2021 年度 電気電子情報工学実験 II (b)
実践的・競技プログラミング 本番コンテスト（Contest 2-2）
解説

廣田悠輔
y-hirota@u-fukui.ac.jp

2021 年 6 月 3 日

目次

1 問題 A：ある数列

解説

問題 A は数式のとおり各項を順に計算すれば正答できる．いくつか注意すべき点を挙げる．

- メモ化などの工夫をしない単純な再帰関数により実装した場合，計算量は指数的に増大する．したがって，そのようなプログラムをコンテストシステムに提出した場合 `TIMELIMIT` と判定されてしまう．メモ化などにより重複計算を回避する再帰関数を実装するか，再帰を使わずに `for` 文などの繰り返し処理を実装すればこの問題は回避できる．
- $n = 0$ または $n = 1$ に対応できないために `WRONG-ANSWER` となった回答が多数見受けられた．問題の制約を満たす入力すべてに対応できるようにプログラムを作成されたい．
- Fibonacci 数列であれば数列の項が指数的に増大するために大きな項番の項を計算するにはオーバフローに気を付ける必要がある．一方，この問題の数列では第 i 項は第 $i-1$ 項に対して最大で $i-1$ しか大きくならないため，$A_i \leq 1 + i(i-1)/2$ $(i = 1, 2, \ldots)$ が保証される．したがって，$0 \leq n \leq 1000$ の範囲ではこの数列の項が 32 ビット符号付き整数で表現可能な最大値を超過する心配はない．

回答例

```c
#include <stdio.h>

enum {
  N_MAX = 1000 + 1
};

int main(void)
{
  int n, i;
  int a[N_MAX];

  scanf("%d", &n);

  a[0] = 0;
  a[1] = 1;
  for (i = 2; i <= n; ++i) {
    a[i] = a[i - 1] + a[i - 2] % i;
  }
  printf("%d\n", a[n]); // Not a[n - 1]!

  return 0;
}
```

2　問題 B：整数距離

解説

候補となる点は MN 個ある．制約 $2 \leq M \leq 100$，$2 \leq N \leq 100$ が与えられていることから，調べるべき点は最大でも 10^4 個である．したがって，MN 個の点すべてについて，二重ループにより，それぞれの点の原点からの距離が整数になるかを調べれば良い．

点 (p, q)（ただし p，q は自然数）の原点からの距離は $\sqrt{p^2 + q^2}$ である．したがって，点 (p, q) の原点からの距離が整数になるか否かは $\sqrt{p^2 + q^2}$ が整数か否か判定できれば良い．値 $\sqrt{p^2 + q^2}$ を数学関数 `sqrt` により計算すると，浮動小数点数計算の誤差により整数か否かの判定が難しくなる．この問題を回避するには，$\sqrt{p^2 + q^2}$ が整数値となるか判定するのではなく，$p^2 + q^2$ が平方数となるかを判定すれば良い．

回答例

```c
#include <stdio.h>

```

```
3  int is_integer_distance(int const x, int const y) {
4    int i;
5
6    for (i = 1; i * i <= x * x + y * y; ++i) {
7      if (i * i == x * x + y * y) {
8        return 1;
9      }
10   }
11
12   return 0;
13 }
14
15 int main(void) {
16   int m, n;
17   int i;
18   int x, y;
19   int count;
20
21   scanf("%d␣%d", &m, &n);
22
23   count = 0;
24   for (x = 1; x <= m; ++x) {
25     for (y = 1; y <= n; ++y) {
26       if (is_integer_distance(x, y)) {
27         count++;
28       }
29     }
30   }
31
32   printf("%d\n", count);
33
34   return 0;
35 }
```

3　問題 C：最大の円

解説

線分が n 本与えられるの条件を満たす最大半径は，図 1 のように線分が一本ずつ選んだけときの円の最大半径を調べ，その最小値をとることで求められる．一見して非常に複雑な問題に見えるが，実は一本の線分と

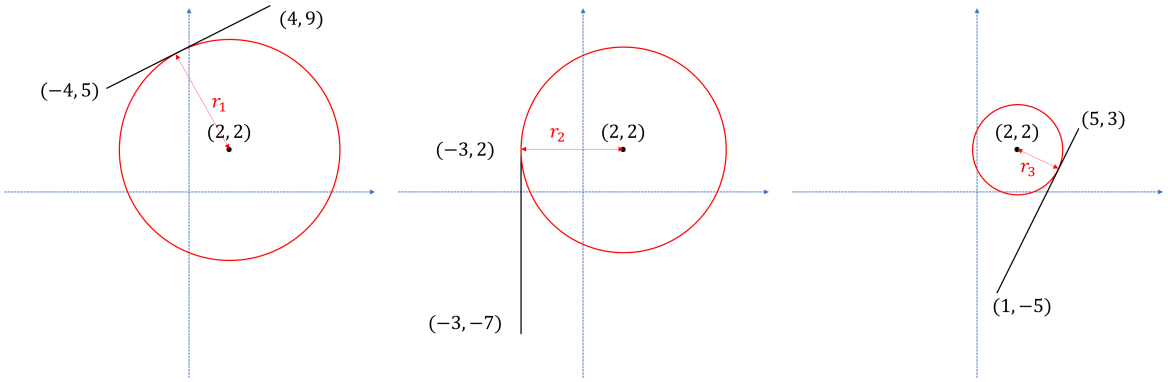

図 1: 問題文の入力例 1 で線分を一本ずつ選んだときの最大半径の円（赤）.

一つの円だけがある場合を考えれば良いのである.

線分が一本だけ存在するときの円の最大半径の調べ方について述べる. このときの円の最大半径は円の中心から線分までの距離に一致する. 円の中心から線分までの距離は, 円の中心と線分の座標の位置関係により 3 つに場合分けして考える.

- 円の中心が線分の端点を通り線分に垂直となる 2 本の直線の間になく, 線分の片側の端点（以下, P_1 と呼ぶ）により近い側にあるとき（図 2 の下側）, 円の中心と線分の距離は円の中心から P_1 の長さとなる.
- 円の中心が 2 本の直線の間になく, 線分の端点のうち P_1 でない点（以下, P_2 と呼ぶ）により近い側にあるとき（図 2 の上側）, 円の中心と線分の距離は円の中心から P_2 の長さとなる.
- 円の中心が 2 本の直線の間（図 2 の 2 本の緑破線の間）にある場合, 円の中心と線分の距離は円の中心から線分に向かう垂線の長さとなる.

あとは, それぞれの場合に分けて円の中心から線分の距離を求めれば良い（回答例 11–30 行目）. 具体的な場合分けの方法, 距離の求め方は初等幾何の内容であるので省略する. 回答例では線分の片側の端点から円の中心に向かうベクトルおよび線分の同じ端点からもう一方の端点に向かうベクトルの内積や外積を用いて計算している.

なお, 回答例に示すプログラムでは浮動小数点数を使用して計算を行っているが, 扱うべき座標の最大値が 100 程度であるのに対して出力すべき値は小数点第 2 位までであるので, 倍精度浮動小数点数（double 型）を用いれば精度は十分である.

ノート

- 本問題は比較的シンプルな問題なのでその必要はないが, 複雑な幾何問題ではすべてを自分で実装するのではなくライブラリを用いるのが容易である. C++ であれば `boost::geometry`（C++ STL ではないので注意）, Java であれば `java.awt.geom` パッケージなどが使用できる. 自分の目的に応じて適切なものを使用されたい.
- Yang Dixin さんは受講者の中でただ一人この問題に正答するという素晴らしい結果を残した.

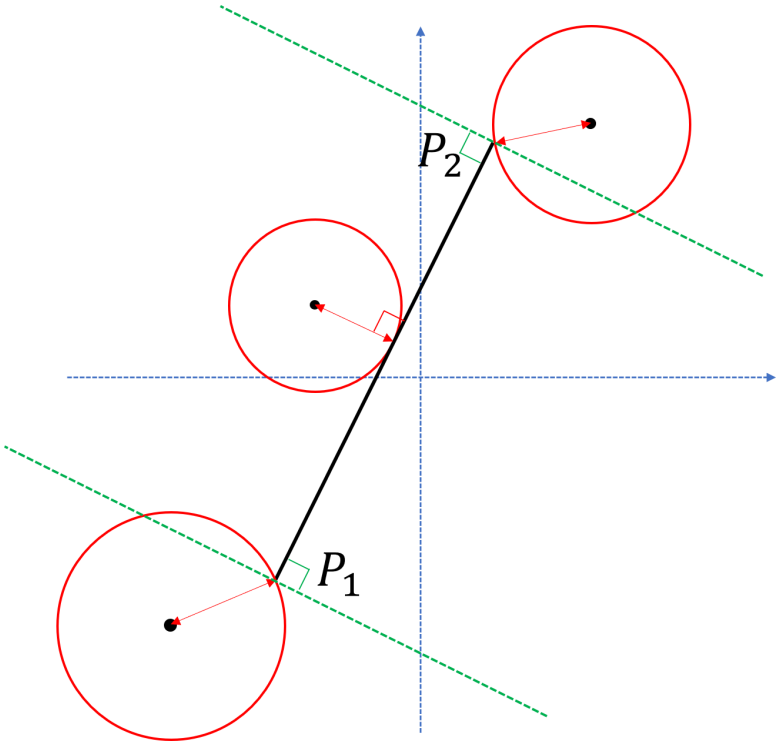

図 2: 線分（黒線分）と円の中心（黒点）の位置と最大円の半径（両側赤矢印付き線分の長さ）の関係．緑破線は線分の端点を通り線分に垂直な直線．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <float.h>
4  #include <math.h>
5
6  struct segment {
7    /* Line segment (x, y) -- (v, w) */
8    double x, y, v, w;
9  };
10
11 double segment_distance(struct segment const s, double const a, double
       const b)
12 {
13   double inner_prod1, inner_prod2, outer_prod;
14
15   /* If the nearest point is (x, y). */
```

```
16    inner_prod1 = (a - s.x) * (s.v - s.x) + (b - s.y) * (s.w - s.y);
17    if (inner_prod1 <= 0.0) {
18      return sqrt((s.x - a) * (s.x - a) + (s.y - b) * (s.y - b));
19    }
20
21    /* If the nearest point is (v, w). */
22    inner_prod2 = (a - s.v) * (s.x - s.v) + (b - s.w) * (s.y - s.w);
23    if (inner_prod2 <= 0.0) {
24      return sqrt((a - s.v) * (a - s.v) + (b - s.w) * (b - s.w));
25    }
26
27    /* Otherwise, the nearest point is on the line segment. */
28    outer_prod = (a - s.x) * (s.w - s.y) - (s.v - s.x) * (b - s.y);
29    return fabs(outer_prod) / sqrt((s.v - s.x) * (s.v - s.x) + (s.w - s.y
          ) * (s.w - s.y));
30  }
31
32  double solve(int const n, struct segment const * const s, double const
       a, double const b)
33  {
34    int i;
35    double min_distance = DBL_MAX;
36    for (i = 0; i < n; ++i) {
37      double const distance = segment_distance(s[i], a, b);
38      if (distance < min_distance) min_distance = distance;
39    }
40    return min_distance;
41  }
42
43  int main(void)
44  {
45    int n;
46    struct segment s[10];
47    double a, b;
48    int i;
49
50    scanf("%d", &n);
51    for (i = 0; i < n; ++i) {
52      scanf("%lf␣%lf␣%lf␣%lf", &s[i].x, &s[i].y, &s[i].v, &s[i].w);
53    }
```

```
54
55    scanf("%lf␣%lf", &a, &b);
56
57    printf("%.2f\n", solve(n, s, a, b));
58
59    return EXIT_SUCCESS;
60 }
```

4　問題 D：お小遣い最大化問題

解説

問題 D は動的計画法（DP: Dynamic Programming）を応用すれば容易に解ける．DP とは，ざっくり言えば以下の性質を満たす解法の総称である [1, 第 II 部 問題 D 3. (7)].

- 最小化問題（あるいは最大化問題）を対象とする．
- 問題を部分問題に分割して解く．
- 一度解いた部分問題の結果を記憶して，同じ部分問題の結果が必要なときに再利用する．
- 以上の方法で効率的に元の最小化問題（最大化問題）の解を求める．

この問題の解（貰えるお小遣いの最大値）をモデル化する．残り期間が n ヶ月で，その期間の最初の月に貰えるお小遣いの金額が p 円であるときに，貰えるお小遣い総額の最大金額を関数 $f(n,p)$ で表す．我々が求めなければならない金額は，残り期間が m ヶ月で最初の月に貰えるお小遣いが 1000 円のときのお小遣い総額の最大値 $f(m,1000)$ である．ところで，残り n ヶ月でのお小遣い総額の最大値 $f(n,p)$ は，残り $n-1$ ヶ月でのお小遣い総額の最大値を使って次のように再帰的に表現できる．

$$f(n,p) = \max\{p + f(n-1,q),\ f(n-1,1000)\}, \tag{1}$$

$$q = \max\{p-100, 0\}. \tag{2}$$

ただし，

$$f(0,p) = 0 \tag{3}$$

である．式 (1) の max 関数は，以下の 2 つの選択肢からより高額のお小遣い総額となる側の選択を意味する．

- その月のお小遣い p と，翌月以降 $n-1$ ヶ月間のお小遣い総額の最大値 $f(n-1,q)$ の合計を受け取る．ただし，q は (2) に表される金額であり，その月のお小遣いから 100 円を引いた金額，または 100 円引いた金額が 0 円未満となるときは 0 円である．
- その月のお小遣いを辞退し，翌月以降 $n-1$ ヶ月間のお小遣い総額の最大値 $f(n-1,p)$ を受け取る．

式 (1)，(2)，(3) にしたがって再帰的に $f(m,1000)$ を計算すれば答えが求まる．

再帰的に $f(m,1000)$ を求める計算には，同じ n, p を引数とする $f(n,p)$ を求める問題（部分問題）が何度も出現する．同じ部分問題が出現する度に解き直してしまうと，この問題の計算量は指数的に増大してしまう．そこで，パラメータ n, p の値ごとに一度求めた $f(n,p)$ の値を記憶し，再び同じ $f(n,p)$ が必要となった場合には記憶した値を呼び出すことで計算を省略する（メモ化）．

回答例では okodukai_dp という関数を使って再帰的な計算を実装している．残り期間が 0 ヶ月の基底部では (3) の通りゼロを返している（15–17 行目）．それ以外の場合は，(1)，(2) のとおり，その月のお小遣いを辞退する場合と貰う場合のお小遣い総額をそれぞれ再帰的に計算し，それらの中でより大きな金額を選択している（24–30 行目）．この実装では，部分問題の解の値を記憶する配列 memo を用意しておき（9 行目），一度解いた問題に対してはその値を再利用している（19–22 行目）．

コメント

- 動的計画法を応用できる問題の出題を予告していたにもかかわらず正答者は一名もいなかった．動的計画法は競技プログラミングに頻出であるだけでなく，実用においても重要な手法であるのでしっかり勉強し応用できるように身に付けて欲しい．
- 授業後にコンテストシステムへ本問題の回答を提出して CORRECT を得ているものが数名いた．しかしながら，その中には嘘回答と呼ばれる回答が見受けられた．嘘回答というのは，問題の要件を満たさない回答であるにもかかわらず，コンテストシステムのテストが甘いために誤答と判定されずに受理されてしまう（あるいは受理されることを期待した）回答である．競技プログラミングでは最後の手段として嘘回答を提出するのはコンテスト・テクニックとして認められるが，業務や研究で使うプログラムとしては要件を満足しない不適切なものであることに注意し，王道の回答ができるように努められたい．

回答例

```
1  #include <stdio.h>
2
3  enum {
4    OKODUKAI_UNIT = 100,
5    OKODUKAI_INITIAL = 10,
6    PERIOD_MAX = 100
7  };
8
9  int memo[PERIOD_MAX][OKODUKAI_INITIAL + 1];
10
11 int okodukai_dp(int const duration, int const current_okodukai)
12 {
13   int jitai, morau, next_okodukai;
14
15   if (duration == 0) {
16     return 0;
17   }
18
19   /* Solved sub-problem. */
20   if (memo[duration - 1][current_okodukai] >= 0) {
21     return memo[duration - 1][current_okodukai];
```

```
22    }
23
24    jitai = okodukai_dp(duration - 1, OKODUKAI_INITIAL);
25
26    next_okodukai = (current_okodukai == 0 ? 0 : current_okodukai - 1);
27    morau = current_okodukai + okodukai_dp(duration - 1, next_okodukai);
28
29    memo[duration - 1][current_okodukai] = (jitai > morau ? jitai : morau
          );
30    return memo[duration - 1][current_okodukai];
31  }
32
33  int main(void)
34  {
35    int i, j;
36    int result;
37    int m;
38    scanf("%d", &m);
39
40    for (i = 0; i < m; ++i) {
41      for (j = 0; j <= OKODUKAI_INITIAL; ++j) {
42        memo[i][j] = -1;
43      }
44    }
45
46    result = okodukai_dp(m, OKODUKAI_INITIAL);
47    printf("%d\n", result * OKODUKAI_UNIT);
48
49    return 0;
50  }
```

参考文献

[1] 筧捷彦，「目指せ！プログラミング世界一 — 大学対抗プログラミングコンテスト ICPC への挑戦」，近代科学社，2009.