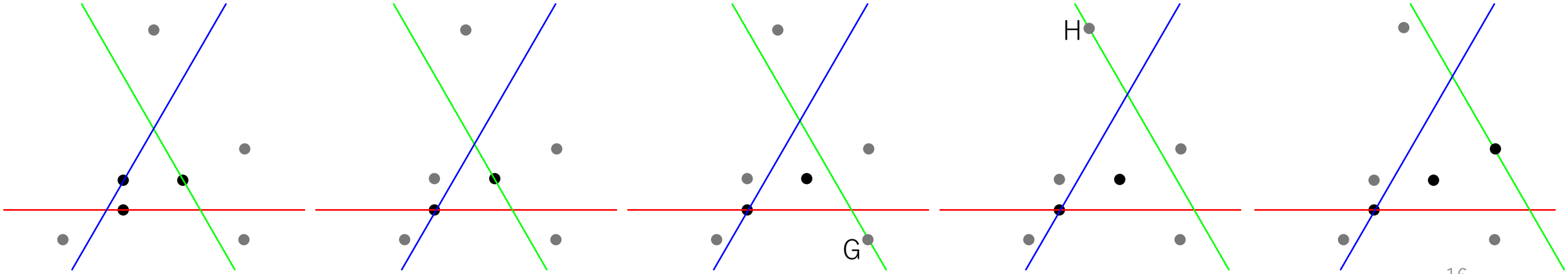
問題F続

- 左右の斜辺を移動する時に必要となる補正
  - 例えば、図(a)において右斜辺を左へ移動して柵外の木の本数を1本増やすことを考える。
  - rightIDを1増やすと右斜辺は別の木に接する位置まで左へ移動し図(b)のようになり、柵外の木の本数が1本増えて4となる。
  - このとき、右斜辺が接している木Bは柵外にあるので、もう一度rightIDを増やすと図(c)のようになる。すなわち、右斜辺を左へ移動して柵外の木を1本増やす際には、右斜辺に接する木が柵内になる位置まで左へ移動することができる。
  - この時点で左斜辺が接していた木Cは柵外になっているので、leftIDを1増やすと左斜辺は別の木に接する位置まで右へ移動し図(d)のようになる。
  - この時点で、左斜辺が接している木Dは柵外なので、もう一度leftIDを1増やすと図(e)のようになる。
  - すなわち、左斜辺も、左斜辺に接する木が柵内になる位置まで右に移動することができる。



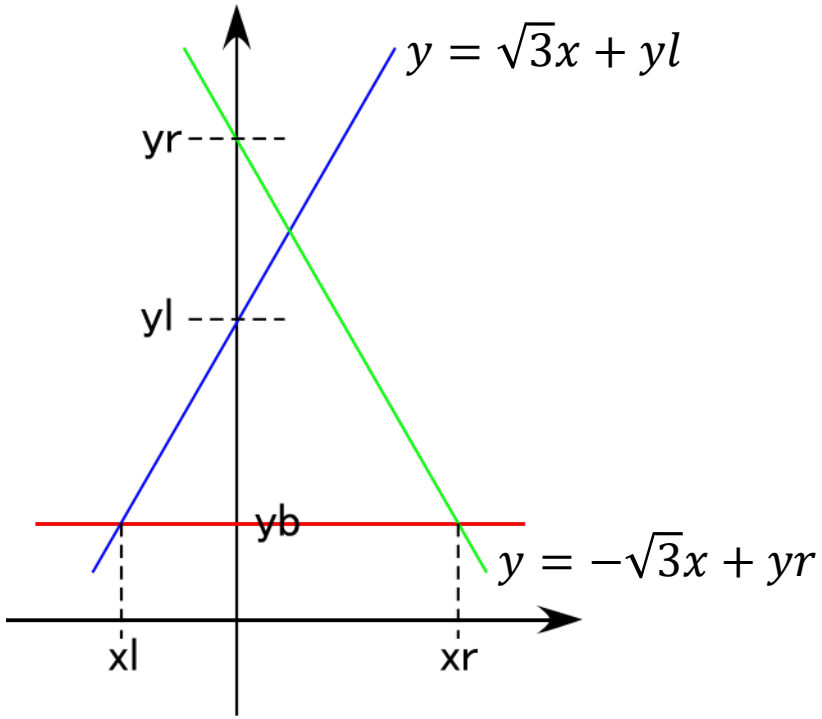
問題F続

- 左右の斜辺を移動する時に必要となる補正(続)
  - 次に、柵外の木の本数を保ったまま柵を右へ移動させることを考える。
  - まず、左斜辺を柵内の木に接するまで右へ移動させる。(図(f))
  - この時点で柵外の本数が1本増えて5本となっているので、右斜辺を右へ移動させて柵外の1本を柵内に取り込む。すなわち
    - rightIDを1減らして右斜辺を右へ移動すると図(g)のようになる。
    - この時右斜辺が接している木Gは柵外なので、もう一度rightIDを1減らして右斜辺を右へ移動すると図(h)となる。
    - この時右斜辺が接している木Hは柵外なので、さらにrightIDを1減らして右斜辺を右へ移動すると図(i)となり、柵外の木の本数が1本減って元の4本となる。
    - つまり、右斜辺はそれに接する木が枠内に入るまで右へ移動させれば良い。
- 柵外の木の本数を保ったまま底辺を上へ移動できる場合もあるが、底辺については下から順に処理していくので補正の必要は無い（上の底辺についても後でチェックされる）。



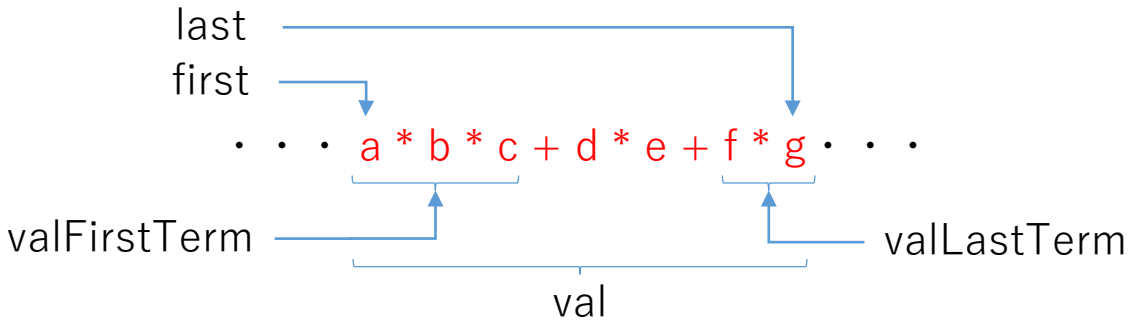
問題F続

- $y = \sqrt{3}x + yl$ と $y = yb$ との交点のx座標 $xl = \frac{yb-yl}{\sqrt{3}}$
- $y = -\sqrt{3}x + yr$ と$y = yb$ との交点のx座標 $xr = \frac{yr-yb}{\sqrt{3}}$
- 正三角形の周囲の長さは $3(xr - xl) = \sqrt{3}(yr + yl - 2yb)$



問題G 数式探し

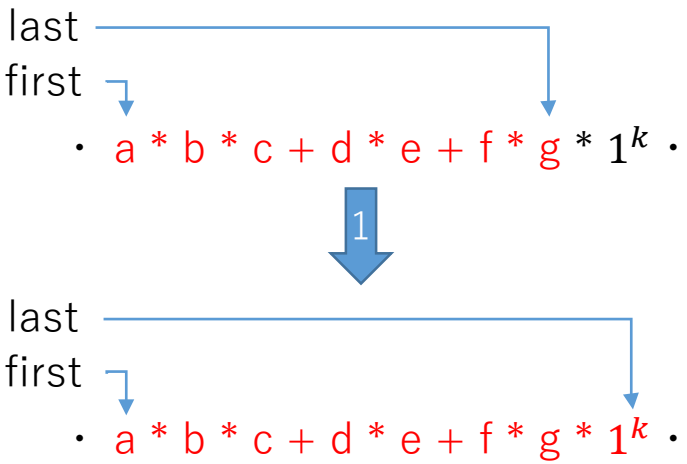
- 括弧で括られた部分式の値は事前に求めておく。
- 1は掛けても式の値が変化しないため特別扱いが必要である。このため、1がk個の積項は1^kと表現しひとまとめに扱う（$1^i * 1^j$は1^{i+j}とする）。
- これにより、2以上の整数や1^kに対する加算と乗算の式となる。
- 与えられた式全体と括弧で括られた各部分式に対して、各々上記のような表現を用いてその部分式で値がnとなるものの個数の総和を求めれば良い。
- 基本的には尺取虫法で求める（注目している部分式の値がnより小さければ右へ伸ばし、nより大きければ左を縮め、nと等しい時にカウントする）が、「$* 1^k$」や「$1^k *$」では式の値が変化しないことを考慮する必要がある。
- 処理対象の部分式の先頭位置をfirst, 最後の位置をlast, 部分式の値をval, 部分式の先頭の積項の値をvalFirstTerm, 最後の積項の値をvalLastTerm, 処理対象の次の位置をnext とする（部分式が一つの積項の場合、valFirstTerm = valLastTerm である）。

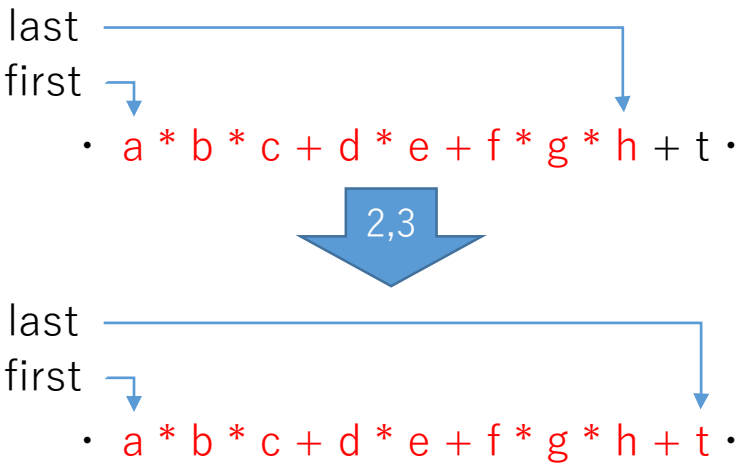


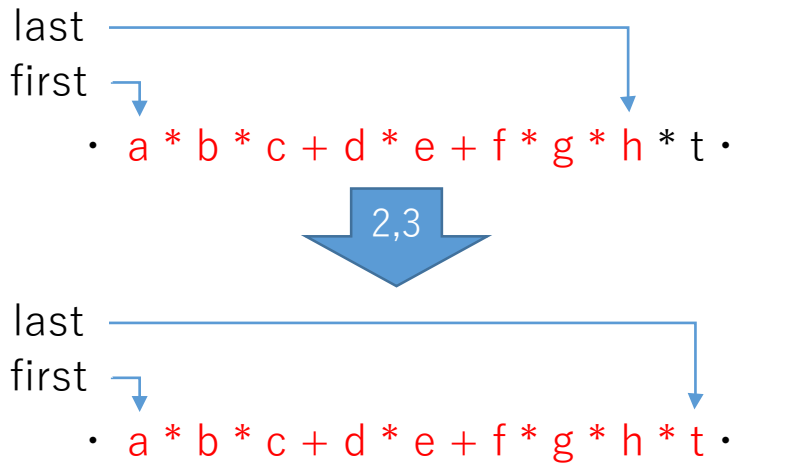
※ 小文字の英字は因子(数値)や1^kを表す

問題G続： val < n の場合　⇒　lastを右にずらしてvalを大きくする

1. lastの次が「$*1^k$」ならlastを「1^k」の位置とする　// 1^k をかけても val は変化しない
2. lastの次が
   - 「$+t$」なら、$\mathrm{val} \mathrel{+}= t; \mathrm{valLastTerm} = t;$
   - 「$*t$」なら、$\mathrm{val} \mathrel{+}= \mathrm{valLastTerm} * (t-1); \mathrm{valLastTerm} \mathrel{*}= t;$
     部分式が一つの積項なら $\mathrm{valFirstTerm} = \mathrm{valLastTerm};$
   - 無ければ終了
3. lastを「t」の位置とする



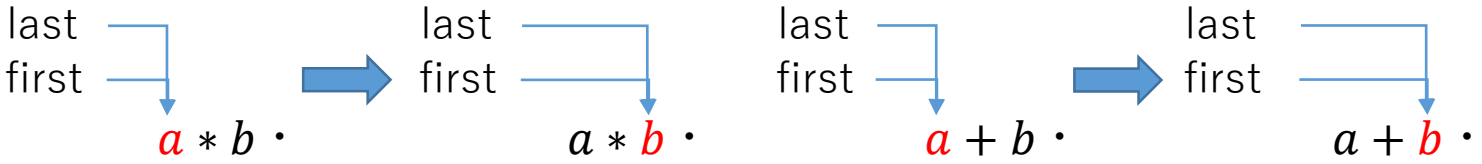




問題G続： val > n の場合　⇒　firstを右にずらしてvalを小さくする

1. firstからの部分式が「$1^k * a$」ならば、「a」の位置をfirstとする　　// valは変化しない

2. first=last ならfirstとlastを次の因子の位置とし、val = valFirstTerm = valLastTerm = 因子の値;



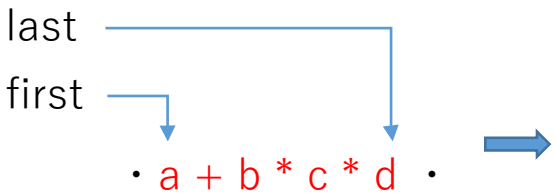
val=valFirstTerm=valLastTerm=b;

first < lastでfirstの次の演算子が
- +なら val −= valFirstTerm;
  - + 以降が一つの項なら valFirstTerm = valLastTerm;
  - + 以降が複数の項なら valFirstTerm = 先頭の項の値;
  - firstを + の右隣りにする
- * なら $\text{val} -= \text{valFirstTerm} - \frac{\text{valFirstTerm}}{\text{first位置の因子の値}}$;
  - $\text{valFirstTerm} = \frac{\text{valFirstTerm}}{\text{first位置の因子の値}}$;
  - * 以降が一つの項なら valLastTerm = valFirstTerm
  - firstを * の右隣にする

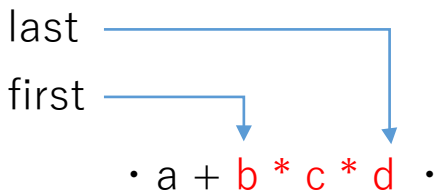
問題G続： val > n で first < last の場合

firstの次が + なら val −= valFirstTerm;
+ 以降が一つの項なら valFirstTerm = valLastTerm;
+ 以降が複数の項なら valFirstTerm = 先頭の項の値;
firstを + の右隣りにする

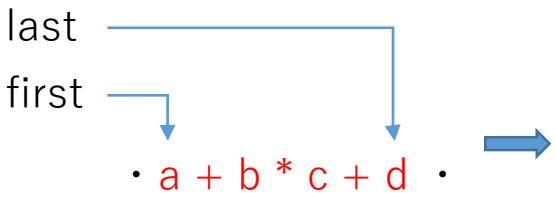
firstの次が * なら $\mathrm{val} -= \mathrm{valFirstTerm} - \frac{\mathrm{valFirstTerm}}{\text{first位置の因子の値}}$;
$\mathrm{valFirstTerm} = \frac{\mathrm{valFirstTerm}}{\text{first位置の因子の値}}$;
* 以降が一つの項なら valLastTerm = valFirstTerm
firstを * の右隣にする



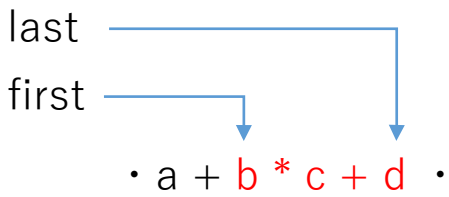
val = a+b*c*d
valFirstTerm = a
valLastTerm = b*c*d



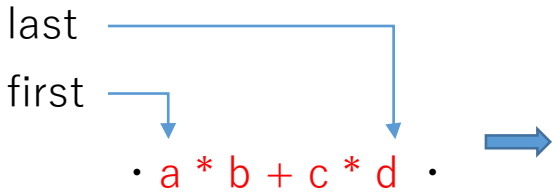
val = b*c*d (val -= a;)
valFirstTerm = b*c*d
vlaLastTerm = b*c*d



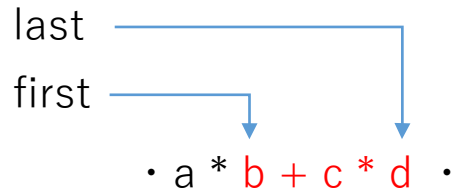
val = a+b*c+d
valFirstTerm = a
valLastTerm = d



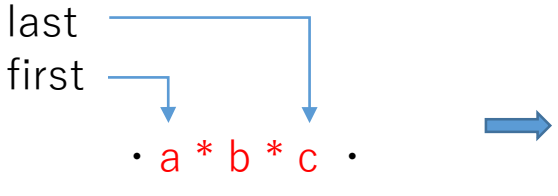
val = b*c+d (val -= a;)
valFirstTerm = b*c (先頭の項)
valLastTerm = d



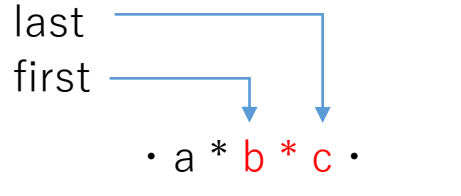
val = a*b+c*d
valFirstTerm = a*b
valLastTerm = c*d



val = b+c*d
(= a*b+c*d − a*b + a*b/a)
valFirstTerm = b (= a*b/a)
valLastTerm = c*d



val = a*b*c
valFirstTerm = a*b*c
valLastTerm = a*b*c



val = b*c
(= a*b*c − a*b*c + a*b*c/a)
valFirstTerm = b*c (= a*b*c/a)
valLastTerm = b*c

問題G続： val = n の場合　⇒　値がnである部分式を数える

1. first = last, val = n = 1 の場合（1^kの場合）
   - 1^k（1がk個の積項）の部分式の個数（値は全て1）$= \frac{k\cdot(k+1)}{2}$
2. first ～ last の式の先頭部分、最後部分の自由度を各々c1, c2 （初期値は1）とする
3. 先頭部分が
$$\begin{cases} 1^k + \cdots \text{ の場合：} c1 = k & // \quad 1\text{の個数が}1\sim k\text{の積項} \\ 1^k * \cdots \text{ の場合：} c1 = k+1 & // \quad 1\text{の個数が}0\sim k\text{の積項} \end{cases}$$
4. 最後部分が
$$\begin{cases} \cdots + 1^k \text{ の場合：} c2 = k & // 1\text{の個数が}1\sim k\text{の積項} \\ \cdots * 1^k \text{ の場合：} c2 = k+1 & // 1\text{の個数が}0\sim k\text{の積項} \\ \text{それ以外で、last の次が } * 1^k \text{ の場合：} c2 = k+1 & //1\text{の個数が}0\sim k\text{の積項} \end{cases}$$
5. first～lastの式の部分式で値がnとなる部分式の個数は $c1 * c2$
6. lastを右へ一つずらす（val < n の時の処理と同じ）

問題G（続）

- 各因子(Factor)に対して、次の構造体で情報を管理する

```
typedef struct {
    long valF;          // factor(因子)の値(括弧つきの場合は左括弧の位置に格納)
    long valT;          // term(積項)の値（積項の開始位置に格納）
    long oneCount;      // 1のみの積項に含まれる1の個数（1のみの積項の開始位置に格納）
    int nextOneST;      // 1のみからなる積項の次の位置（1のみの積項の開始位置に格納）
    int nextRB;         // 左括弧の場合、対応する右括弧（閉じ括弧）の次の位置
    int termID;         // 積項番号
} result_t;
```

- 構文解析の過程で
  - 1を含む積項については、その開始位置、1の個数、終了位置などを事前に求め、それらを開始位置に保存。
  - 積項の開始位置には、積項の値を保存し、積項を構成する各因子に対し、同じ項番号を付与する。
  - 括弧で括られた部分式の値は、対応する右括弧位置の情報と共に、左括弧の位置に保存する。
- 目標値nの最大値は10^9であるが、積項や括弧でくくられた式の値はint型ではオーバーフローするためlong型で計算する。さらに、長い積項の値などはlong型でもオーバーフローする可能性があるので、INFを2×10^9としてnより大きい値は全てINFとして計算する。

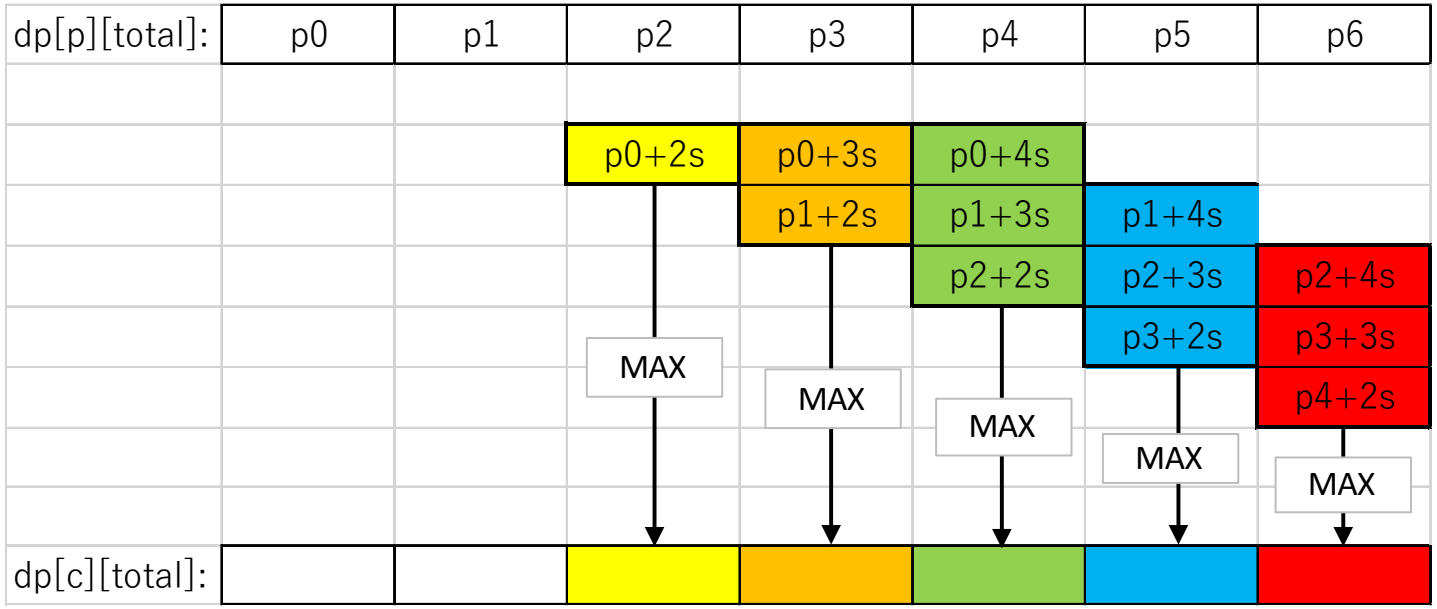
問題H 優秀なプログラマになるには

- 動的計画法の表： dp[スキルp][総授業料total] = 総合力評価値の最大値
  - これまでに支払った授業料の総額がtotalで、スキルpが初めて受講可能になった状態での総合力評価値の最大値を格納
  - スキルpのレベルの上限をupper, スキルpの重要度をs, スキルpを前提とするスキルをcとし、スキルcの習得に最低限必要な前提スキルpのレベルをmin, スキルpの授業料をpayとすると

    $0 \leq total \leq k$ (総予算) に対し、

    $$dp[c][total] = \mathrm{MAX}\left\{dp[c][total],\ \max_{min \leq pay \leq upper} dp[p][total - pay] + s \times pay\right\}$$

    但し、$x < 0$ や $y < 0$ の時は $dp[x][y] = -\infty$とする。
  - min=2, upper=4, k=6 の例



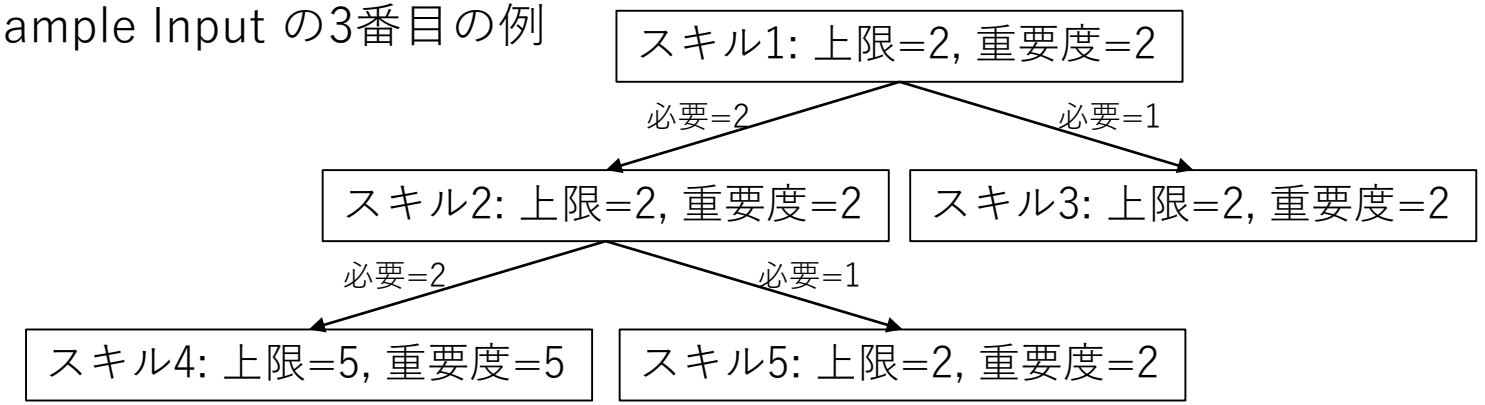
問題H（続）

- この例を見ると、{p0+4s, p1+3s, p2+2s}, {p1+4s, p2+3s, p3+2s}, {p2+4s, p3+3s, p4+2s} などの最大値を順に求めることになる。
- このためには、両端キューを用い、
  - ◆新たなデータ(p2+2s, p3+2s, p4+2s, …)はキューの末尾に追加するものとし
  - ◆その際、キューの末尾のデータが追加するデータ以下の間、キューの末尾のデータを順に削除する。
  - ◆これにより、両端キューの先頭が最大値となる。
  - ◆また、最大値が求まるたびに、最大値を求める範囲がひとつ後ろへずれるので、キューの先頭の要素が範囲外になれば、それを削除する。
  - ◆なお、キュー内のデータは、最大値を求める範囲が後ろへずれるたびに s 増加し、また、キューの先頭からは範囲外になった要素を削除する必要があるため、キューには実際のデータではなく、インデックス(p0, p1, p2, p3, p4 に対し、1, 2, 3, 4)を格納する。

問題H（続）

- 前提条件の関係を枝とする木構造を作成し、preorderで前述の動的計画法の表の値を求める。
  - 各スキルに対して、その前提スキルを親とすると、基本スキル（前提条件のないスキル１）を根とする木構造が得られる。
  - 各スキルに対して、その子からなる後続スキルリストを作成する。
  - 後続スキルリストでは、前提スキルの必要レベルの降順にソートしておく。
  - 後続スキルリストの最後に、深さ優先探索で次に訪問するスキルを追加する。（次に訪問するスキルが無い場合は、ダミースキルを追加する。）これにより、自分より先に訪問したスキルに対する授業料に基づいて、その時点での総合力評価の最大値が求まる。
  - この木構造を深さ優先探索しながら、preorder で後続スキルリストに含まれるスキルに対応する動的計画法の表の部分を更新する。
  - 動的計画法の表で、訪問したダミースキルで総授業料が最大の所の値が、総合力評価の最大値である。

Sample Input の3番目の例



後続スキルリスト

1: 2 3 6
2: 4 5 3
3: 6
4: 5
5: 3

※ 青字は次に訪問するスキル

問題H（続）

Sample Input の3番目の例

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]											
dp[3][ ]											
dp[4][ ]											
dp[5][ ]											
dp[6][ ]											

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]			4								
dp[3][ ]		2	4								
dp[4][ ]											
dp[5][ ]											
dp[6][ ]	0	2	4								

p=1, c=2, min=2, upper=2, s=2

p=1, c=3, min=1, upper=2, s=2

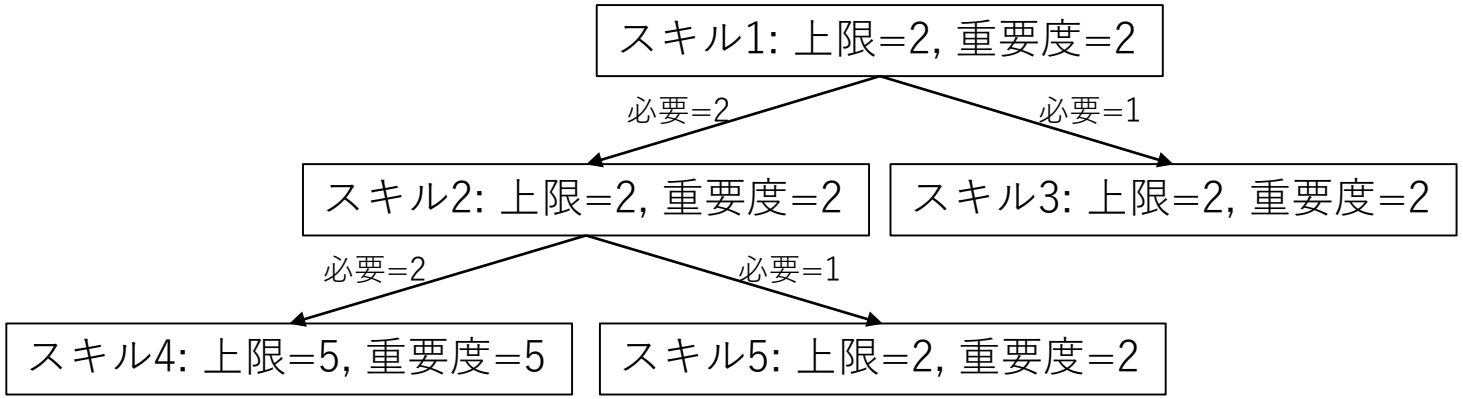
p=1, c=6, min=0, upper=2, s=2

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]			4								
dp[3][ ]		2	4	6	8						
dp[4][ ]					8						
dp[5][ ]				6	8						
dp[6][ ]	0	2	4								

p=2, c=3, min=0, upper=2, s=2

p=2, c=4, min=2, upper=2. s=2

p=2, c=5, min=1, upper=2, s=2



後続スキルリスト

1: 2 3 6
2: 4 5 3
3: 6
4: 5
5: 3

問題H（続）

Sample Input の3番目の例

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]			4								
dp[3][ ]		2	4	6	8						
dp[4][ ]					8						
dp[5][ ]				6	8	13	18	23	28	33	
dp[6][ ]	0	2	4								

p=4. c=5, min=0, upper=5, w=5

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]			4								
dp[3][ ]		2	4	6	8	13	18	23	28	33	35
dp[4][ ]					8						
dp[5][ ]				6	8	13	18	23	28	33	
dp[6][ ]	0	2	4								

p=5, c=3, min=0, upper=2, s=2

	0	1	2	3	4	5	6	7	8	9	10
dp[1][ ]	0										
dp[2][ ]			4								
dp[3][ ]		2	4	6	8	13	18	23	28	33	35
dp[4][ ]					8						
dp[5][ ]				6	8	13	18	23	28	33	
dp[6][ ]	0	2	4	6	8	13	18	23	28	33	35

p=3, c=6, min=0, upper=2, s=2

後続スキルリスト

1: 2 3 6
2: 4 5 3
3: 6
4: 5
5: 3