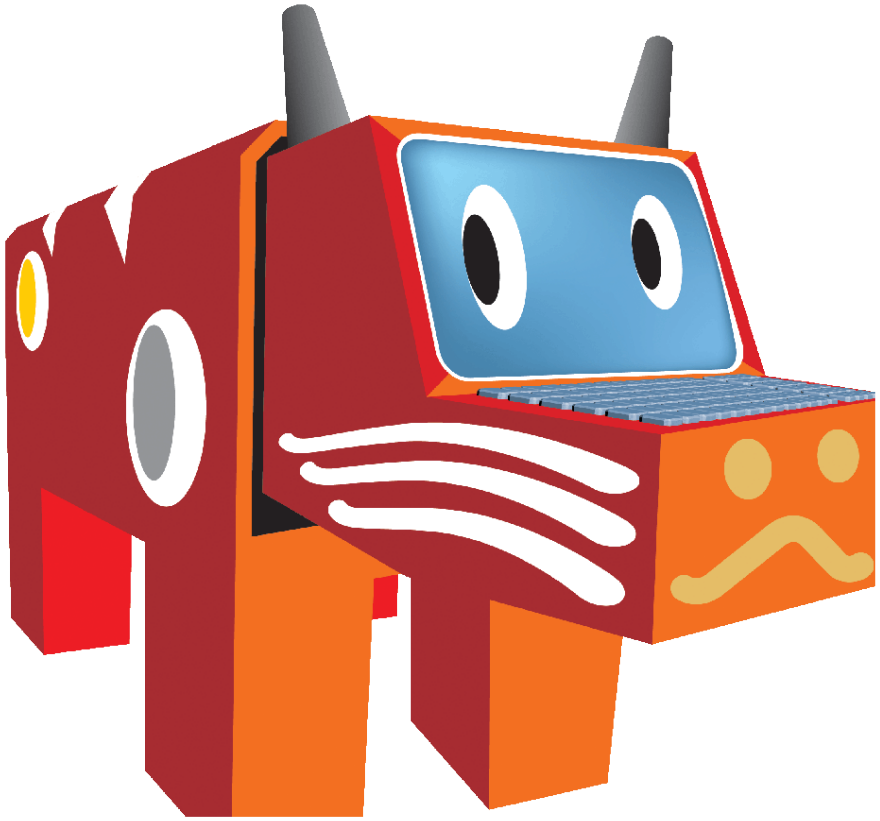
パソコン甲子園2014本選

プログラミング部門　解説

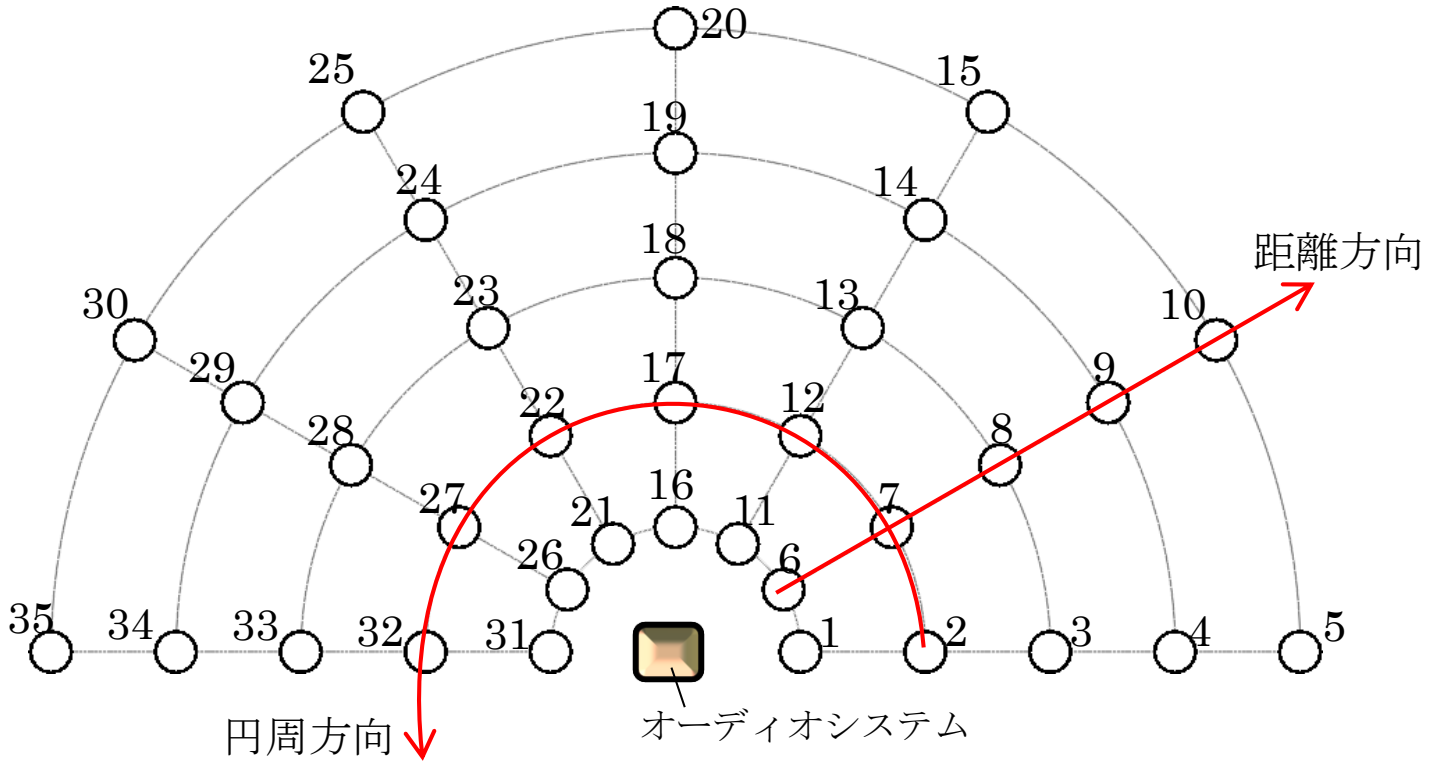


会津大学

問1　有益氏のオーディオルーム

問題概要

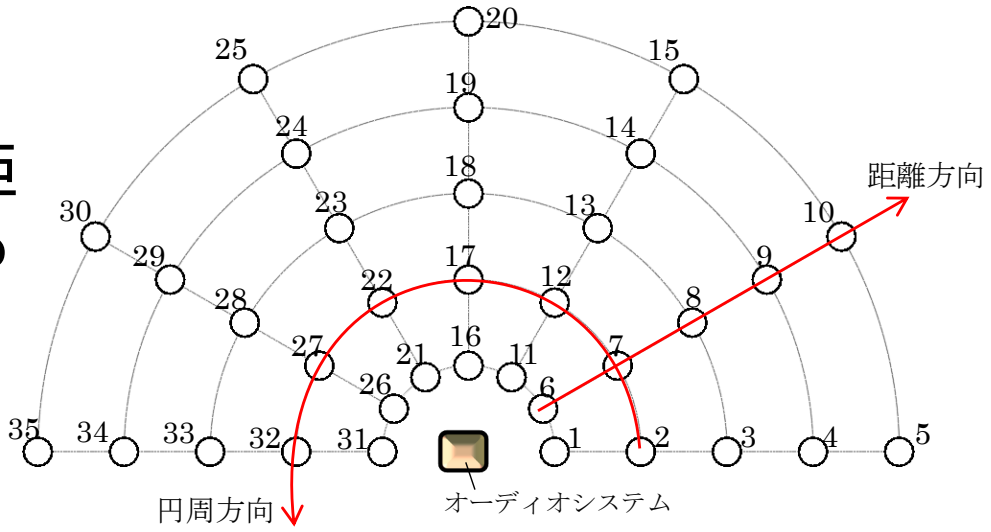
- オーディオシステムを中心に、計測点が半円状にある
- 計測点は番号が固定
  - 円周方向に0°から180°まで30度刻み
  - 距離方向に100cmから500cmまで100cm刻み



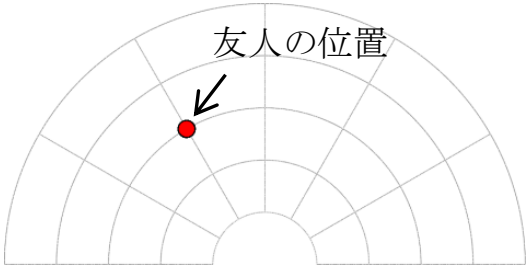
問1　有益氏のオーディオルーム

問題概要

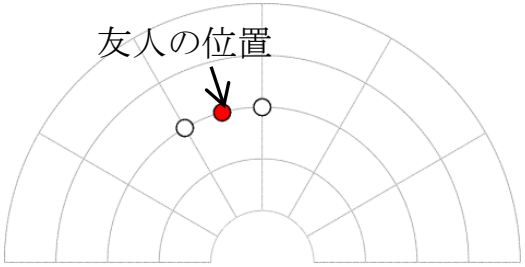
- 入力で、円周方向の角度と距離(友人の位置)が与えられるので、<u>最寄の点</u>を選択する.



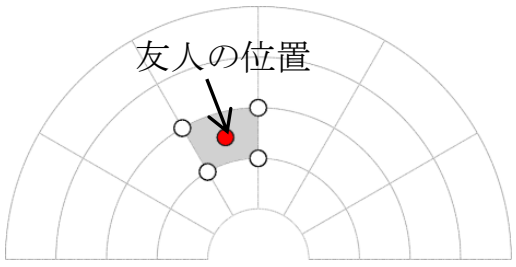
<u>最寄の点</u>の取り方:



友人がちょうど計測点の位置にいれば、その 1 点を選びます。この図の例の場合、選ばれるのは 23 番の点です。



友人が円弧（または線分）の上にちょうど乗っていれば、その円弧状（または線分）上にある、友人に最も近い 2 点を選びます。この図の例の場合、選ばれるのは 18 番と 23 番の点です。



友人が円弧と線分によって囲まれた区画の内側（円弧や線分の上ではない）にいれば、区画を作っている 4 点を選びます。この図の例の場合、選ばれるのは 17、18、22、23 番の点です。

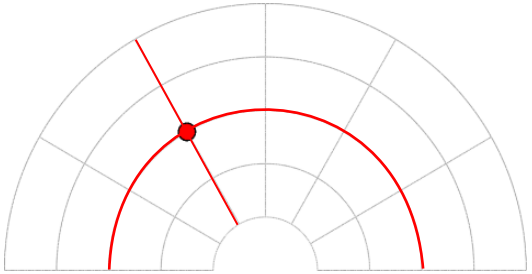
問1　有益氏のオーディオルーム

講評

- 提出数35、正答数24.
- 与えられた位置が、線の上に乗っているか判定する.
- ただし、友人の位置は角度と距離で与えられるので、幾何的に判定する必要はない！

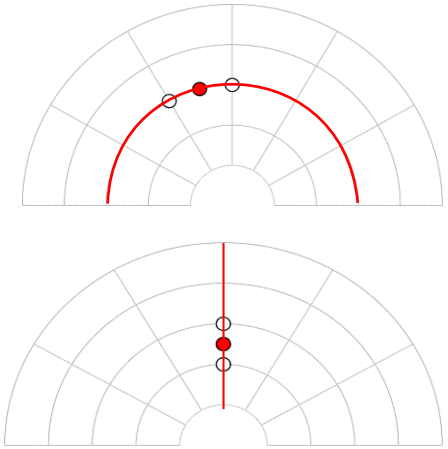
１点選ぶ場合：

円周方向と距離方向、
両方の線上



２点選ぶ場合：

円周方向だけ、または
距離方向だけの線上



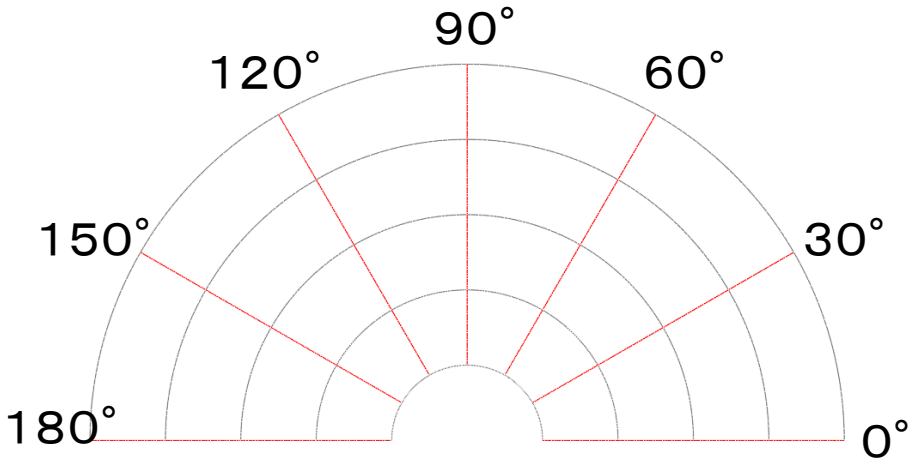
４点選ぶ場合：

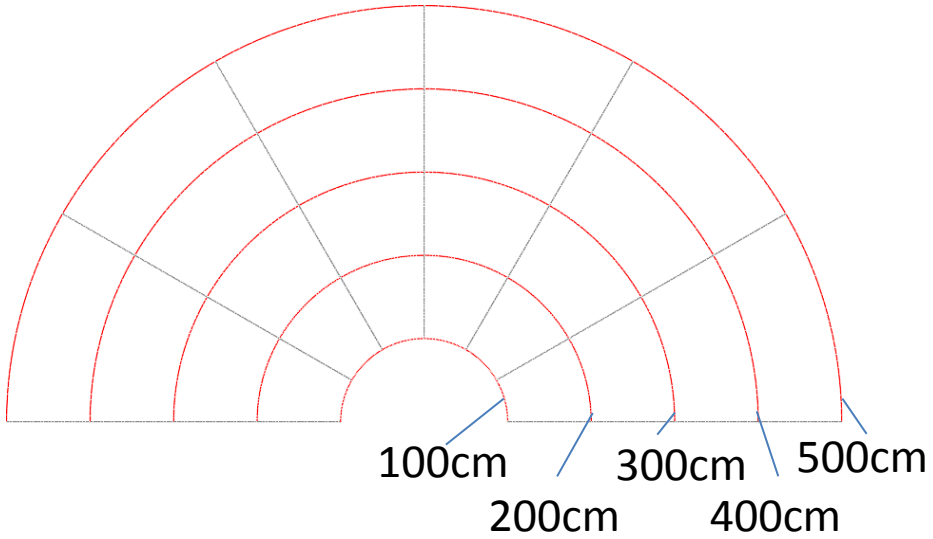
それら以外

問1　有益氏のオーディオルーム

解法

- 円周方向と距離方向について、線の上にのっているか判定
- のっているなら、何本目か





円周方向の線にのっているか？

$t\%30 = 0$か判定

円周方向に何本目か？

$m = t/30$

距離方向の線にのっているか？

$r\%100 = 0$か判定

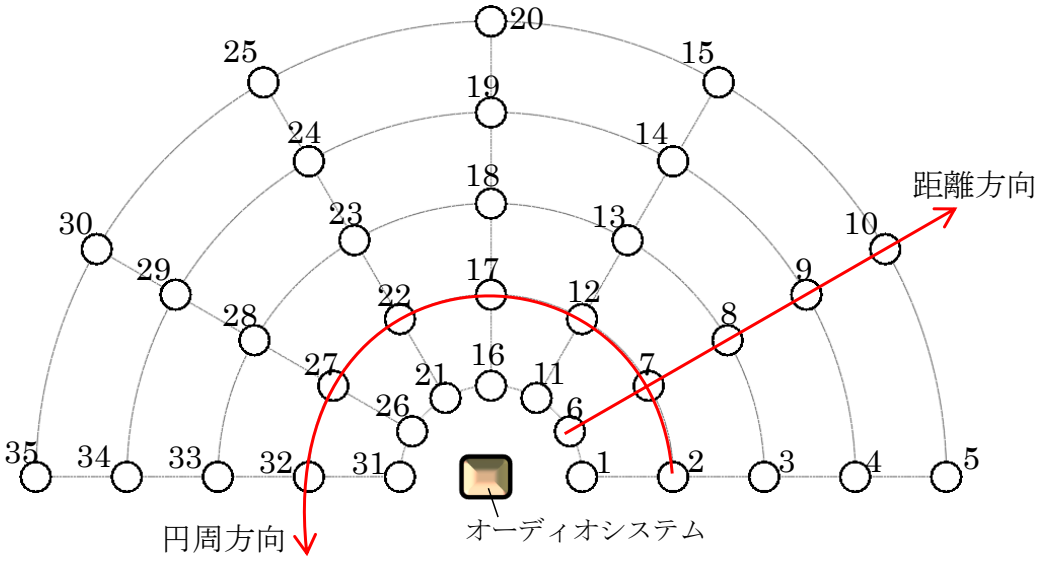
距離方向に何本目か？

$n = r/100$

問1　有益氏のオーディオルーム

解法

- 何本目か分かれば、点番号は簡単な式でわかる



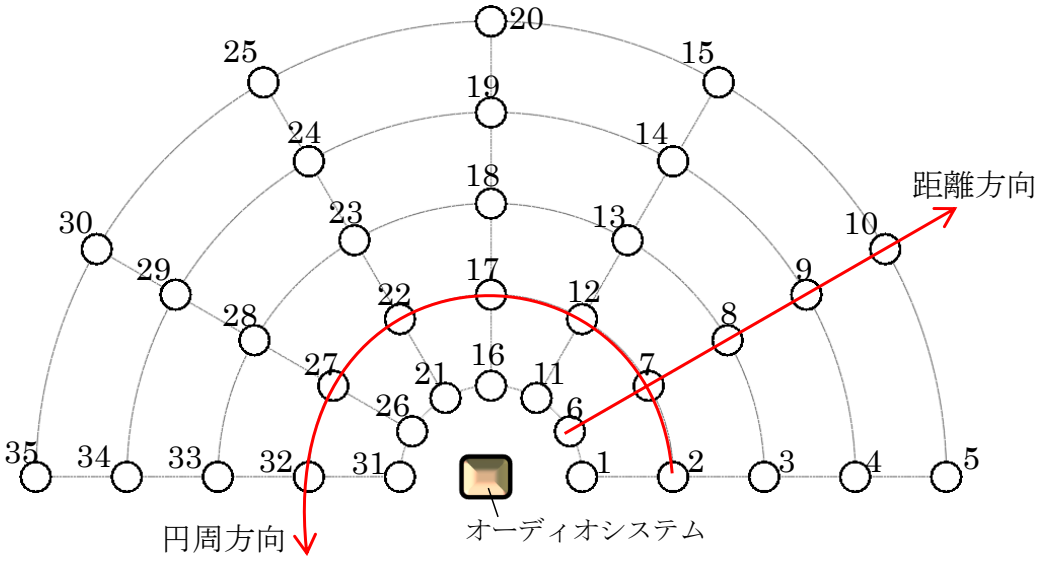
円周方向に何本目か？
$m = t/30$

距離方向に何本目か？
$n = r/100$

円周方向に一つ進むと点番号は5増える
距離方向に一つ進むと点番号は1増える

関数 F(m, n) = 5m + n　を用意

問1　有益氏のオーディオルーム



円周方向に何本目か？
$m = t/30$

距離方向に何本目か？
$n = r/100$

関数 F(m, n) = 5m + n

出力の仕方

- １点の場合：F(m, n)だけ
- ２点で円周方向の線にのっている場合：F(m, n) と F(m+1, n)
- ２点で距離方向の線にのっている場合：F(m, n) と F(m, n+1)
- その他、つまり４点：F(m, n) とF(m, n+1)とF(m+1, n) とF(m+1, n+1)

問1　有益氏のオーディオルーム

```
int CalcID(int r, int t) { return r + 5*t; }

main() {

  int n, r, t;
  scanf("%d", &n);
  for ( int i=0; i<n; ++i ) {

    scanf("%d %d", &r, &t);

    bool bOnR = r%100 == 0 ? true : false;
    bool bOnT = t%30 == 0 ? true : false;
    r = r/100;
    t = t/30;

    if ( bOnR && bOnT ) printf("%d¥n", CalcID(r, t));
    else if ( bOnR ) printf("%d %d¥n", CalcID(r, t), CalcID(r, t+1));
    else if ( bOnT ) printf("%d %d¥n", CalcID(r, t), CalcID(r+1, t));
    else {
      printf("%d %d %d %d¥n", CalcID(r, t),
             CalcID(r+1, t), CalcID(r, t+1), CalcID(r+1, t+1));
    }
  }
}
```

問2　対称3進数

問題概要

- $1 \leq w \leq 100{,}000$が表現できる3^nの重りがある
- 天秤で、wと釣り合う重りの置き方を求める
  - n番目の重りを　使わないならn桁目は「0」
    重りの側に置くならn桁目は「+」
    wの側に置くならn桁目は「-」

講評

- 提出数29、正答数17
- wを対称3進数に変換する問題と考えられる
- 与えられたwに対して、1から試していっても間に合う

問2　対称3進数

解法1

- $w \leq 100{,}000$ なので、000…00, 000…01, 000…1-, 000…10と、総当たりでwと等しくなるまで試す.
  - 3^{11}=177,147なので、100,000は3進数で12ケタ以内.
  - 各桁を配列で持っておいて、1ステップで1を足す
  - 繰り上げを計算

計算量はO(w × 桁数)

問2　対称3進数

解法2

*w*を3で割った商と余りを使って、下位の桁から求めていく。ただし、余りが2のときだけ商に1を加える.

1. 空の列digitsを用意する。
2. d=w%3として、dの値で場合分け
   - d=0のとき、「O」をdigitsの先頭に追加し、w=w/3とする。
   - d=1のとき、「＋」をdigitsの先頭に追加し、w=w/3とする。
   - d=2のとき、「ー」をdigitsの先頭に追加し、w=1+w/3とする。
3. digitsを出力する。

計算量はO(桁数)

問2　対称3進数

解法2

例: w=132.

1. d=0 → 0, w=44
2. d=2 → -, w=14+1=15
3. d=0 → 0, w=5
4, d=2 → -, w=1+1=2
5, d=2 → -, w=0+1=1
6, d=1 → +

「＋－－0－0」

問2　対称3進数(解法1)

```
#define DIGIT 12

int TernaryValue( int* expr ) {
  int val = 0;
  int m = 1;
  for ( int i=0; i<DIGIT; ++i ) {
    val += expr[i]*m;
    m *= 3;
  }
  return val;
}

void PrintTernaryExpr( int* expr ) {
  int j = DIGIT - 1;
  while ( expr[j] == 0 ) --j;

  for ( int i=j; i>=0; --i ) {
    if      ( expr[i] == 1 ) printf("+");
    else if ( expr[i] == 0 ) printf("0");
    else                     printf("-");
  }
  printf("¥n");
}
```

```
void TernaryExpr( int w ) {

  int expr[DIGIT];
  std::fill( expr, expr+DIGIT, 0 );

  int val = 0;
  while ( w != val ) {

    ++expr[0];
    for ( int i=0; i<DIGIT-1; ++i ) {
      if ( expr[i] <= 1 ) break;
      ++expr[i+1];
      expr[i] = -1;
    }
    val = TernaryValue( expr );
  }

  PrintTernaryExpr( expr );
}

main() {
  int w;
  while ( scanf("%d", &w) == 1 ) {
    if ( w == 0 ) break;
    TernaryExpr( w );
  }
}
```

問3　ニッシン館マラソン部

問題概要

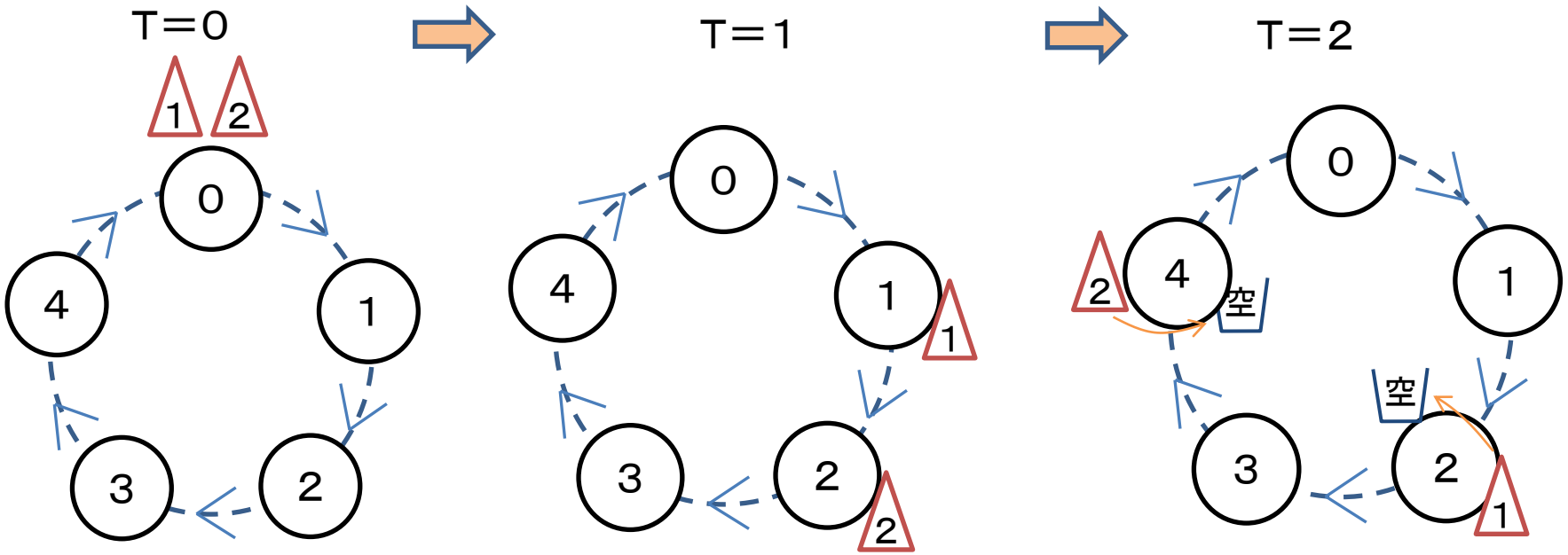
- 周回コースをN人の部員が、決まった時間まで走る.
- 各部員は1単位時間に走る距離が決まっている.
- 全員が1単位時間ごとに必ず給水.
- その後、1単位時間で着いた給水所で空の容器を置き、さらに中身の入った容器をとることを繰り返す.
- 空の容器は届いてから1単位時間の時点で飲料が補充され再利用できる
- 容器はいくつ必要になるか？

講評

- 提出数32、正答数14.
- 設定がやや複雑な問題.
- さらに、それをプログラムで表現できるか.

問3　ニッシン館マラソン部

例：N=2 (部員数), R=5 (周回コース長さ), T=4 (単位時間数)
p1 = 1 (部員1のペース), p2 = 2 (部員2のペース)



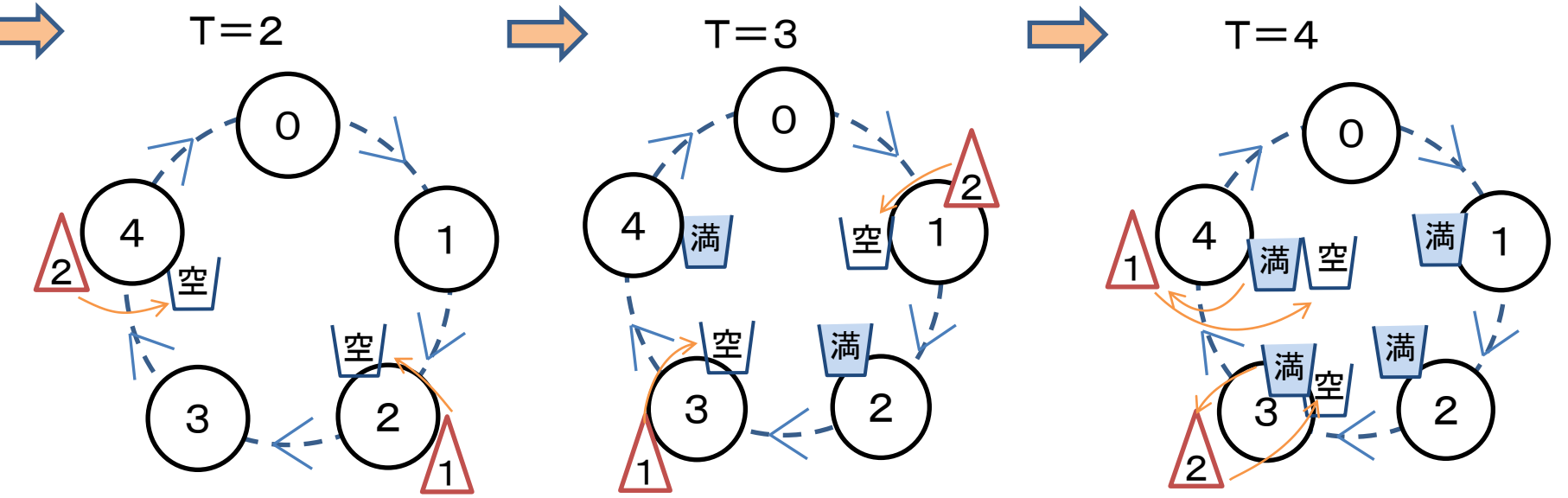
部員1と部員2がスタート地点に居る

部員1と部員2は新しい容器が必要
(必要容器数=2)

部員1と部員2は新しい容器が必要
(必要容器数=4)
さらに空容器を置く

問3　ニッシン館マラソン部

例：N=2 (部員数), R=5 (周回コース長さ), T=4 (単位時間数)
p1 = 1 (部員1のペース), p2 = 2 (部員2のペース)



部員1と部員2は新しい容器が必要
(必要容器数=4)
さらに空容器を置く

部員1と部員2は新しい容器が必要
(必要容器数=6)
さらに空容器を置く

部員1と部員2は補充された容器を再利用できる
(必要容器数=6)
これで終わり. 答えは6

問3　ニッシン館マラソン部

解法

- 時間を1ステップ増やしては、全員走らせるシミュレーション. N=100, T=1000までなので十分間に合う.
- 周回コースの長さ分の配列を2つ用意する.
  - 配列の1つは、各地点に空容器が幾つあるか.
  - 配列のもう1つは、各地点に補充済み容器が幾つあるか.
- 部員が各地点に付いたら、その地点の空容器数に+1．ただし、初回のみ空容器を置かない！
- その地点に補充済み容器があれば、その地点の補充済み容器数を-1、なければ必要な容器数(答え)に+1.
- １単位時間分のシミュレーションを終えたら、全地点の補充済み容器数に+空容器数. さらに空容器数=0

問3　ニッシン館マラソン部

```
class Person {
public:
  int m_nCurr;
  int m_nPace;
public:
  Person() {
    m_nCurr = 0;
    m_nPace = 0;
  }
};

int nPerson, nRound, nTime;
Person person[MAX_N];

int cup = 0;//必要な容器カウント
int empty[MAX_R];//空容器カウント
int filled[MAX_R];//補充された容器カウント

void Input( ) {

  scanf("%d %d %d", &nPerson, &nRound, &nTime);
  for ( int i=0; i<nPerson; ++i ) scanf("%d", &person[i].m_nPace);

  std::fill(empty, empty+nRound, 0);
  std::fill(filled, filled+nRound, 0);
}
```

問3　ニッシン館マラソン部

```
main() {

  Input( );

  for ( int t=1; t<=nTime; ++t ) {

    for ( int i=0; i<nPerson; ++i ) {

      Person &p = person[i];
      p.m_nCurr = (p.m_nCurr + p.m_nPace)%nRound;

      if ( t > 1 ) ++empty[p.m_nCurr];
      if ( filled[p.m_nCurr] > 0 ) --filled[p.m_nCurr];
      else ++cup;
    }

    //1単位時間終了 空容器に補充
    for ( int i=0; i<nRound; ++i ) {
      if ( empty[i] > 0 ) {
         filled[i] += empty[i];
         empty[i] = 0;
      }
    }
  }
  printf("%d¥n", cup);
}
```

問4　デッドロックを検出せよ

問題概要

- 各ユーザが各データをlockまたはwaitしているという状況が与えられる.
- それらの依存関係で作られたグラフに、循環があればデッドロックに陥っている.
- デッドロックを検出せよ.

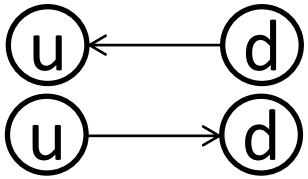
講評

- 提出数29、正答数12
- 循環を検出する探索問題.
- 少々注意する点アリ.

問4　デッドロックを検出せよ

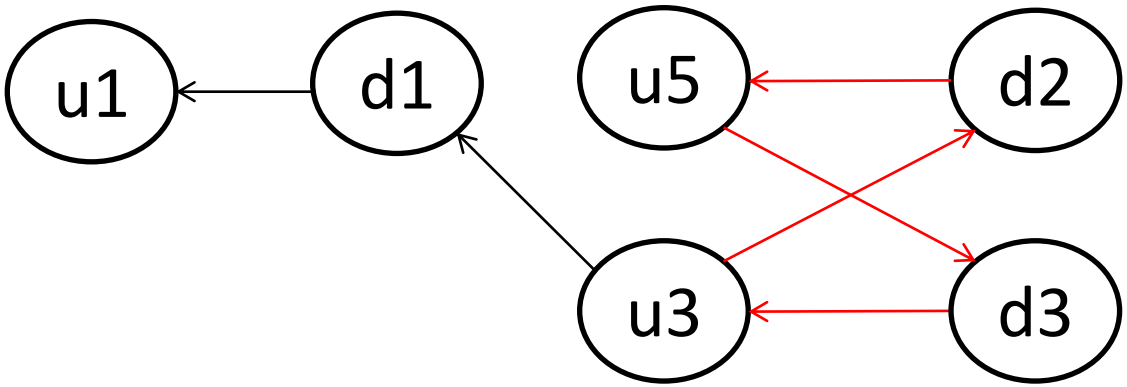
解法

- 依存グラフを作る.
  - u lock dなら
  - u wait dなら



- DFSなどで、始点へ戻るか調べる.
- ただし、入力に現れるすべてのノードを始点にする.
- このとき、一度訪ねたノードを再び訪ねることが無いように（無限ループ！）.

ユーザ1やデータ1を開始点にして探索しただけではダメな例：



1 lock 1
3 lock 3
5 lock 2
3 wait 1
5 wait 3
3 wait 2

問4　デッドロックを検出せよ

```
typedef std::vector<int>    vInt;
#define ITER(i,c)    for(__typeof((c).begin()) i=(c).begin();i!=(c).end();++i)
#define MAX_U 100
#define MAX_D 100
#define MAX_N MAX_U + MAX_D

class Node {
public:
  int m_id;
  vInt m_child;
public:
  Node() { m_id = -1; }
};
Node aNode[MAX_N];
std::bitset<MAX_N> visit;

bool IsCycle( int root, const Node& node ) {

  if ( node.m_id < 0 || visit.test( node.m_id ) ) return false;

  visit.set( node.m_id, true );
  if ( node.m_id == root ) return true;

  ITER ( itN, node.m_child ) {
    if ( IsCycle( root, aNode[*itN] ) ) return true;
  }
  return false;
}
```

問4　デッドロックを検出せよ

```
void Solve() {
  for ( int i=0; i<MAX_N; ++i ) {
    visit.reset( );
    ITER ( itN, aNode[i].m_child ) {
      if ( IsCycle( aNode[i].m_id, aNode[*itN] ) ) {
        printf("1¥n");
        return;
      }
    }
  }
  printf("0¥n");
}

main() {
  int nLine;
  int u, d;
  char str[8];

  scanf("%d", &nLine);
  for ( int i=0; i<nLine; ++i ) {
      scanf("%d %s %d", &u, str, &d);
      --u; --d; d += MAX_U;
      aNode[u].m_id = u;
      aNode[d].m_id = d;
      if ( strncmp( str, "lock", 4 ) == 0 ) aNode[d].m_child.push_back( u );
      else if ( strncmp( str, "wait", 4 ) == 0 ) aNode[u].m_child.push_back( d );
  }
  Solve();
}
```

問５　新薬開発

問題概要

- 素材(素材ノード)と選択方法(選択ノード)から成る樹形図が与えられる.
- ノードにはオプション属性(選択してもしなくても良いノード)が付く場合がある.
- １番目のノードから樹形図を下って行って、作成可能な薬の数を数え上げる.

講評

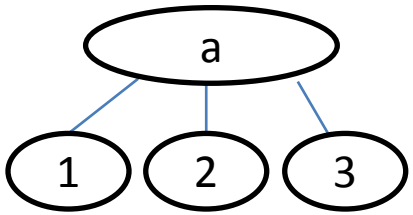
- 提出数１２、正答数８.
- 選択方法やオプション属性など、どのように数え上げればよいか考えるのが難しい.
- 再帰で書くのがやりやすい.

問5　新薬開発

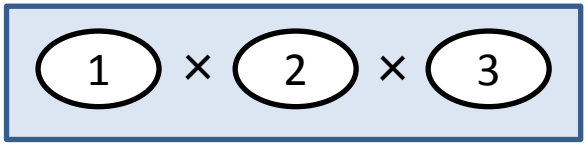
解法

- DFSによって、子ノードで作成可能なパターン数を求めていく.

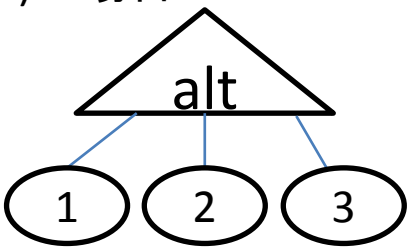
E(Elem)の場合



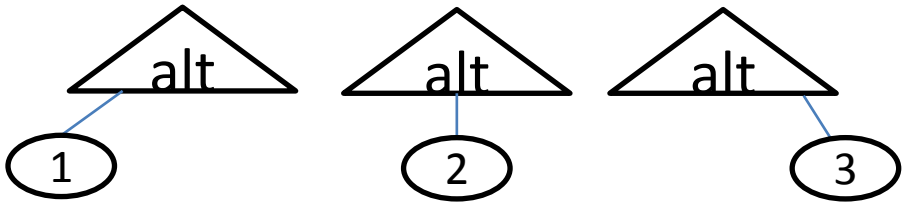
子のどれも使うので、子のパターン数の積

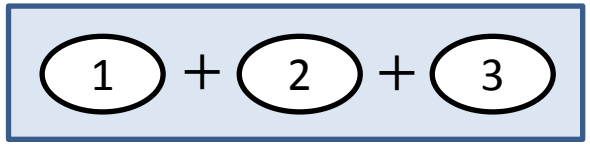


A(Alt)の場合



子をどれか一つ選ぶので、子のパターン数の和



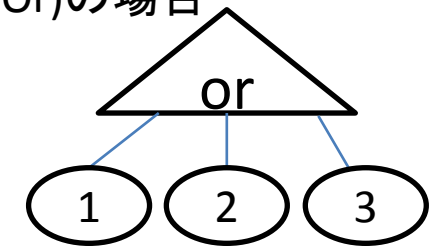


問5　新薬開発

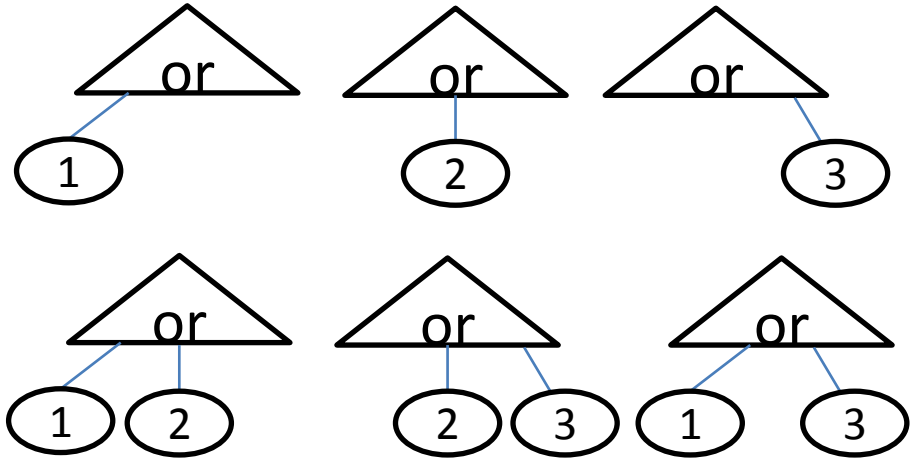
解法

- DFSによって子ノードで作成可能なパターン数を求めていく

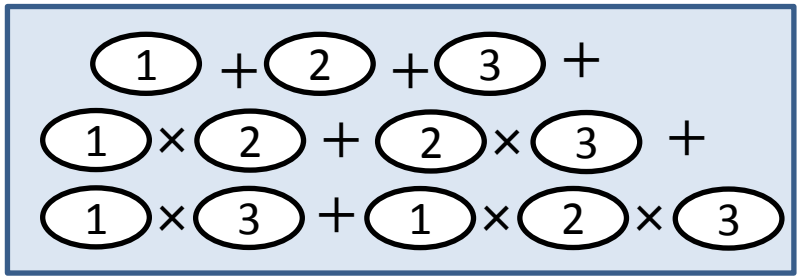
R (Or)の場合



一つ以上選ぶ組み合わせの総和



各々の組み合わせ中では子が持つパターン数の積を計算



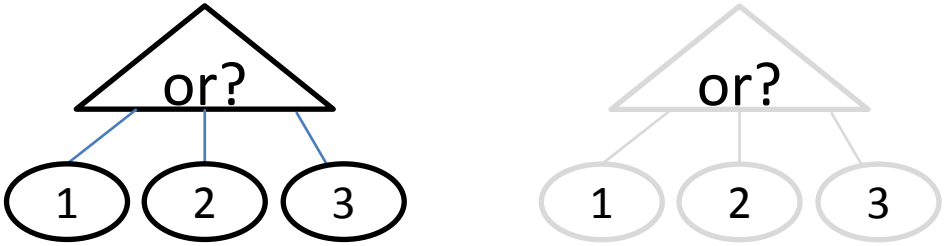
大きな数になるので、1,000,000,007で割った余りを毎回求める

問5　新薬開発

解法

- DFSによって子ノードで作成可能なパターン数を求めていく

Optionであれば、自身を選ばないという選択肢が一つ増えるだけ



自身のパターン数に+1する
この場合orの結果に+1

問5　新薬開発

```
typedef std::vector<int>    vInt;
typedef unsigned long long ullong;

#define ITER(i,c)   for(__typeof((c).begin()) i=(c).begin();i!=(c).end();++i)
#define FOR(i,n)    for(int i=0;i<(int)(n);++i)

#define MAX_N 1000
#define MAX_CHILD 10
#define MOD   1000000007

class Node {
public:
  enum {ELEM, ALT, OR};
  int m_type;
  bool m_opt;
  vInt m_child;

public:
  Node() {
    m_type = ELEM;
    m_opt = false;
  }
};

int Elem( const Node& node );
int Or( const Node& node );
int Alt( const Node& node );
int Sum( const Node& node );

int  nNode;
Node aNode[MAX_N];
```

問5　新薬開発

```
int Elem( const Node& node ) {
  ullong res = 1;
  ITER ( itN, node.m_child ) res = ( res*Sum( aNode[*itN] ) )%MOD;
  return node.m_opt ? (int)res+1 : (int)res;
}

int Alt( const Node& node ) {
  ullong res = 0;
  ITER ( itN, node.m_child ) res = ( res + Sum( aNode[*itN] ) )%MOD;
  return node.m_opt ? (int)res+1 : (int)res;
}

int Or( const Node& node ) {

  int aChild[MAX_CHILD];
  FOR ( i, node.m_child.size() ) {
    aChild[i] = Sum( aNode[node.m_child[i]] );
  }

  ullong res = 0;
  int m = 1 << node.m_child.size();
  for ( int i=1; i<m; ++i ) {

    ullong partSum = 1;
    FOR ( k, node.m_child.size() ) {
      if ( (i >> k) & 1 ) partSum = (partSum*aChild[k])%MOD;
    }

    res = (res + partSum)%MOD;
  }

  return node.m_opt ? (int)res+1 : (int)res;
}
```

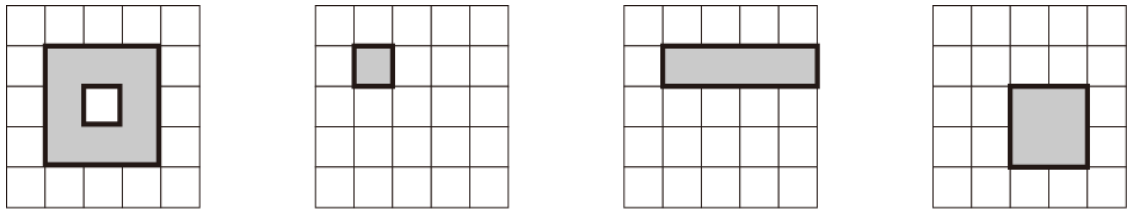
問5　新薬開発

```
int Sum( const Node& node ) {
  if ( node.m_type == Node::ELEM )    return Elem(node);
  else if ( node.m_type == Node::OR ) return Or(node);
  return Alt(node);
}

void Solve() { printf(“%d¥n”, Sum( aNode[0] )); }

main() {
  int s, t;
  char str[8];

  while ( scanf("%d", &nNode) == 1 ) {
    if ( nNode == 0 ) break;
    for ( int i=0; i<nNode; ++i ) {
      scanf("%s", str);

      Node& node = aNode[i];
      if      ( str[0] == 'E' ) node.m_type = Node::ELEM;
      else if ( str[0] == 'A' ) node.m_type = Node::ALT;
      else if ( str[0] == 'R' ) node.m_type = Node::OR;

      if ( str[1] == '?' ) node.m_opt = true;
    }

    for ( int i=0; i<nNode-1; ++i ) {
      scanf("%d %d", &s, &t);
      --s; --t;
      aNode[s].m_child.push_back( t );
    }
    Solve();
  }
}
```

問6　枠

問題概要

- Ｎ×Ｎのピクセルデータが与えられる
- 幅1のピクセルで作られる枠のうち、ピクセルデータの和の最大値を求める



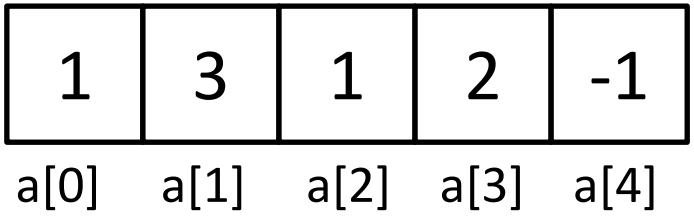
講評

- 提出数１１、正答数６.
- 累積和を利用したO(N^4)の解が、比較的思いつきやすい.
- しかし、N=300なのでO(N^4=81億)は時間制限で間に合わない.
- もうひと工夫必要だが、それが難しい.

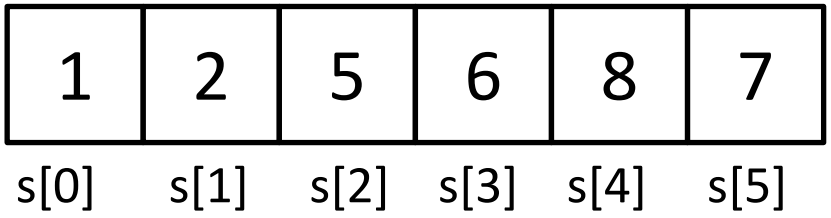
問6　枠

累積和とは？

- こういう1次元配列があるとき



- 累積和の配列sのi+1番目の要素は、配列aのi番目の要素以前を全て足したもの(ただし、要素を一つ増やす)



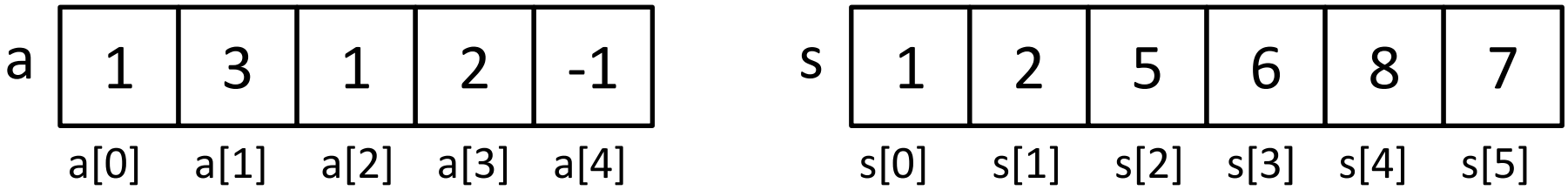
- O(n)で求まる (nに関する1重ループ)

```
s[0] = a[0];
for ( int i=0; i<n; ++i ) s[i+1] = s[i] + a[i];
```

問6　枠

累積和の何がうれしいのか？

- 累積和が分かれば、任意の区間の和が引き算一回で求まる！



例：a[2]からa[4]の和は？

s[4+1] – s[2] = 7 – 5 = 2 = a[2]+a[3]+a[4]

例：a[0]からa[0]の和は？

s[0+1] – s[0] = 2 – 1 = 1 = a[0]

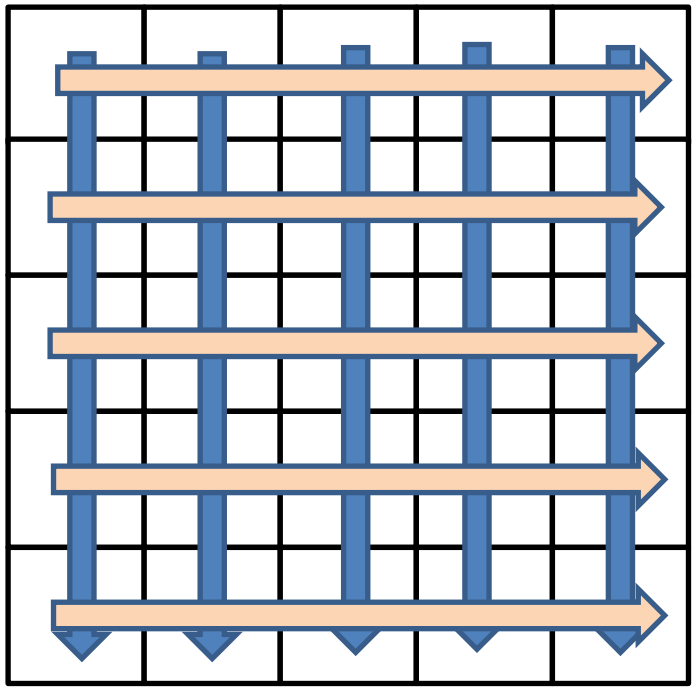
一般化：a[i]からa[j]の和は(j≧i)？

s[j+1] – s[i]

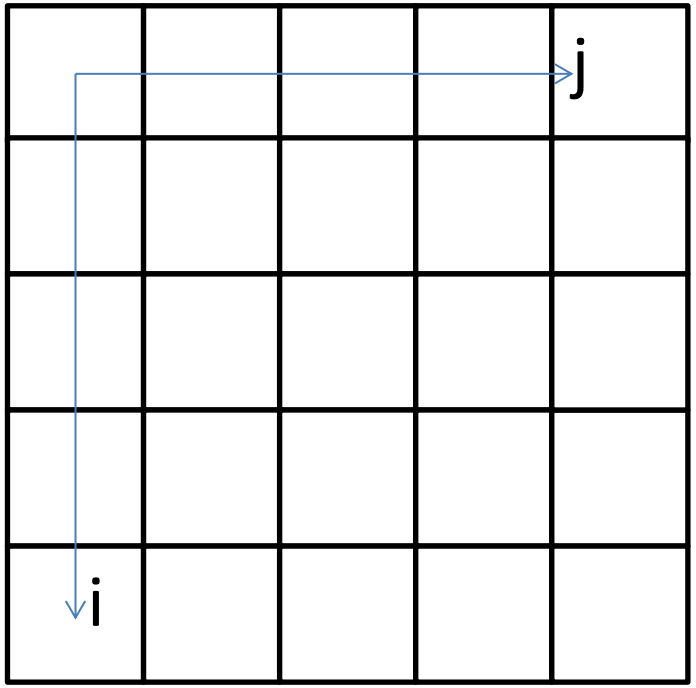
問6　枠

累積和を利用したO(N^4)解

1. 全ての行、全ての列について累積和を計算



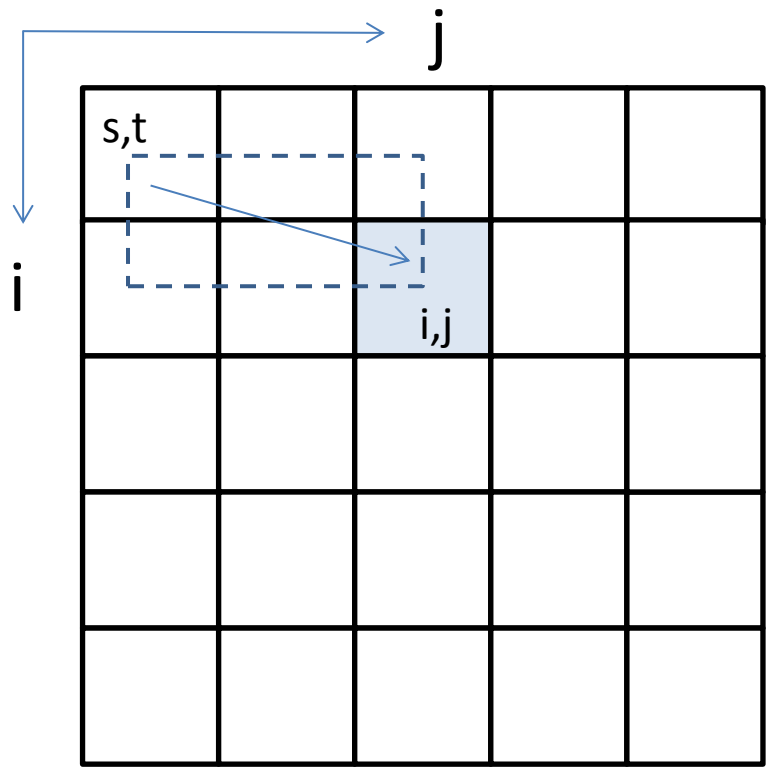
2. 縦(i)横(j)についてループを回す



問6　枠

累積和を利用したO(N^4)解

3. 現在のi,jの値について: i,jを右下の頂点とし、
　0≦s≦i, 0≦t≦jを左上の頂点とした枠を計算.　最大値であれば更新.



行と列についての累積和を計算済みなので、和の計算に関するループは無い.

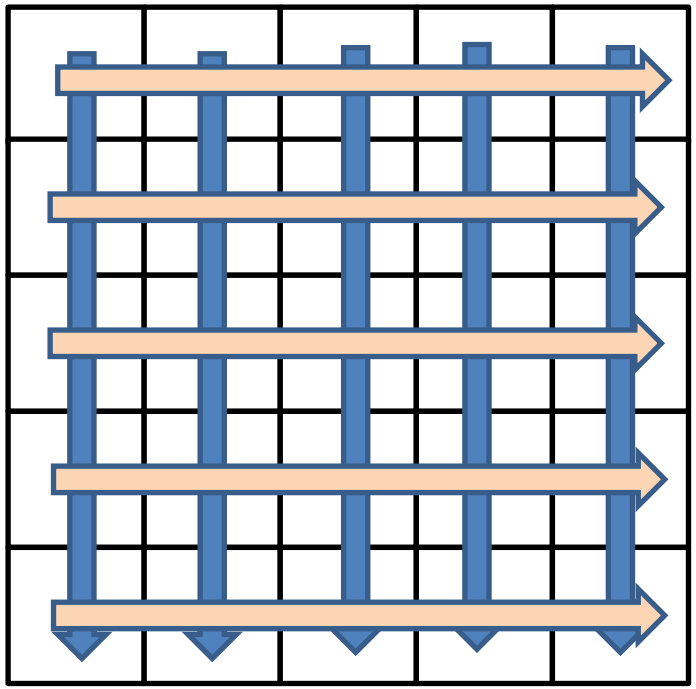
i,j,s,tに関する4重ループでO(N^4).

まだ間に合わない！

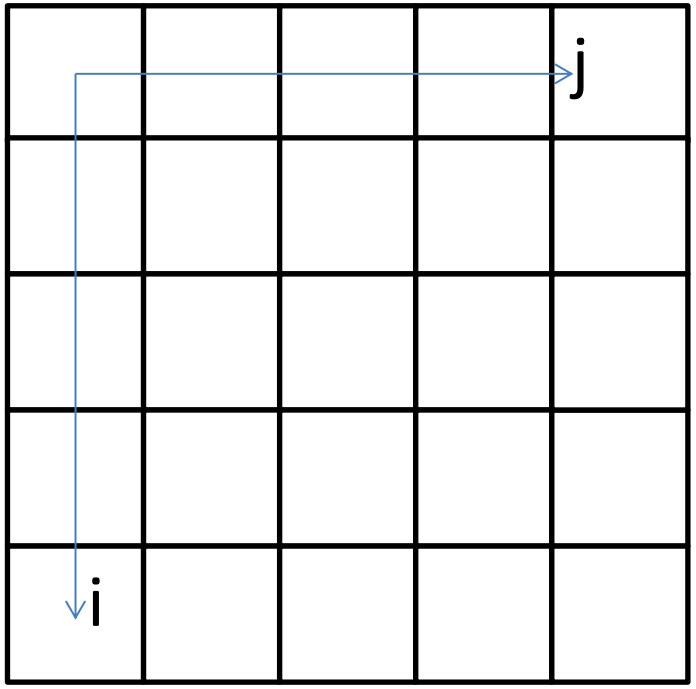
問6　枠

累積和を利用したO(N^3)解

1. 全ての行、全ての列について累積和を計算



2. 縦(i)横(j)についてループを回す



ここまではO(N^4)解と同様

3. i,jについてループを回し、行i, jのペアを上と下、列kを右端とした枠の最大値を求める． 行ペア(行iと行j) について、列kで左側の
⊏ 状の部分の最大値を求めていく

i=0, j=2, k=0

1	1	1		
2	-1	2		
-1	2	3		

i=0, j=2, k=1

1	1	1		
2	-1	2		
-1	2	3		

1	1	1		
2	-1	2		
-1	2	3		

i=0, j=2, k=2

1	1	1		
2	-1	2		
-1	2	3		

1	1	1		
2	-1	2		
-1	2	3		

問6　枠

```
#define MAX_N 300

int nSize;
int frame[MAX_N][MAX_N];

int maxSum;
int rowSum[MAX_N][MAX_N+1];
int colSum[MAX_N][MAX_N+1];

int RowSum(int i, int s, int e) { return rowSum[i][e+1] - rowSum[i][s]; }
int ColSum(int j, int s, int e) { return colSum[j][e+1] - colSum[j][s]; }

void InitSum() {
  for ( int i=0; i<nSize; ++i ) {

    rowSum[i][0] = 0;
    colSum[i][0] = 0;

    for ( int k=1; k<=nSize; ++k ) {
      rowSum[i][k] = rowSum[i][k-1] + frame[i][k-1];
      colSum[i][k] = colSum[i][k-1] + frame[k-1][i];
    }
  }
}
```

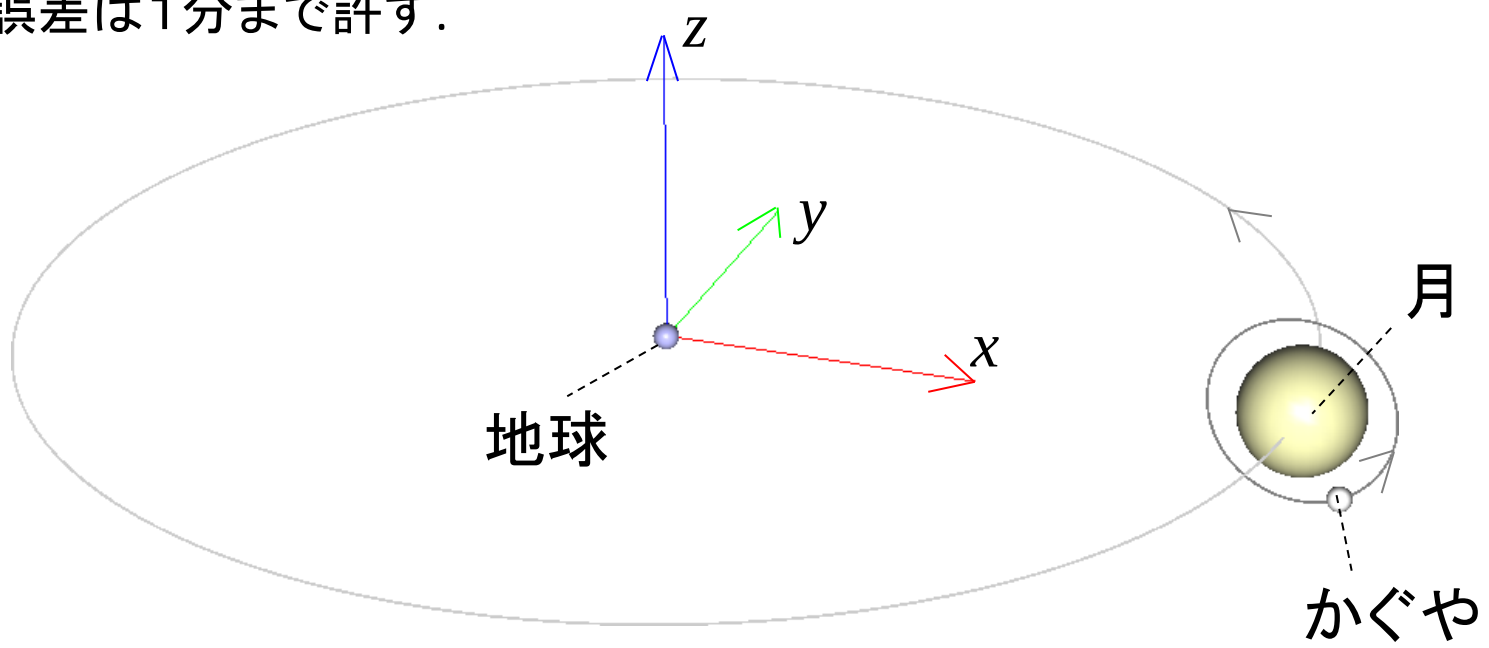
問6　枠

```
void Solve() {

  InitSum();

  for ( int i=0; i<nSize; ++i ) {

    for ( int j=i; j<nSize; ++j ) {

      maxSum = std::max( RowSum( i, i, j ), maxSum );//横一列のものや、1ピクセルのものを別に処理
      if ( j == i ) continue;

      int colMax = ColSum( 0, i, j );
      for ( int k=1; k<nSize; ++k ) {

          int col = ColSum( k, i, j );
          maxSum = std::max( maxSum, std::max( colMax + col, col ) );
          colMax = std::max( colMax + frame[i][k] + frame[j][k], col );
      }
    }
  }
  printf("%d¥n", maxSum);
}

main() {
  scanf("%d", &nSize);
  for ( int i=0; i<nSize; ++i ) {
    for ( int j=0; j<nSize; ++j ) scanf("%d", &frame[i][j]);
  }
  Solve();
}
```

問7　かぐや

問題概要

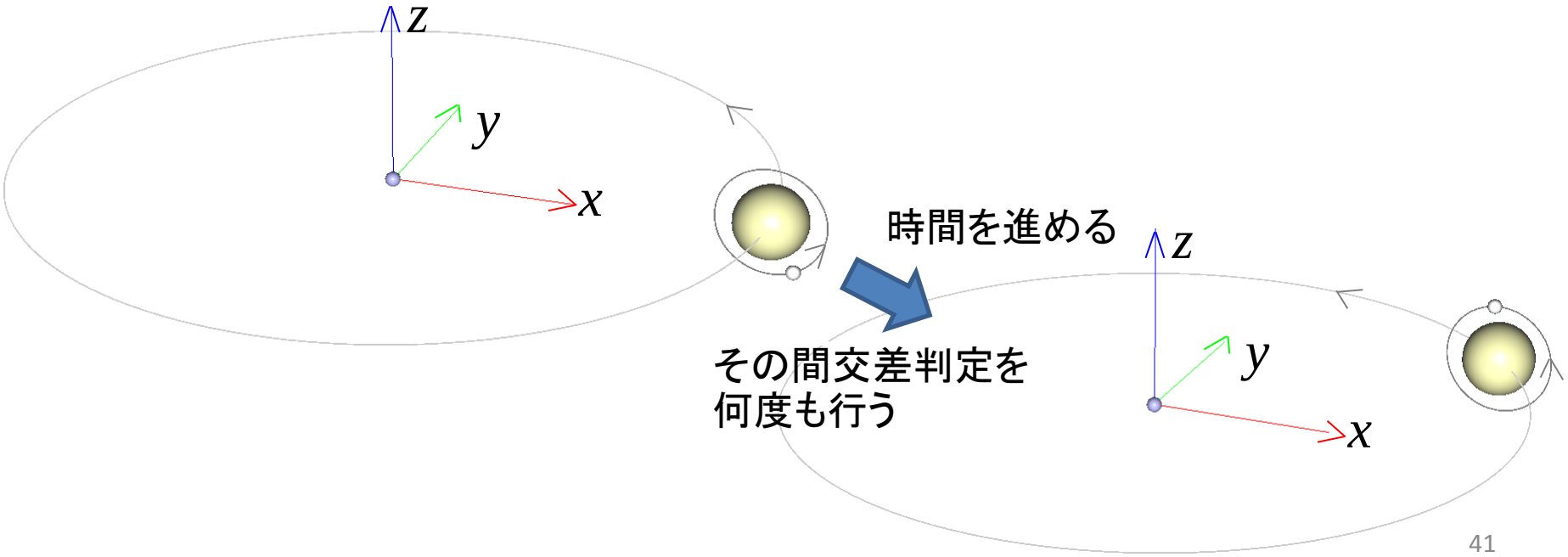
- 地球を原点とした空間座標に、地球の周りを回る月と、月の周りを回るかぐやがある.
- 地球とかぐやは点（大きさが0）、月は半径1800kmの球とみなす.
- 月はxy平面上、かぐやは月を中心としたxz平面と並行な平面上を回る
- 地球から見て、かぐやが月に隠れる総時間を求める
- 誤差は1分まで許す.



問7　かぐや

解法

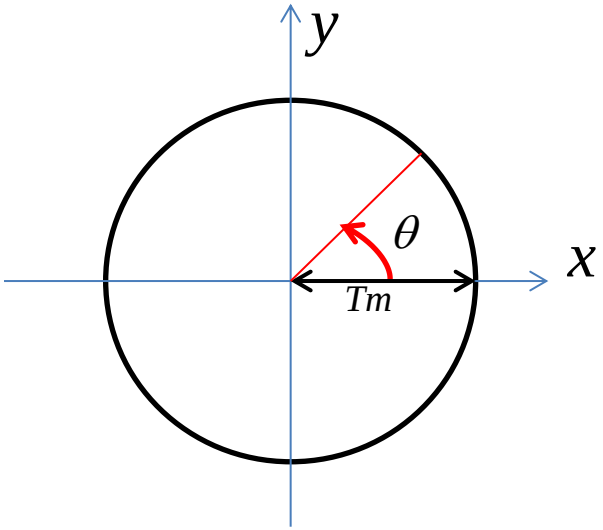
- 提出数7、正答数1.
- 幾何シミュレーション
- 時刻tの月とかぐやの位置を求める式を考える
- 地球から見たとき、かぐやが月に隠れているか、2次方程式の解で求める
- tを少しずつ増やして、隠れていれば結果にtの増分を加算していく



問7　かぐや

月の位置

- 半径がTm=380000kmで
  xy平面の円周上にある



cosとsinは単位円周上のxとy
座標なのでこう書ける

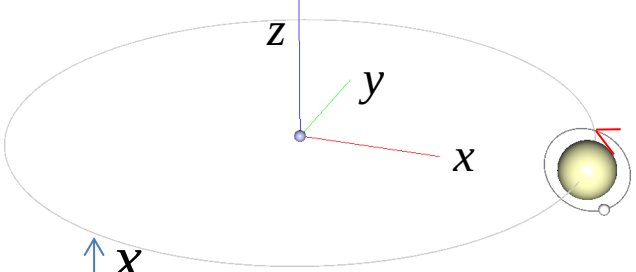
$$x_m = T_m \cos(a_m t)$$
$$y_m = T_m \sin(a_m t)$$
$$z_m = 0$$

tを経過時間として、amの値は？

- 2,500,000秒(=2,500,000/60分)で一周する

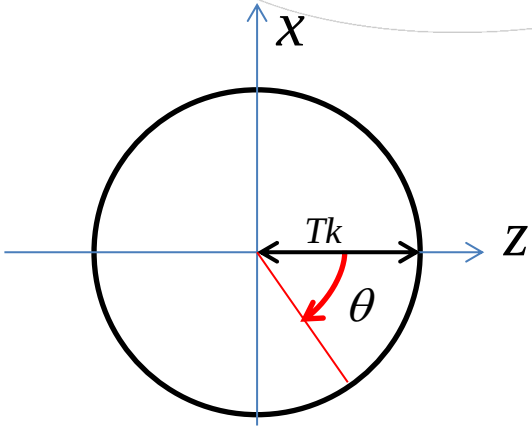
$$a_m = 2\pi/(\frac{2500000}{60})$$

問7　かぐや



かぐやの位置

- 半径がTk=(1800+100)kmで、月を中心とした、xz平面に平行な円周上にある



- かぐやの初期位置から考えると、xがsin, zがcosになっている.
- かぐやの回転の向きがz軸からx軸の負の方向なので、xにはマイナスが付く

軸をxとzと考えた円周上の座標
+ 月の座標

$$x_k = -T_k \sin(a_k t) + x_m$$
$$y_k = y_m$$
$$z_k = T_k \cos(a_k t) + z_m$$

- 2時間(=120分)で一周する　　$a_k = 2\pi/120$

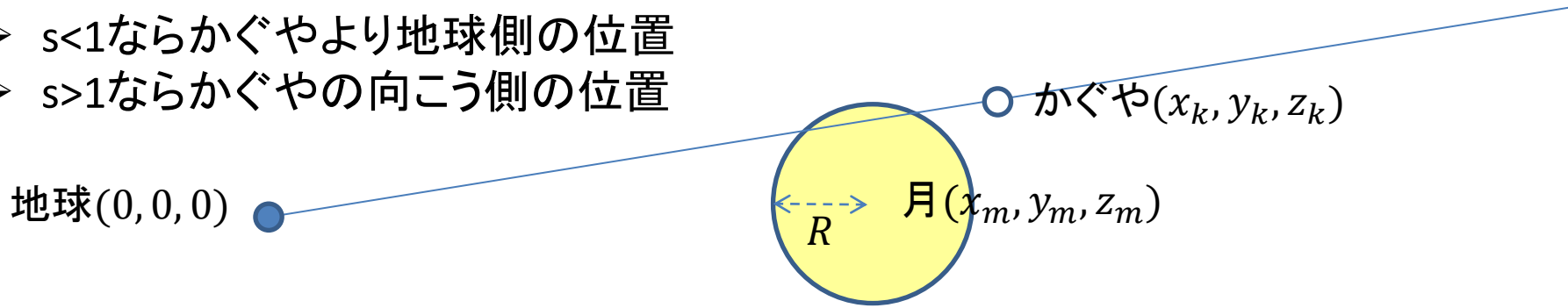
問7　かぐや

月の表面の場所を表す式：
「月の中心を基準として距離が一定(R=1800km)の場所」を2乗

$$(x-x_m)^2+(y-y_m)^2+(z-z_m)^2=R^2$$

地球からかぐやを見る視線：(sx_k, sy_k, sz_k)

- s=1ならちょうどかぐやの位置
- s<1ならかぐやより地球側の位置
- s>1ならかぐやの向こう側の位置



地球からかぐやを見る視線が、月の表面に触れるか？
という問題

問7　かぐや

月の位置を表す式に地球からかぐやを見る視線を代入：

$$(sx_k - x_m)^2+(sy_k - y_m)^2+(sz_k - z_m)^2= R^2$$

未知数sに関して整理すると、sに関する2次式にできる：

$$\left(x_k^2 + y_k^2 + z_k^2\right)s^2 - 2(x_k x_m + y_k y_m + z_k z_m)s + T_m^2 - R^2 = 0$$

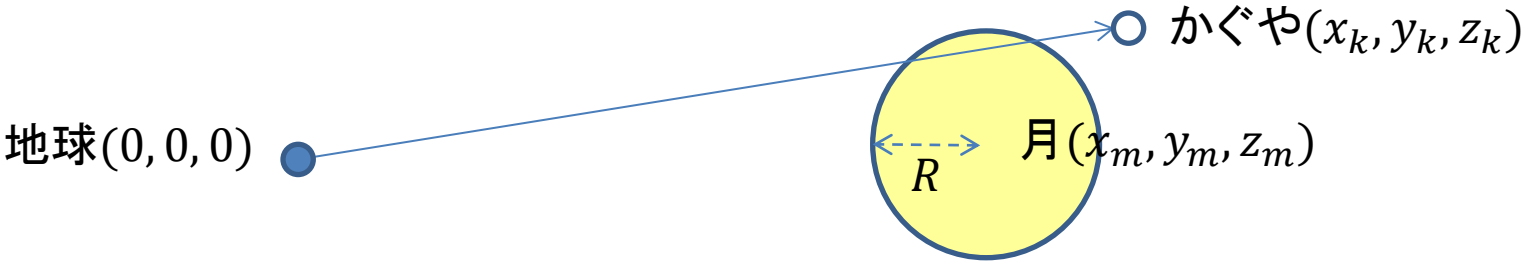
解の公式から：　$s = \dfrac{-b \pm \sqrt{b^2 - ac}}{a}$

ここで定数

$a = x_k^2 + y_k^2 + z_k^2$

$b = x_k x_m + y_k y_m + z_k z_m$

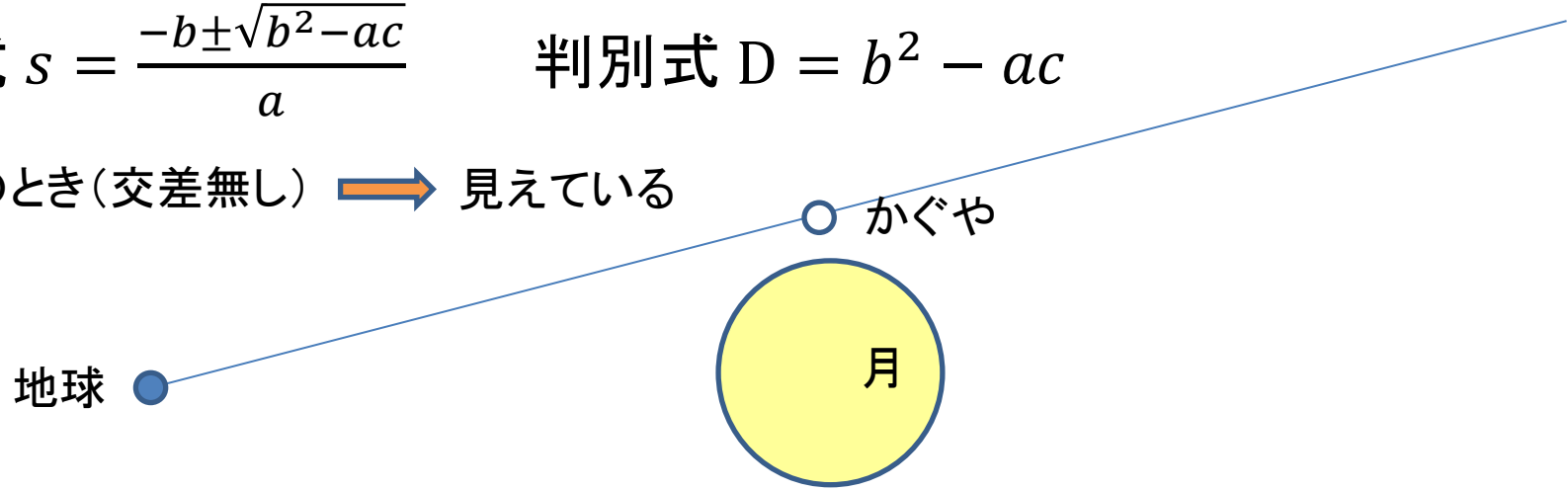
$c = T_m^2 - R^2$



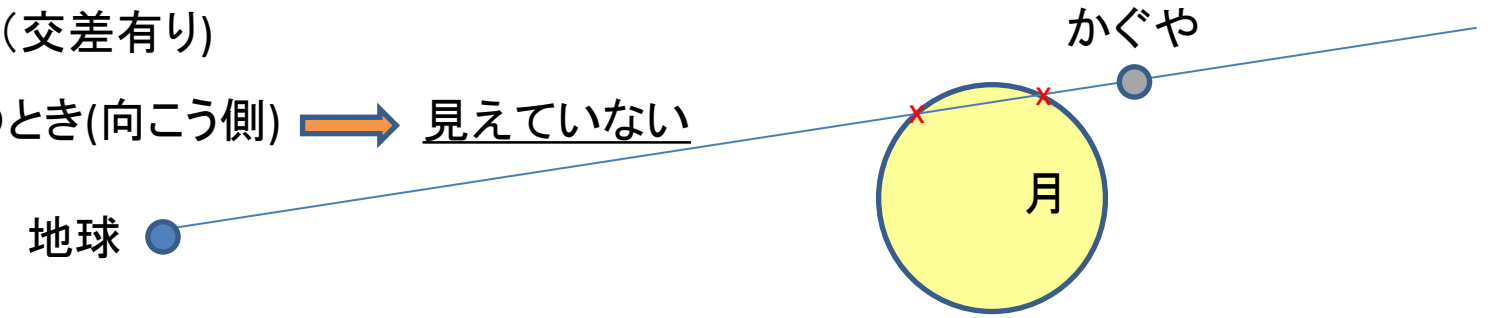
問7　かぐや

解の公式 $s = \frac{-b \pm \sqrt{b^2 - ac}}{a}$　　判別式 $D = b^2 - ac$

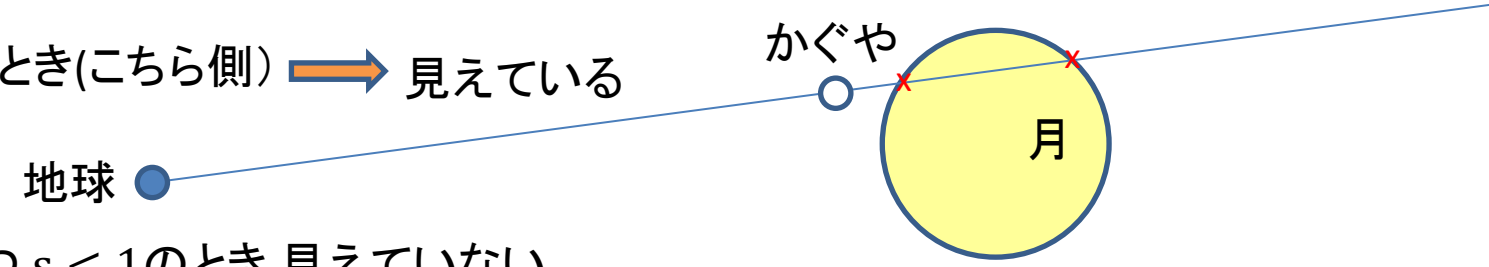
- D < 0のとき（交差無し） ⇒ 見えている



- D ≥ 0のとき（交差有り）
  - s < 1のとき(向こう側) ⇒ 見えていない



  - s > 1のとき(こちら側) ⇒ 見えている

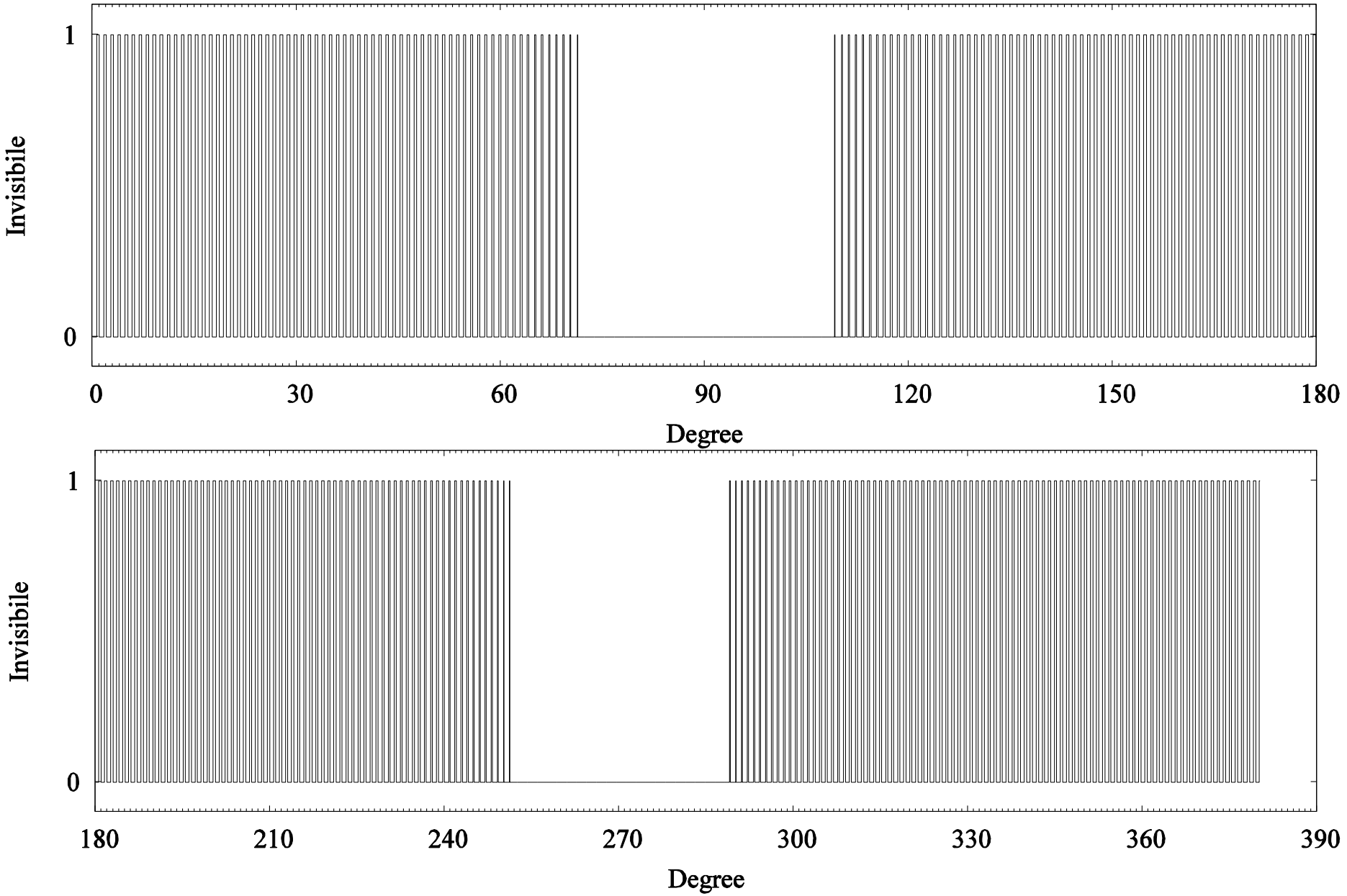


D ≥ 0のとき かつ s < 1のとき 見えていない.

tを少しずつ動かして毎回これを判定していく.

問7　かぐや

おまけ：不可視性の移り変わり



問7　かぐや

```
main() {

  double t_end, m0;
  scanf("%lf %lf", &m0, &t_end);
  double dt = 0.01;
  double t_curr = 0.0;

  double dR = 1800.0; //月の半径
  double dTm = 380000.0; //月の公転半径
  double dTk = dR + 100.0; //かぐやの公転半径
  double dWm0 = m0*M_PI/180.0;
  double dWm = 2.0*M_PI/(2500000/60.0);
  double dWk = 2.0*M_PI/(120.0);

  double dRes = 0.0;

  while ( t_curr + 0.5*dt < t_end ) {

    double cm = cos(dWm*t_curr + dWm0);
    double sm = sin(dWm*t_curr + dWm0);
    double ck = cos(dWk*t_curr);
    double sk = sin(dWk*t_curr);
```

つづく

10000分で可視不可視切り替えは
高々２＊10000分/120分≒167回

誤差1分以内なので刻み幅
1/167≒0．006あれば十分過ぎる
実際はΔt=0．01程度なら十分．

10000分までなので、計算量100万程度

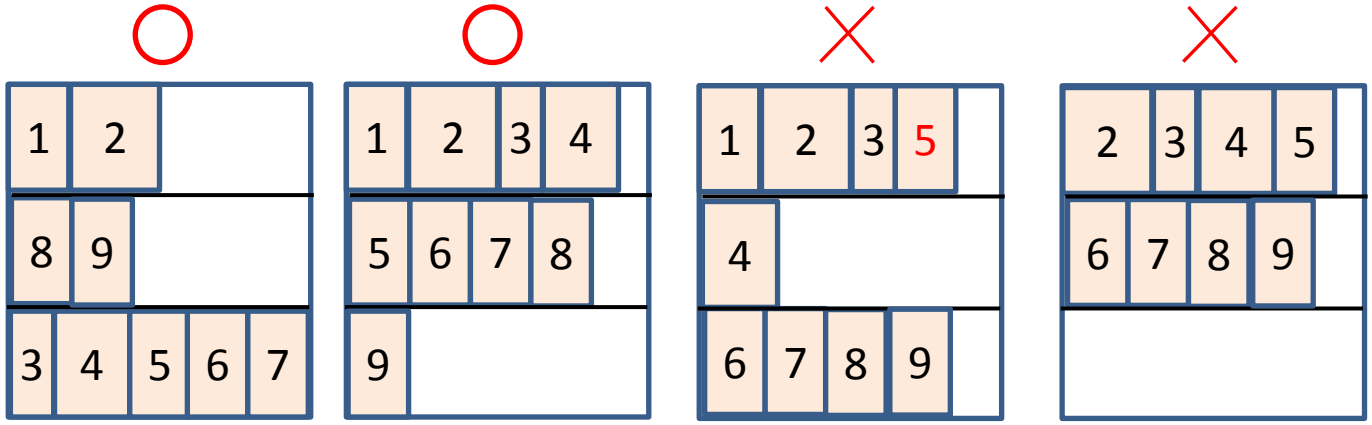
問7　かぐや

```
    double xm = dTm*cm;
    double ym = dTm*sm;
    double zm = 0.0;

    double xk = xm - dTk*sk;
    double yk = ym;
    double zk = zm + dTk*ck;

    double xk_leng2 = xk*xk + yk*yk + zk*zk;
    double dot = xk*xm + yk*ym + zk*zm;
    double d = dot*dot - xk_leng2*(dTm*dTm - dR*dR);

    if ( d > 0.0 ) {
      double s = (dot + sqrt(d))/xk_leng2;
      if ( s < 1.0 ) dRes += dt;
    }

    t_curr += dt;
  }

  printf("%lf¥n", dRes);
}
```

問8　ネットカフェ

問題概要

- 本棚に1巻目から最大何巻目まで置けるか？
- 本棚の各段中で、単行本の巻が連続した配置であれば良い.

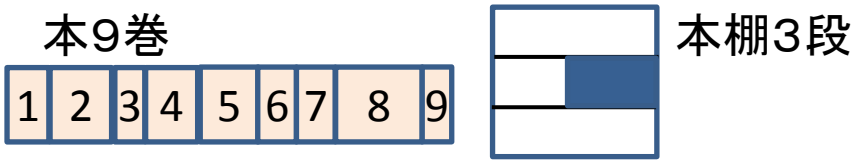


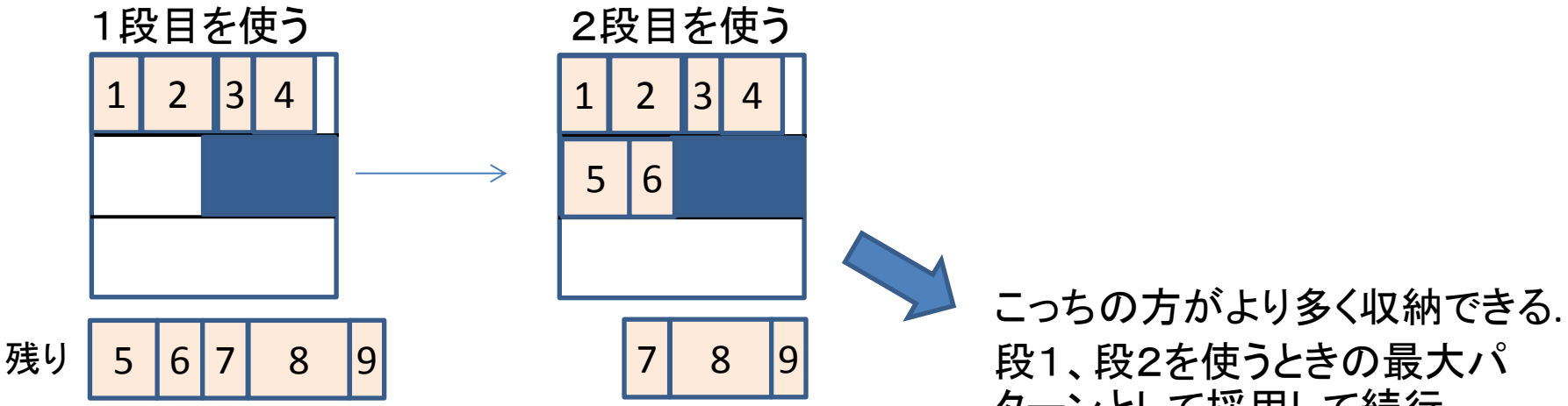
講評

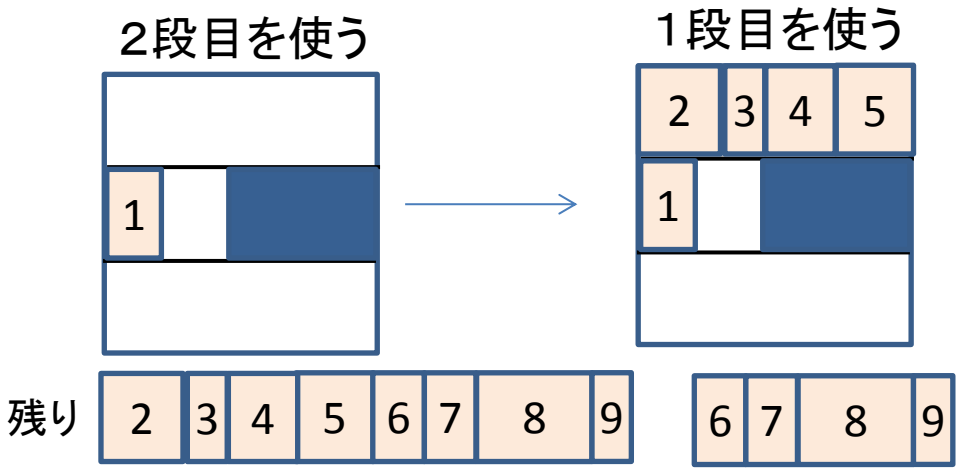
- 提出数19、正答数8.
- 巻数200,000、本棚の段数15まで.
- 本棚の各段の使い方が2^{15}通りある.
- 各段に本が入るか効率よく求める.

問8　ネットカフェ

例：


本9巻
1 2 3 4 5 6 7 8 9
本棚3段

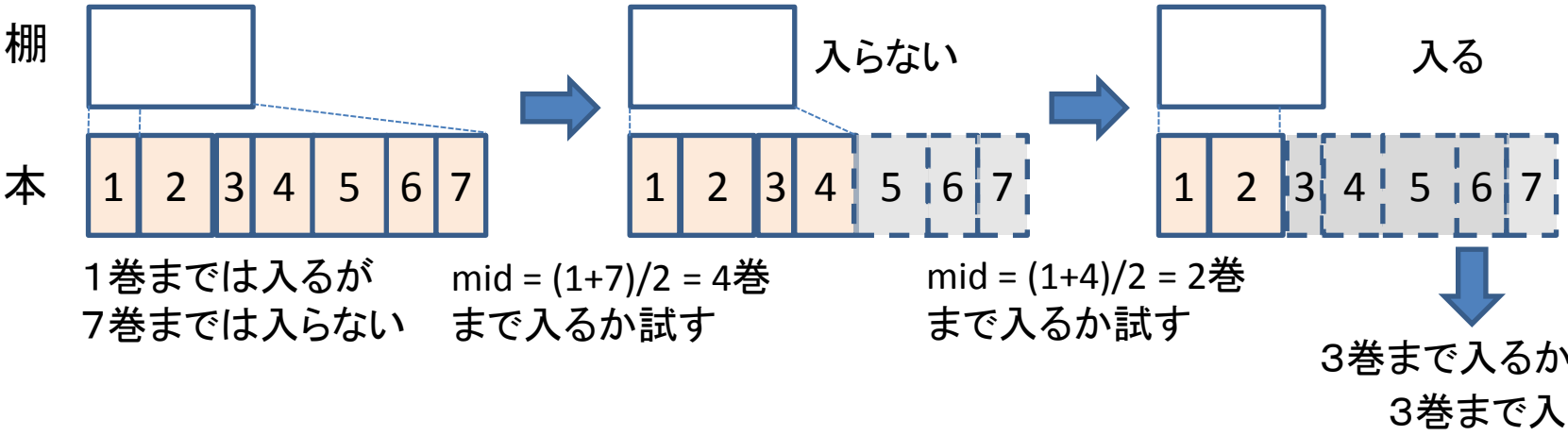

1段目を使う
1 2 3 4
残り
5 6 7 8 9
2段目を使う
1 2 3 4
5 6
7 8 9
こっちの方がより多く収納できる.
段1、段2を使うときの最大パターンとして採用して続行


2段目を使う
1
残り
2 3 4 5 6 7 8 9
1段目を使う
2 3 4 5
1
6 7 8 9

問8　ネットカフェ

解法

- 本棚の段の使い方をビット列としたbit DP (2^N).
- まだ使っていない段について、今までに入れた本から始まり何巻目まで入れられるかを2分探索 (1段あたりlogM).



- 本iから本jまでの合計厚さ、合計重さが定数時間で求まるように、累積和も計算しておく.

問8　ネットカフェ

```
typedef struct {
  int w, t;//重さ,厚さ
} Book;

typedef struct {
  int c, b;//重さ上限,厚さ上限
} Shelf;

#define MAX_M 1000000
#define MAX_N 10

int m, n;
Book book[MAX_M];
Shelf shelf[MAX_N];
int placeable[1<<MAX_N];
int maxPlaceable;

int bookWSum[MAX_M+1];
int bookTSum[MAX_M+1];

int BookWeightSum(int i, int j) { return bookWSum[j] - bookWSum[i-1]; }
int BookThickSum(int i, int j) { return bookTSum[j] - bookTSum[i-1]; }

bool IsPlaceable(int i, int s, int e) {
  if ( s > e ) return false;
  return BookWeightSum(s, e) <= shelf[i-1].c &&
    BookThickSum(s, e)  <= shelf[i-1].b;
}
```

問8　ネットカフェ

```
int PlaceableBook(int shelf, int book_start) { //shelf段目に book_start冊目から何冊目まで置けるか

  if ( !IsPlaceable(shelf, book_start, book_start) ) return book_start-1;
  if ( IsPlaceable(shelf, book_start, m) ) return m;

  int s = book_start;
  int e = m;
  while ( e - s > 1 ) {
    int mid = (s + e)/2;
    if ( IsPlaceable(shelf, book_start, mid) ) s = mid;
    else e = mid;
  }
  if ( IsPlaceable( shelf, book_start, e ) ) return e;
  return s;
}

void Check() {
  for ( int used=0; used<(1<<n); ++used ) {//使用済み段の全パターン
    for ( int i=0; i<n; ++i ) {

      if ( !( (used>>i)&1 ) ) {//i段目をまだ使っていないなら調べる
          int b = PlaceableBook(i+1, placeable[used]+1);
          int used_update = used | (1<<i);
          placeable[used_update] = std::max(placeable[used_update], b);
          if ( placeable[used_update] > maxPlaceable )
            maxPlaceable = placeable[used_update];
          if ( maxPlaceable == m ) return;
      }
    }
  }
}
```

問8　ネットカフェ

```
void Solve() {

  bookWeightSum[0] = 0;
  bookThickSum[0] = 0;
  for ( int i=0; i<m; ++i ) {
    bookWeightSum[i+1] = bookWeightSum[i] + book[i].w;
    bookThickSum[i+1] = bookThickSum[i] + book[i].t;
  }

  std::fill(placeable, placeable+(1<<n), 0);
  maxPlaceable = 0;
  Check( );
  printf("%d¥n", maxPlaceable);
}

main() {
  while ( scanf("%d %d", &m, &n) == 2 ) {

    if ( m == 0 && n == 0 ) break;

    for ( int i=0; i<m; ++i ) scanf("%d %d", &book[i].w, &book[i].t);
    for ( int i=0; i<n; ++i ) scanf("%d %d", &shelf[i].c, &shelf[i].b);

    Solve();

  }
}
```

問9　未知の病原菌

問題概要

- アクダマキンとゼンダマキンから成る一列の鎖が与えられる.
- 鎖は任意の位置で切断、または2つの鎖の端同士で結合できる.
- アクダマキンの数がゼンダマキンの数よりも多くなるような切断は禁止.
- 長さが2以下の鎖に分解可能か.
- 可能であればその操作を出力（最短である必要はない）.

講評

- 提出数5、正答数2.
- 禁止操作以外、いろいろやってください.

問9　未知の病原菌

- x（アクダマキン）の方が多い鎖は-1.
- xとo（ゼンダマキン)が同じ数なら「xo」と「ox」のみから成る鎖以外、xの方が多い鎖ができてしまう.
  - xoとoxのみの場合、前から2番目で切ることを繰り返す(split 鎖ID 1)
  - そうでなければ-1
- oの方が多いとき、いろいろな方法がある. (泥臭くて可)

問9　未知の病原菌

- おおざっぱな方法

1. どのような鎖でも、oの方がxより一つ多いようなsub鎖を作れる

例1　o7個x6個: oxoxxxoooxxoo:　左から9個目

例2　o7個x6個: oxoxxxoxxoooo:　左から13個目（切断なし）

2. そのようなsub鎖では、oが右端に必ずある

例1　oxoxxxooo　xxoo
例2　oxoxxxoxxoooo

3. xが右端に現れる一つ手前まで、右端のoを切断して左端に結合することを繰り返す

例1　ooooxoxxxo
例2　oooooxoxxxoxxo

4. 右端のxoを、安全に切り出し消滅させることができる

例1　o7個x6個: oooxoxx　xo
例2　o7個x6個: ooooxoxxxox　xo

問9　未知の病原菌

- おおざっぱな方法

5a. 右端がoなら、3から繰り返す

例1　ooooxoxxxo
例2　oooooxoxxxoxxo

5b. 右端がxで、左端がoなら、左端のoを右端に結合させてxoをさらに消せる

例1　o7個x6個：oooxoxx　xo
例2　o7個x6個：ooooxoxxxox　xo

5c. 右端がxで、左端がxなら1から繰り返す

6. oが一つだけ残るので、切断で残された側のsub鎖の右端にoを一つ結合して、同じように消していく

ただし、oだけの鎖になった場合、無限ループになってしまうので、
oだけになったら、前から2つずつ切って終わるようにする.

問9　未知の病原菌

- おおざっぱな方法

操作回数のおおまかな見積もり(操作制限回数2万回以内に終わるか？）

- 両端がxのパターンでも、一度のsplitでoを右端に露出させることができる.
- そのような鎖で、鎖の長さ回数程度のsplit&joinでxoを一つ消せる.
- 菌xoのペアを消すために、長さ(n)回程度の操作
- n回くらいあれば十分間に合う(10000回もかからない)
- 多少無駄な操作が入っても十分余裕がある

問10　アカベコ国王の配慮

問題概要

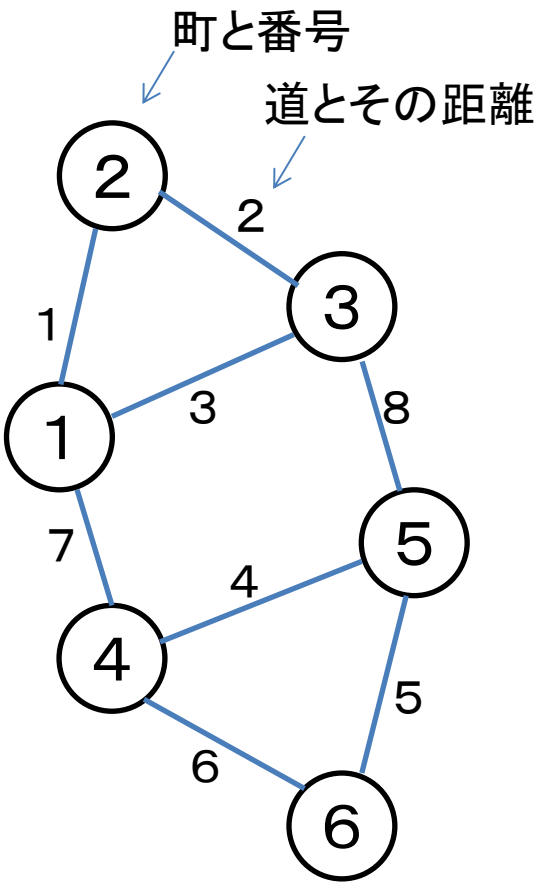
- アカ国とベコ国に町と道を分配する.
- 両国の間の距離は、両国の街をつなぐ道のうち最短のもの.
- 両国の間の距離が最大になる分配方法の数と、その最大値を求める.
- 町は全て配分されるが、配分されない道はありえる.

講評

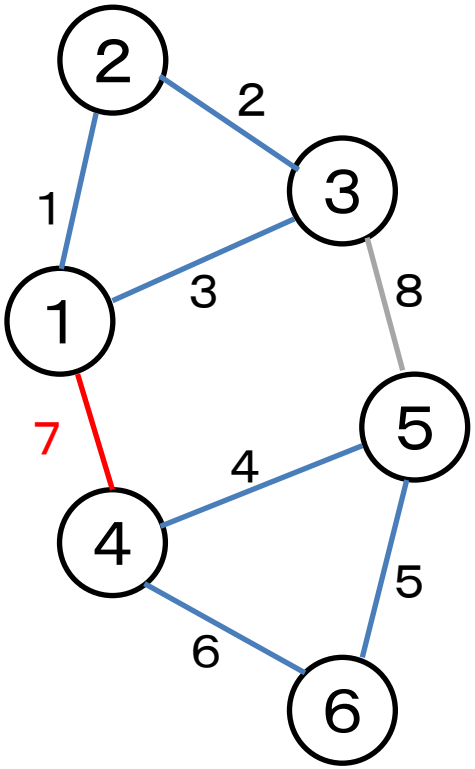
- 提出数0、正答数0.
- グラフに関する高度な知識と実装力.

問10　アカベコ国王の配慮

例1　普通のデータ



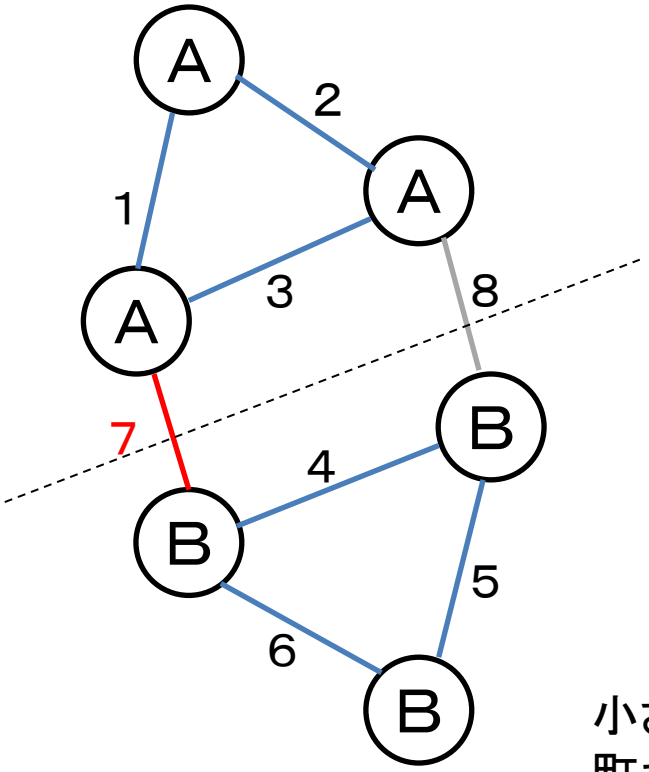
分割後の2国間の最大距離の最小値7



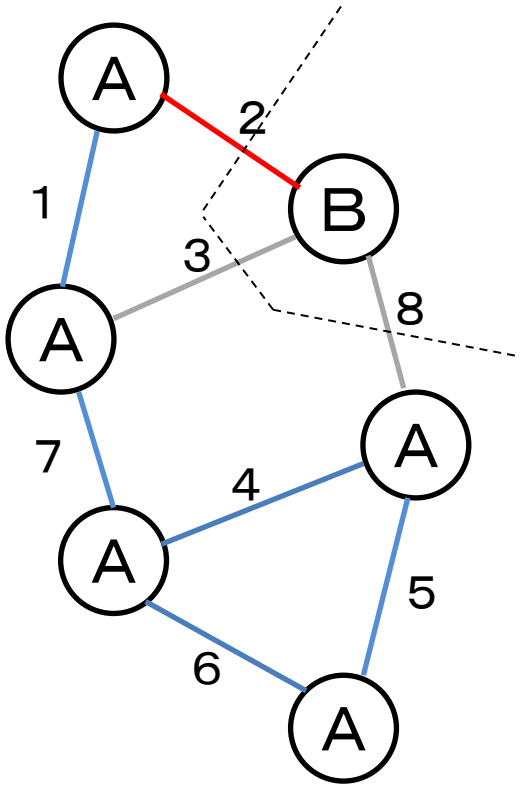
問10　アカベコ国王の配慮

例1

分割後の2国間の最大距離の最小値7



最大距離の最小値7を8にはできない



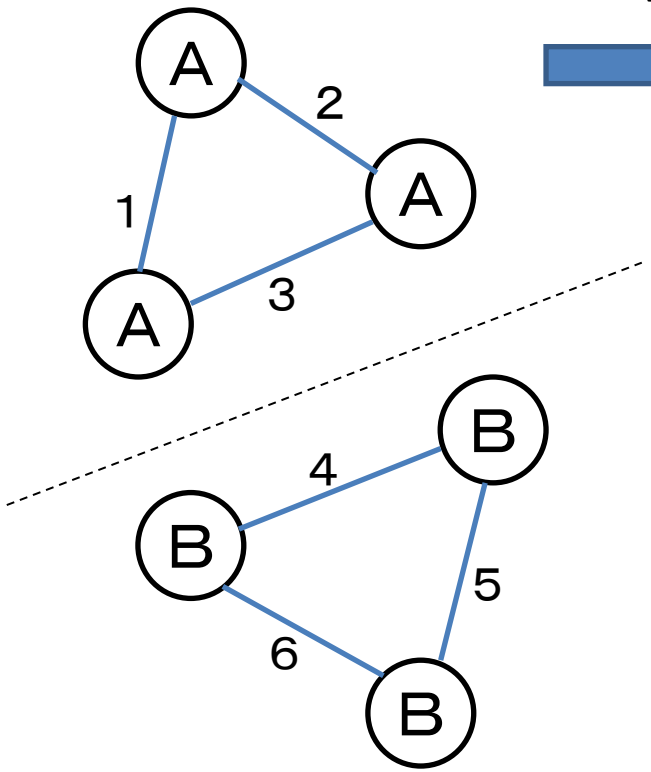
小さな辺から使用していって、全ての町が連結になったときの辺のコストが、最大距離の最小値になる

➡ クラスカル法

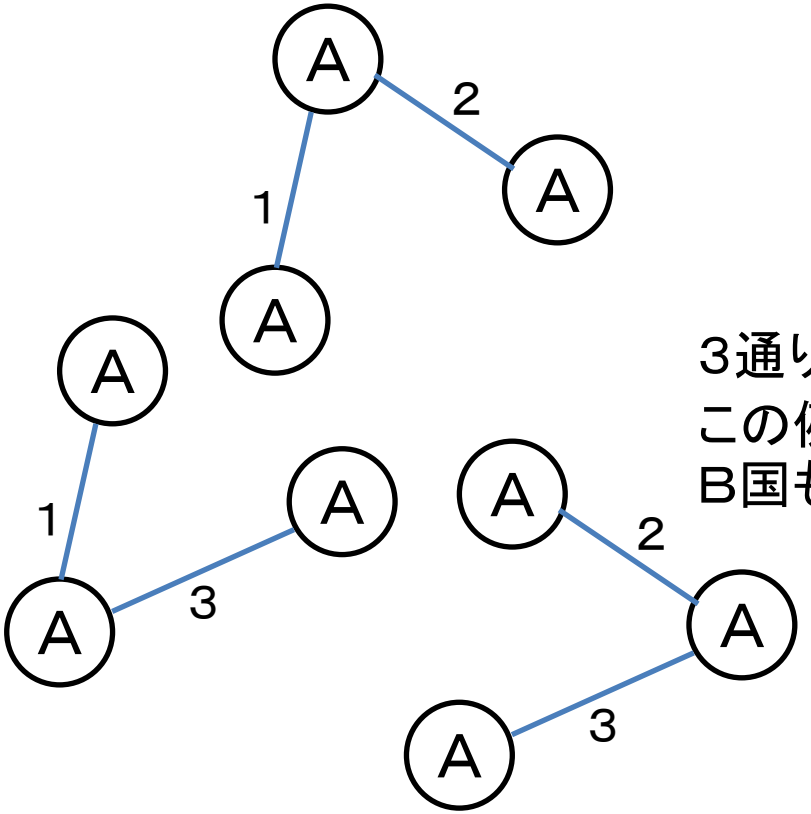
問10　アカベコ国王の配慮

例1

分配方法の数はいくつあるか？



A国



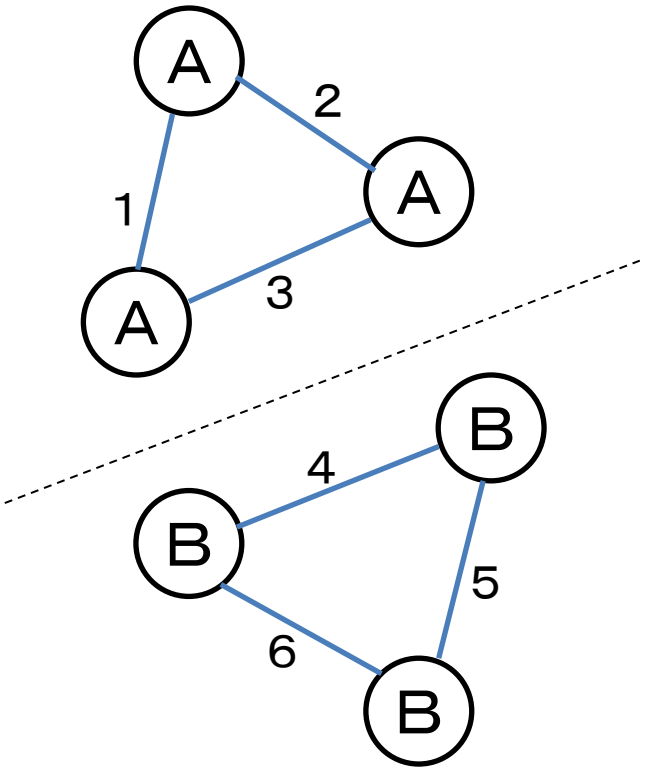
3通り
この例では、
B国も同様

A国の町を参照する辺で作られる全域木の総数

問10　アカベコ国王の配慮

例1

分配方法の数はいくつあるか？



A側のLaplacian行列の行列式：

対角成分がエッジ数
非対角がエッジ有無
（隣接行列ではない）

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

N行目
N列目
を消す

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

これの行列式

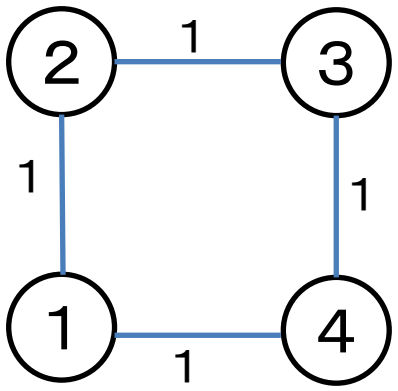
行列式 2*2 – 1 = 3 (行列演算ライブラリを利用するのが良い)

A国内では3通り。同様にB国内では3通り。
全部で、３＊３＝９通り
さらにAとBを入れ替えて倍 = １８通り

問10　アカベコ国王の配慮

例2　最短になる最大距離のエッジが複数あるとき
（5本まであり得る）

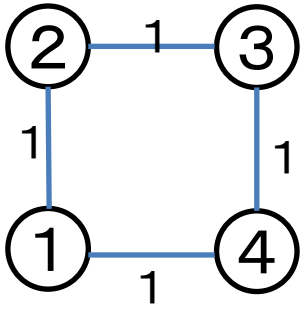
ここからが本当の地獄！

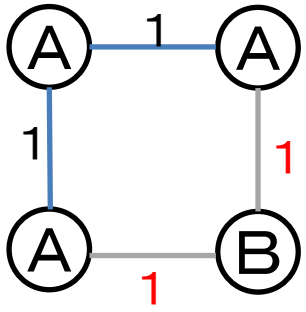


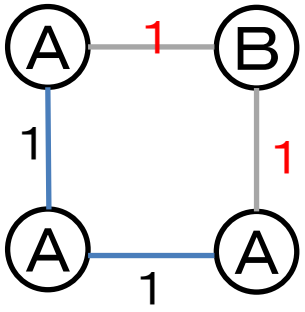
問10　アカベコ国王の配慮

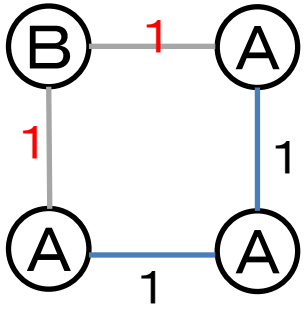
例2　最短になる最大距離のエッジが複数あるとき
（5本まであり得る）

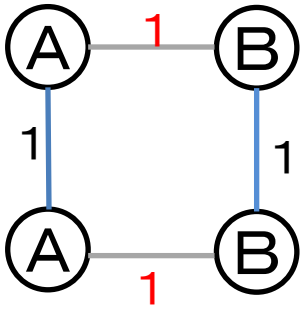
町の分け方自体がいろいろある











等々.

辺の使い方を全て試し(2^6通りまで)、それらの全域木の数を計算し、全て足す(modを忘れずに).

問10　アカベコ国王の配慮

問題概要（グラフ）

- 重み付き無向グラフG(V,E)が与えられる.
- 端点がそれぞれ集合A,集合Bに含まれるような辺の重みの最小値が最大となるように、各頂点をA,Bどちらかに分類し、それぞれで全域木を作る方法の数をmod 1,000,000,007で求めよ.

問10　アカベコ国王の配慮

解法

- 重みの小さな辺から順に採用していく.
- 全体が連結になったときに採用した辺の重みをwとする.
- このwが、両国の間の距離が最大となる距離.※証明後述
- w未満の辺を使って、グラフの連結成分を全て求める.
- 同じ重みの辺は5本以下なので、この連結成分の数は6以下である。
- あらためて、各連結成分をA,Bに分類すれば、全域木はどのように作っても良い.
- 分類の組み合わせ数は最大2^6.　ただし、全ての辺を使うものと、一つの辺も使わないものは除く.
- 全域木の数はラプラシアン行列を用いると求まる（行列木定理）. この計算量は$O(|V|^3)$.

問10　アカベコ国王の配慮

「w=最小全域木に含まれる最大の辺の重み」の証明

グラフG(V,E)において、任意の頂点集合A,B(A≠Φ, B≠Φ, A∪B=V,A∩B=Φ)とする. このとき、A-B間の最短辺の重みの最大値は、Gの最小全域木Tに含まれる最長辺の重みに等しい.

（証明）

Tに含まれる辺の集合をE'とし、E'の中で最長辺の重みをwとする。A-B間の少なくとも一つの辺がE'に含まれるのは明らか. よって、Tに含まれるA-B間の辺の重みはw以下なので、A-B間の最短辺の重みの最大値は、Gの最小全域木Tに含まれる最長辺の重みに等しい.（証明終了）

ここで、クラスカル法を考える。木の構築の際に全体が連結になる直前に追加する辺の重みはwであり、この時、重みがw以上の辺しか残っていない. よって、A-B間の最短辺の重みの最大値はwは、クラスカル法で最後に追加された辺の重みである.