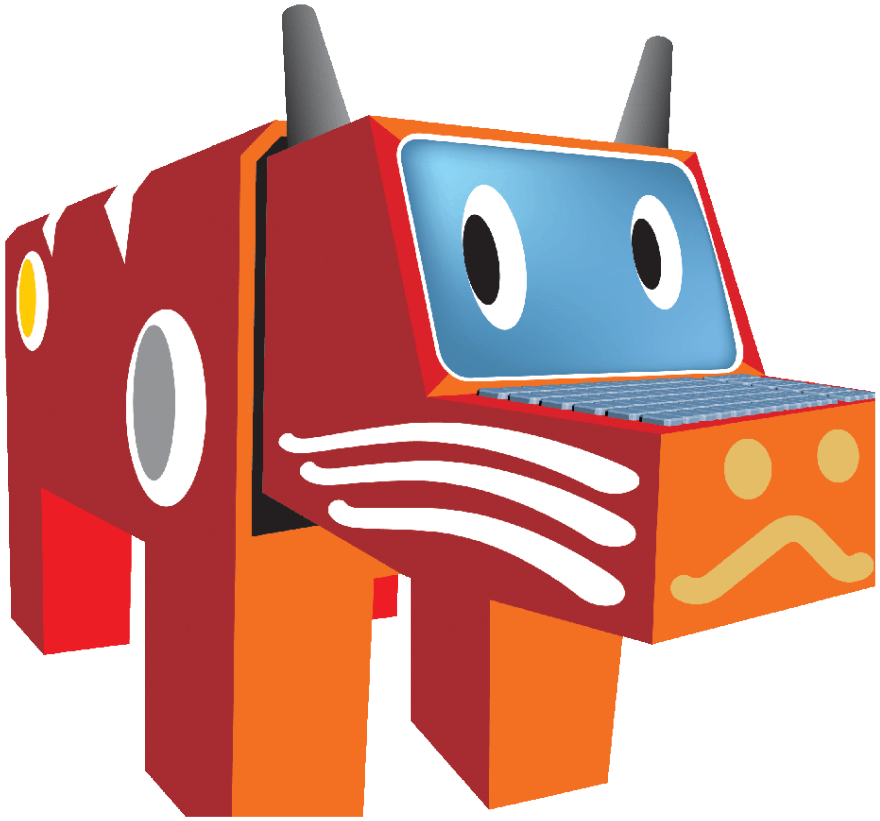
パソコン甲子園2013予選

プログラミング部門　解説



会津大学

問1　気温の差

問題概要

- 7 個の整数の組 a と b (a ≧ b)を読み込み、それぞれについて差 (a-b) を計算する.

講評・解法

- 変数宣言、代入演算、減算、標準入出力処理が行えるかを問う最も基本的な問題である.
- 7 回の入力・減算・出力処理でも書けるが、解答例のようなループ構造を用いて繰り返し処理を行う方が望ましい.

問1　気温の差

Cによる解答例

```
#include <stdio.h>

int main(){
    int i, nHigh, nLow;

    for ( i = 0; i < 7; i++ ){
      scanf("%d %d", &nHigh, &nLow);
      printf("%d¥n", nHigh - nLow);
    }

    return 0;
}
```

問1　気温の差

C++による解答例

```
#include <iostream>

int main(){
    int nHigh, nLow;

    for ( int i = 0; i < 7; i++ ){
        std::cin >> nHigh >> nLow;
        std::cout << nHigh - nLow << std::endl;
    }

    return 0;
}
```

問1　気温の差

Javaによる解答例

```
import java.io.*;
import java.util.*;

class Main{
    private void compute(){
        Scanner scanner = new Scanner(System.in);
        for ( int i = 0; i < 7; i++ ){
            int nHigh = scanner.nextInt();
            int nLow  = scanner.nextInt();
            System.out.println(nHigh - nLow);
        }
    }

    public static void main(String a[]){
        new Main().compute();
    }
}
```

問2　チケットの売り上げ

問題概要

- S席，A席，B席，C席の 4 種類の席が有り、各席の売り上げ枚数が与えられたとき、席の種類ごとに売り上げ金額の合計を求める.
- S席6000円、A席4000円、B席3000円、C席2000円と値段が決まっている.
- 入力ではS席は1、A席は2、B席は3、C席は4と表される.
- 各チケットの売り上げ枚数はS，A，B，Cの順番に与えられるとは限らない.

講評・解法

- 入出力処理に加えて、簡単な分岐処理を行う、または配列を使ったプログラムが実装できるかが問われている.
- 繰り返し処理を必要としない、最も簡単な部類の問題である.

問2　チケットの売り上げ

Cによる解答例: if-elseによる条件分岐

```
#include <stdio.h>

main(){
  int nType, nNum;

  for ( int i = 0; i < 4; i++ ){
    scanf("%d %d", &nType, &nNum);

    if      (nType == 1) printf("%d¥n", nNum*6000);
    else if (nType == 2) printf("%d¥n", nNum*4000);
    else if (nType == 3) printf("%d¥n", nNum*3000);
    else                 printf("%d¥n", nNum*2000);
  }
}
```

問2　チケットの売り上げ

C++による解答例: 配列

```
#include <iostream>

main(){
  int nType, nNum;
  int aPrice[4] = {6000, 4000, 3000, 2000};

  for ( int i = 0; i < 4; i++ ){
    std::cin >> nType >> nNum;
    std::cout << aPrice[nType-1]*nNum << std::endl;
  }
}
```

問2　チケットの売り上げ

Javaによる解答例：　switch文

```
class Main{
    public void run(){
        Scanner sc = new Scanner(System.in);
        for (int i=0; i<4; i++) {
           int nType = sc.nextLine();
           int nNum  = sc.nextLine();
           switch(nType) {
              case 1: System.out.println(6000*nNum); break;
              case 2: System.out.println(4000*nNum); break;
              case 3: System.out.println(3000*nNum); break;
              case 4: System.out.println(2000*nNum); break;
           }
        }
    }

    public static void main(String[] a){ new Main().run();}
}
```

問3　入場料金

問題概要

- 使用したい「入浴券」と「プール券」の枚数と、各券の単価が与えられたとき、一番安く済む合計料金を求める.
- ただし、以下のことに注意:
  - 「入浴券5枚以上かつプール券2枚以上」でまとめて買うと、すべての券が2割引になる.
  - 実際使用する枚数より多く券を買うことで合計料金が安くなるなら、買った券すべてを使わなくともよい.

問3　入場料金

講評・解法

- 割引に関するルールより

  使用したい「入浴券」と「プール券」の枚数が割引ルール外でも、あえて余分に買って割引ルールを適用したほうが安い場合がある.

  例：　入浴券1000円、プール券100円、入浴券6枚、プール券0枚の場合

  入浴券6枚を買う　→ 1000円 × 6枚 = 6000円

  入浴券6枚とプール券2枚を買う(割引ルール適用可能になる)

  → (1000円 × 6枚 + 100円 × 2枚)を2割引 = 6200円 × 0.8 = 4960円

- 券の単価は100円以上1000円以下で、50円刻みなので、2割引き(0.8掛け)でも小数点以下を考慮する必要が無い.
- 使用したい券が入浴券5枚以上かつプール券2枚以上という状況なら明らかに割引ルールを使った方が安い.
- そうでない場合だけ、余分に買って割引ルールを使った方が良いか判断する.

問3　入場料金

C++による解答例(1)

```
#include <cstdio>
#include <algorithm> //std::min

int Normal(int x, int y, int b, int p)   { return x*b + y*p; }
int Discount(int x, int y, int b, int p) { return 4*Normal(x,y,b,p)/5;}

main() {
  int n, x, y, b, p;
  while (scanf("%d", &n) == 1 && n != 0) {
    while (n--) {
      scanf("%d %d %d %d", &x, &y, &b, &p);
      if (b >= 5 && p >= 2) printf("%d¥n", Discount(x, y, b, p));
      else  printf("%d¥n", std::min(Normal(x, y, b, p),
                                    Discount(x, y, (b < 5) ? 5 : b,
                                             (p < 2) ? 2 : p)));
    }
  }
}
```

問3　入場料金

C++による解答例(2)

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
  int N, x, y, b, p;
  cin >> N;
  for(int i = 0; i < N; ++i) {
    cin >> x >> y >> b >> p;
    cout << min(x*b+y*p, (x*max(b,5)+y*max(p,2))*4/5) << endl;
  }
  return 0;
}
```

問4　おそろいの景品

問題概要

- あるカプセル玩具自販機(ガチャポン)で、景品が何種類あるかと、各々の景品がいくつ残っているかわかっている.
- このとき、同じ景品を必ず2つ以上得るには最悪何回引けばよいかを求める.
  - 最悪何回か　→　想定される最悪の順番で景品が出たときの挑戦回数

講評・解法

- 最悪のケースとは何かを思考する必要がある.
  - 最悪のケースは、自販機の中に残っている全種類の景品を1つずつ手に入れてしまう場合. そのとき、もう一度引けば確実に同じ景品が2つ得られる.
  - つまり、残り数0以外の景品の数+1が答え.
  - 例外として、2つ以上残っている景品が1種も無ければ不可能(NAと出力).
- 思いつけば実装は比較的単純.

問4　おそろいの景品

Cによる解答例:

```
main() {

  int i, n, k;
  while (scanf("%d", &n) == 1 && n != 0) {

    int nLess1Cnt = 0;
    int nZeroCnt = 0;

    i = n;
    while (i--) {
      scanf("%d", &k);
      if (k <= 1) ++nLess1Cnt; //NAでないかチェック
      if (k == 0) ++nZeroCnt;  //残り数0の景品をカウント
    }

    if (n == nLess1Cnt) printf("NA¥n");
    else printf("%d¥n", n - nZeroCnt + 1);
  }

}
```

問5　坊主めくりの結末

問題概要

- 百人一首の札を使ったゲーム、「坊主めくり」のシミュレーションをする問題.
- ６４枚の「男」、１５枚の「坊主」、２１枚の「姫」、計１００枚の札.
- N人が札山から順番に1枚ずつ札を取る.
  - 引いた札が男なら、引いた人はその札を手に入れる.
  - 引いた札が坊主なら、引いた人はその札を含め、持っている札をすべて「場」に出す.
  - 引いた札が姫なら、引いた人はその札を含め、場にある札をすべて手に入れる.
- N人が最終的に持っている札を昇順(小さい方から順番)にならべた結果を出力する. さらに場に残っている札の数を一番最後に出力する.

問5　坊主めくりの結末

講評・解法

- ルールを再現するシミュレーションができるか.
- 多少のイレギュラー(場札の数)を考えた昇順ソートができるか.
- 比較的単純なシミュレーションと基本的なアルゴリズム(ソート)を使いこなせるかを問う問題.

問5　坊主めくりの結末

C++による解答例(シミュレーションから出力まで):

```
char c;
int aCard[nPlayer+1]; //各人の持ち札数．aCard[nPlayer]は場札数とする
int nTurn = 0; //現在誰が引く番か
for (int i=0; i<100; ++i) {
  scanf("%c", &c);
  switch (c) {
  case 'M': ++aCard[nTurn]; break;
  case 'S':
    aCard[nPlayer] += aCard[nTurn] + 1;//引いた人の札+1を場へ
    aCard[nTurn] = 0; break;
  case 'L':
    aCard[nTurn] += aCard[nPlayer] + 1;//場札+1を引いた人の札へ
    aCard[nPlayer] = 0; break;
  }
  nTurn = (nTurn + 1)%nPlayer;
}
std::sort(aCard, aCard + nPlayer);

for (int i=0; i<nPlayer; ++i) printf("%d ", aCard[i]);
printf("%d¥n", a[nPlayer]);
```

問6　陣形

問題概要

- 3種類の文字C、A、Nのそれぞれの個数が与えられる.
- これらを使って、3文字の文字列CCA、CCC、またはCANが最大何組作れるか求めよ.
- 各文字の個数の最大値は1000.

講評

- 作った文字列の個数が最大になるような文字の選び方を行う、アルゴリズムを考える必要がある.
- 問題の仕様から自らアルゴリズムを設計する能力が求められている.
- プログラムの実装は簡単.

問6　陣形

解法(1):貪欲法

- CAN → CCA → CCC の順番で可能な限り組を作る.

C++による解答例

```
void solve(){
  int C, A, N, T = 0;

  cin >> C >> A >> N;

  for(; C > 0 && A > 0 && N > 0; T++ ){
    C--; A--; N--;
  }

  for(; C >= 2 && A > 0; T++ ){
    C-=2; A--;
  }

  for(; C >= 3; T++ ){
    C-=3;
  }

  cout << T << endl;
}
```

問6　陣形

解法(2)：総当たり

- CAN、CCAの個数の組み合わせ全てについて、作成可能かどうかを調べる.
- CCCを考慮して最大値を計算.

C++による解答例

```
void solve(int c, int a, int n){
  int maxp = 0;

  for ( int i = 0; i < MAX; i++ ){
    for ( int j = 0; j < MAX; j++ ){
      if ( (i+2*j <= c && i+j <= a && i <= n) ){ // i個のCAN,j個のCCAが可能か？
        maxp = max(maxp, i+j+ (c - (i+2*j))/3);
      }
    }
  }

  cout << maxp << endl;
}
```

問7　プログラミングコンテスト

問題概要

- Nチームがプログラミングコンテストに参加している.
- コンテスト中、各チームの得点が加点・減点される.
- その時点で最も得点の高いチームがテレビに映る.
- コンテスト開始から終了までについて、テレビに映った時間帯が最も長いチームを報告せよ.
- チーム数N≦100,000、得点増減のレコードの数R<1,000,000.

問7　プログラミングコンテスト

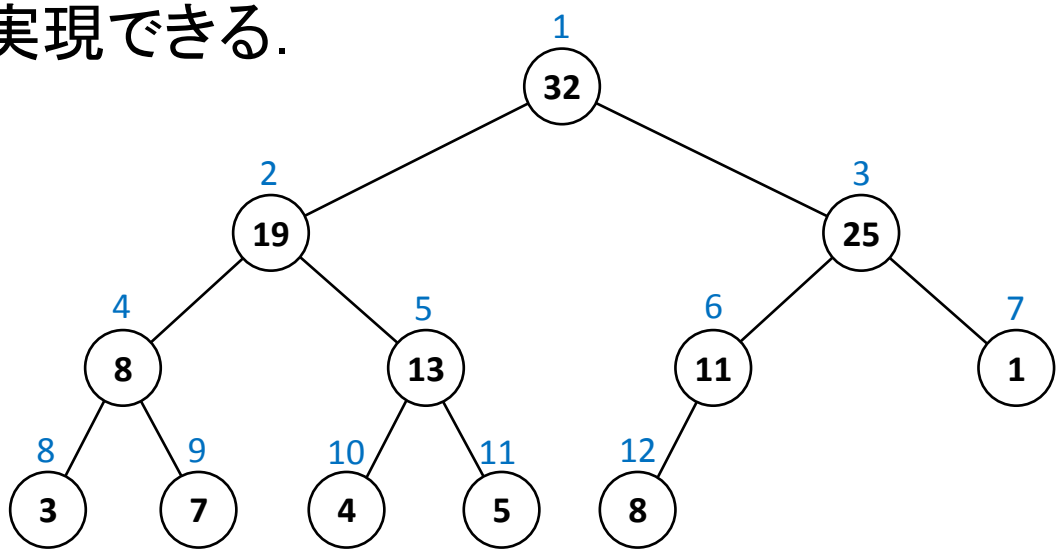
講評

- チーム数が多いため、愚直にシミュレーションを行うと制限時間オーバーで不正解となる.
- 現時点で最高得点のチームを効率よく保持・管理するためのデータ構造が必要.
- やや高等的なデータ構造が実装できるかが問われています.
- データ構造を独自に実装する、あるいは標準ライブラリのデータ構造を有効利用できるかが問われています.

問7　プログラミングコンテスト

解法（1）:ヒープを実装する

- ヒープ（完全2分木）を用いて最高得点チームを保持する.
- 節点の値がその親の値以下（Maxヒープの条件）であるヒープをMaxヒープと呼ぶ.
- ヒープは1次元配列で実現できる.
  - 節点の番号をkとすると
  - 親の番号は k/2
  - 左の子の番号は 2k
  - 右の子の番号は 2k + 1



1	2	3	4	5	6	7	8	9	10	11	12
32	19	25	8	13	11	1	3	7	4	5	8

問7　プログラミングコンテスト

解法（1）:ヒープを実装する

- 加点された場合は、Maxヒープを維持するために、その節点から根（上）に向かって親と交換していく（upHeap操作）.
- 減点された場合は、Maxヒープを維持するために、その節点から葉（下）に向かって子の大きい方と交換していく（downHeap操作）.
- upHeap, downHeapともに計算量はO（log N）.
- 全体の計算量は O(R log N).

問7　プログラミングコンテスト

C++による解答例:ヒープの実装例

```
void downHeap(int i){
  int l, r, largest;
  l = 2*i;
  r = 2*i+1;
  if ( l <= N && isGreater(H[i], H[l]) ) largest = l;
  else largest = i;
  if ( r <= N && isGreater(H[largest], H[r]) ) largest = r;
  if ( largest != i ){
    swp(i, largest); // i と largest の親子関係を交換
    downHeap(largest);
  }
}

void upHeap(int i){
  while ( i > 1 && isGreater(H[i/2], H[i]) ){
    swp(i, i/2);
    i = i/2;
  }
}

void update(int d, int t, int x){
  S[d] += x;
  if ( x > 0 )      upHeap(P[d]);
  else if( x < 0 ) downHeap(P[d]);
}
```

問7　プログラミングコンテスト

C++による解答例：priority_queueを応用する

```
priority_queue<pair<int, int> > q;
int N, R, L, S[1000001], T[1000001], d, t, x;  // T[i]にチームiの合計時間を記録

scanf("%d %d %d", &N, &R, &L);
q.push(make_pair(0, -1));

for ( int i = 0; i <= N; i++ ) S[i] = T[i] = 0;
int pret = 0, pred = 1;

for(int i = 0; i < R; i++){
  scanf("%d %d %d", &d, &t, &x);
  S[d] += x;
  q.push(make_pair(S[d], -d));
  while(1){
    pair<int, int> p = q.top();
    if(p.first == S[-p.second]){
       T[pred] += t - pret;
       pred = -p.second;
       pret = t;
      break;
    }
    q.pop();
  }
}
T[pred] += L - pret;
```

問8　勉強会

問題概要

- N人の生徒から何人かがリーダとなり勉強会を開催する.
- リーダ以外の生徒は、自分のスコアよりも低いスコアのリーダが開催する勉強会には参加できない.
- また、0以上のある値 r を1つ決め、勉強会に参加する生徒とリーダのスコアの差が r 以内となるようにする.
- 以下の操作をシミュレートするプログラムを作成せよ:
  - 生徒をリーダに加える.
  - 生徒をリーダから外す.
  - 要求時点でのリーダの組み合わせについて、どの勉強会にも参加できない生徒が x 人以下になるような、最小の r を求める.
- 生徒の数≦1,000,000、処理要求の数≦1000、スコア≦10^9
- シミュレーション中にリーダの数Mが100を超えることはない.

問8　勉強会

講評

- シミュレーションを行いｒを愚直に求めると制限時間オーバーで不正解となる.
- 大きなデータに対して、効率的な探索アルゴリズムあるいはデータ構造が実装できるかが問われている.

問8　勉強会

解法

- 生徒スコアの上限が大きいので、あるスコアを持つ生徒の人数の累積和を求めておく.
- 累積和によって、r が与えられたとき「スコアの範囲内に何人の生徒がいるか」を効率よく探索できるようになる.
- 与えられた X に対して2分探索によって r を求める.
- 計算量
  - 累積和の計算　　N
  - r の決定　　log MAX(スコアの最大値)
  - 累積和から範囲が重ならないように、それぞれのリーダについて調べ、勉強会に参加できる生徒の数を数える　　M(リーダの数)
  - 全体の計算量 O(N + M log MAX)

問8　勉強会

C++による解答例：スコアの累積和の計算

```
ATI.push_back(-1);
ATV.push_back(0);
int pre = 0;
for ( map<int, int>::iterator it = PM.begin(); it != PM.end(); it++ ){
   ATI.push_back((*it).first);
   ATV.push_back((*it).second + ATV[pre++]);
}
```

問8　勉強会

C++による解答例：r について参加可能な人数の計算

```
int getCount(int r){
  seg.clear();

  if ( MS.size() > 0 ) seg.push_back(make_pair(MS[0]-r, MS[0]));

  for ( int i = 1; i < MS.size(); i++ ){
    int b = MS[i]-r;
    int e = MS[i];
    if ( b <= seg[seg.size()-1].second){
      seg[seg.size()-1].second = max(seg[seg.size()-1].second, e);
    } else {
      seg.push_back(make_pair(b, e));
    }
  }

  int cnt = 0;

  for ( int i = 0; i < seg.size(); i++ ){
    int a = distance(ATI.begin(), lower_bound(ATI.begin(), ATI.end(), max(0, seg[i].first) )) - 1;
    int b = distance(ATI.begin(), lower_bound(ATI.begin(), ATI.end(), max(0, seg[i].second) ));
    cnt += ATV[b] - ATV[a];
  }

  return cnt;
}
```

問8　勉強会

C++による解答例：Xに対するｒの2分探索

```
void bsearch(int X){
  init();

  int l = 0;
  int r = MAX_P+2000000;
  int m;
  int diff = r - l;
  int cnt = getCount(r);

  if ( cnt < X ){ cout << “NA” << endl; return;}

  while(1){
    m = (l+r)/2;
    cnt = getCount(m);
    if ( cnt >= X ){
      r = m;
    } else if ( cnt < X ){
      l = m;
    }
    if ( diff == l - r ) break;
    diff = l - r;
  }
  cout << r << endl;
}
```

問9　ハッピーエンド問題

問題概要

- xy平面上に最大N=40個の点が与えられる.
- N個の点で作成可能な単純凸k角形のうち、面積が一番小さいものを求める.
- kは3からNでクエリ(質問)として与えられる.
- 入力される点の座標は全て異なり、どの3点も同一直線上に無い.

問９　ハッピーエンド問題

講評

- 幾何学の問題で、計算幾何学ライブラリ等を駆使しながら、作成可能な単純凸k角形のみに絞り面積を計算するような方法を考え、実装が行えるかが問われている.
- 全ての点の組み合わせを重複なく試すと、最悪計算量$_{N}C_{N/2}$で時間切れとなる.

解法

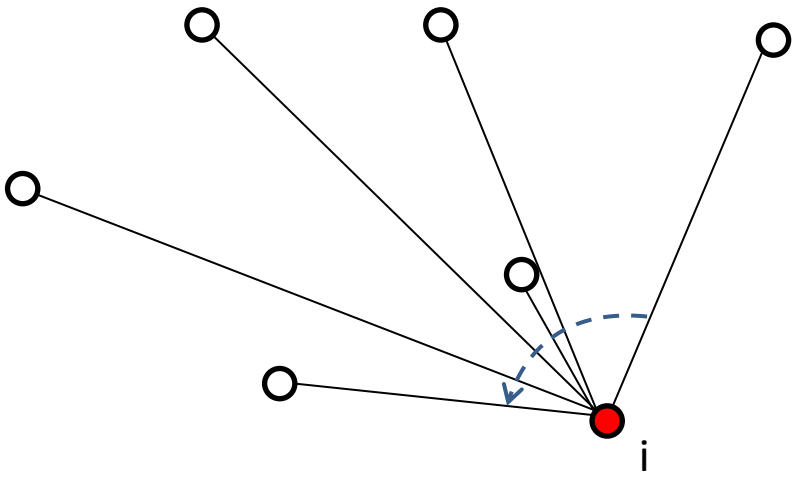
- 動的計画法（DP: Dynamic Programming）

問9　ハッピーエンド問題

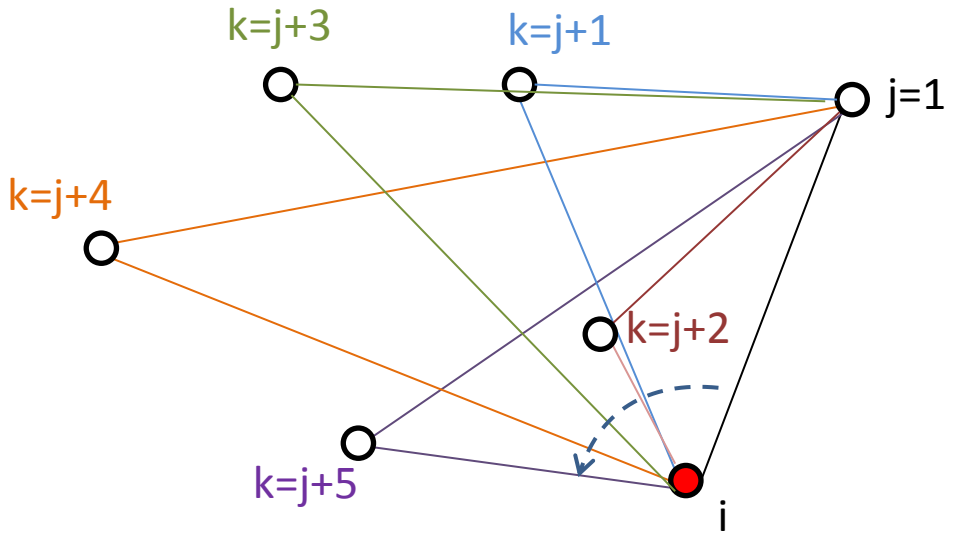
解法

- 現在までにできたm角形の端に新たに3角形を付け加えてm+1角形を作るとき、凸多角形の条件を崩さないものに関してのみ面積を計算し、最小のものを記録していく.

- 現在一番下にある点(複数あればそのうち一番左にある点)を基準点iと決め、他の点を反時計回り順にソートしておく(同一直線状に3点が無いという制限から一意にソート可能)

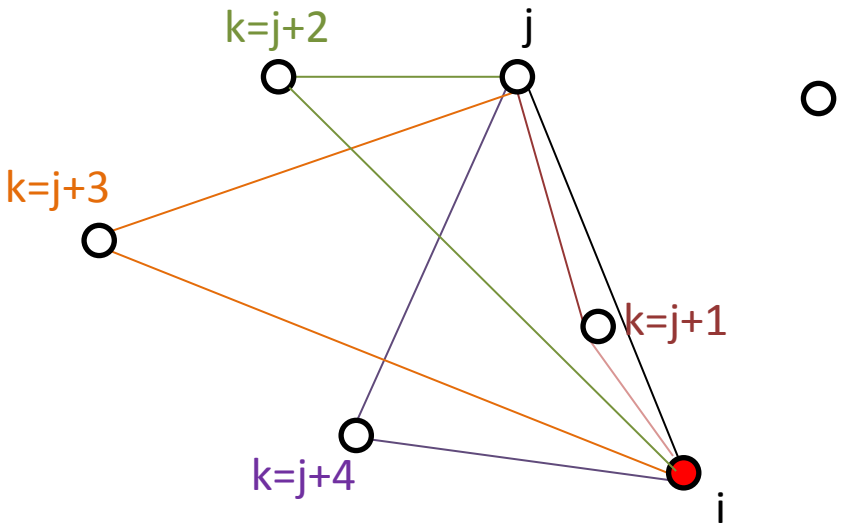


- 反時計回りで一番小さい角度の点をj=1とする. k=j+1からk=Nまでi,j,kを繋ぐと、線分ijを辺の一つとする三角形を全て作れる



問9　ハッピーエンド問題

- 同様に、反時計回りで2番目に小さい角度の点を次のjとしk=j+1からk=Nまでで三角形を作る. これをj=N-1まで行う (点はiを基準に角度ソート済みである)



このように、iを固定したまま、jを反時計回りで次々変えていくと、点iを使う三角形を全て作れる.　そこで基準点iを固定したとき、以下のようなDP表を埋めることを考える

double　area[K][N][N];

第一添字は作った多角形の角数
第二添字はiと最初に繋ぐ点の番号
第三添字はiと最後に繋ぐ点の番号

例えば、ここまでの処理は固定したiについてarea[3][N][N]を埋めることに相当する.

ここから第一添字が4以上についてDP表を埋めていく.

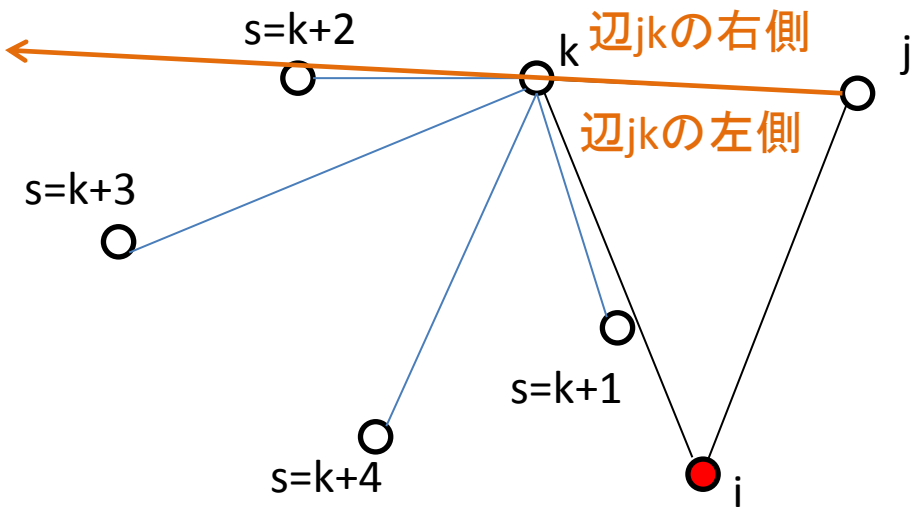
表の更新が終わったら現在のiを消去し、改めて一番下にある点のうち一番左にある点を新たな基準点iとして同じことを繰り返せばよい. 残りの点の数が2以下になったら終了. これで、各クエリに対して答える準備ができる.

問9　ハッピーエンド問題

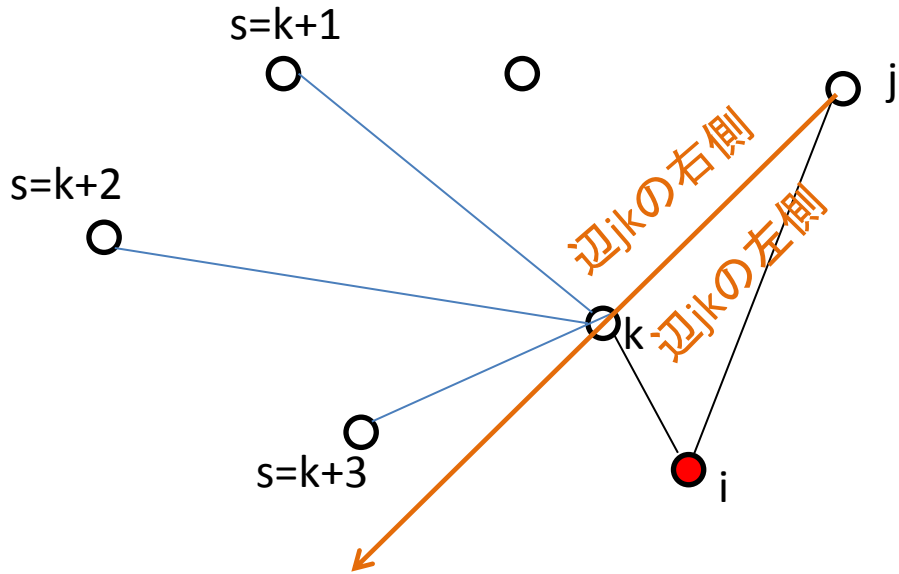
- 現時点までで area[3][N][N]は埋め終わっている．　これらの3角形について、新たに三角形を左端に付け加えたとき、凸多角形のままなら面積を計算し、DP表を更新する

➡ 辺ikの左側に新たな点sを使った三角形iksを付け加えたとき、凸4角形になっているかを、jとkとsに関してループを回してチェックしていく．

- 点sが線jkの左側 (三角形jksの符号付き面積が正) ならば凸4角形の条件を満たす．
- DP表は線jkの左側にある点sについて 、「area[4][j][s] = area[3][j][k] + 三角形iks」と更新される



- 次のkでは、点sが線jkの右側にあるものしかない．凸多角形が作れないので、DP表は更新されない



問９　ハッピーエンド問題

- 同様の処理を多角形の角数の最大値まで計算する.
- 各iについてO(KN^3)なので、合計O(KN^4)程度.
- 「各質問に対して面積最小の凸多角形は１つであり、２番目に小さい凸多角形との面積の差は 0.0001以上」という制限から、新たに見つかった面積の値が＜(小なり)の場合だけ、最小の面積とそれを構成する点インデックスを更新すればよい.
- 多角形インデックスは、面積のDP表の形に、点列を記録する分を拡張したint point[K][N][N][K]に記憶.
- 新たな最小面積多角形が見つかったら、それを構成するひとつ角が少ない多角形から配列をコピーした後、新たな点sの番号を付け加えればよい.