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

廣田悠輔
y-hirota@u-fukui.ac.jp

目次

1 問題 A：合格者数

解説

受験者ごとの 3 科目合計点を求めて降順ソートし，同点の受験者に気をつけて合格者数を数えあげれば良い．回答例ではソートに `qsort` 関数を使用しているが，受験者数についての制約 $1 \leq N \leq 10^3$ があるのでバブルソートなどの計算量 $\mathrm{O}(N^2)$ のソートを使っても（おそらく）制限時間内に実行可能である．

コメント

入力を受け取ってソートするだけの基本的な問題であるので，この問題が解けなかった学生は真剣にプログラミング（およびデータ構造とアルゴリズム）を復習すること．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int descending_int(const void *a, const void *b) {
5    return *((int *)b) - *((int *)a);
6  }
7
8  int main(void) {
9    int n, k;
10   int *sum;
11   int i;
12   int ans;
13
```

```
  scanf("%d␣%d", &n, &k);
  sum = malloc(sizeof(int) * n);
  for (i = 0; i < n; ++i) {
    int m, p, e;
    scanf("%d␣%d␣%d", &m, &p, &e);
    sum[i] = m + p + e;
  }

  qsort(sum, n, sizeof(int), descending_int);

  ans = k;
  for (i = k; i < n; ++i) {
    if (sum[k - 1] == sum[i]) {
      ans++;
    } else {
      break;
    }
  }
  printf("%d\n", ans);

  free(sum);
  return EXIT_SUCCESS;
}
```

2 問題 B：防災計画

解説

入力を二次元配列などに読み込み，$M \times M$ のすべてのマスについて避難対象の世帯が存在するか調べれば良い．あるマスについて世帯が存在するか，火災発生地点から半径 N 以内であるかの判定は $O(1)$ で可能であるので，すべてのマスを調べるのに必要な計算量は $O(M^2)$ となる．地域のサイズ M は高々 100 であるので，この方法で制限時間に余裕をもって実行可能である．

コメント

入力が意図したとおりに読み込めない受講者が多く見受けられた．そのような受講者は C 言語（あるいはその他の自分の使用するプログラミング言語）の文字列処理に関して十分に学び直すこと．本問題では `fgets` と `scanf` さえ使えれば回答作成には苦労しないが，例えば `fgetc`, `sscanf` などはいろいろな状況で利用されるし，`string.h` をインクルードして使用する `strcmp`, `strcpy`, `strtok` などが使えると大変便利である．文字や文字列に関する C 言語の機能についての理解をより深めるよう努力されたい．

C 言語の関数の仕様がわからなくなったら，ウェブ検索をしても良いが，演習室の CentOS 上で `man fgets` のようにコマンドを打って調べる手もある．これは C 言語の関数だけでなく，UNIX/Linux コマンドの使い方がわからなくなったときにも有用である．例えば，`top` コマンドの使い方がわからなくなったのであれば `man top` とコマンド入力すれば解説が表示される．

回答例

```c
#include <stdio.h>
#include <stdlib.h>

int is_in(int const p, int const q, int const x, int const y, int const n)
{
  int const dpx = p - x;
  int const dqy = q - y;
  if (dpx * dpx + dqy * dqy <= n * n) {
    return 1;
  } else {
    return 0;
  }
}

#define BUFLEN (0x200)
#define MAX_STRING_LENGTH (100)

int main(void)
{
  int m, p, q, n;
  char buf[BUFLEN];
  char map2d[MAX_STRING_LENGTH][MAX_STRING_LENGTH + 3]; /* M-by-M char map
     , '\n' and '\0'.*/
  int i, j;
  int num_evacuation;

  fgets(buf, BUFLEN, stdin);
  sscanf(buf, "%d", &m);
  for (i = 0; i < m; ++i) {
    fgets(map2d[i], MAX_STRING_LENGTH + 3, stdin);
  }
  fgets(buf, BUFLEN, stdin);
  sscanf(buf, "%d␣%d␣%d", &p, &q, &n);

  /* Solve */
  num_evacuation = 0;
  for (i = 0; i < m; ++i) {
    for (j = 0; j < m; ++j) {
      if (map2d[i][j] == '#' && is_in(p, q , i + 1, j + 1, n)) {
        num_evacuation++;
      }
    }
  }

```

```
  printf("%d\n", num_evacuation);

  return EXIT_SUCCESS;
}
```

3 問題 C：出張費用最小化

解説

直接移動可能な都市間の移動費用を都市間の距離とみなして，最短経路問題として扱う．

最短経路問題を効率よく方法としてよく知られた Dijkstra 法によりある都市からある都市への最小移動費用を求めると，必要な計算量は $O(N^2)$ から $O((N+M)\log N)$ となる（実装法による変わる）．最小移動費用を求めるべき問合せは L 回あるので，そのたびに Dijkstra 法により最小移動費用を求めると，全体の計算量は $O(LN^2)$ から $O(L(N+M)\log N)$ となる．N, M, L はそれぞれ高々 $10^2, 10^4, 10^2$ であるので，この方法を使って制限時間内に回答可能である．

問合せ回数 L の値がもっと大きく（例えば $L = 10^5$ であるような）制限時間にそれほど余裕がないという状況を考えよう．このような場合には，すべての都市間の最小移動費用をあらかじめ何らかの方法で調べておき，L 回の問合せの際にはその結果を毎回参照するという方法が考えられる．最多経路問題のすべてのノード間の距離（この問題てはすべての都市間の最小移動費用）を調べる方法に Warshall-Floyd 法 [2, Sec. 2.5] がある．Warshall-Floyd 法を用いると，すべての都市間の最小移動費用が $O(N^3)$ の計算量で求まる．以後は問い合わせがあるたびに結果を参照する（$O(1)$ の操作）だけで最小移動費用を答えられる．したがって，全体の計算量は $O(N^3 + L)$ となり，L が大きな値である場合に計算量を抑えられる．

回答例（Warshall-Floyd 法）

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int distance(int **a, int const i, int const j)
{
  return (i > j) ? a[j][i] : a[i][j];
}

/* Find (i, j) shortest distances for all 0 <= i, j < n. */
void warshall_floyd(int const n, int **a)
{
  int i, j, k;

  for (k = 0; k < n; ++k) {
    /* Update */
    for (i = 0; i < n; ++i) {
      for (j = i + 1; j < n; ++j) {
        int const candidate = distance(a, i, k) + distance(a, k, j);
        if (a[i][j] > candidate) {
          a[i][j] = candidate;
```

```
22          }
23        }
24      }
25    }
26  }
27
28  int main(void)
29  {
30    int n, m, l;
31    int **a;
32    int i, j, k;
33
34    scanf("%d", &n);
35    scanf("%d", &m);
36    scanf("%d", &l);
37
38    a = malloc(sizeof(int *) * n);
39    for (i = 0; i < n; ++i) {
40      a[i] = malloc(sizeof(int) * n);
41    }
42
43    for (i = 0; i < n; ++i) {
44      for (j = 0; j < i; ++j) {
45        // Lower part elements are not used.
46        a[i][j] = -1;
47      }
48    }
49    for (i = 0; i < n; ++i) {
50      // Self
51      a[i][i] = 0;
52    }
53    for (i = 0; i < n; ++i) {
54      for (j = i + 1; j < n; ++j) {
55        // Default values (no direct path between i and j).
56        a[i][j] = INT_MAX / 4; // Divided by 4 to avoid overflow.
57      }
58    }
59
60    // Problem setting.
61    for (k = 0; k < m; ++k) {
62      int ii, jj, d;
63      scanf("%d␣%d␣%d", &ii, &jj, &d);
64      a[ii][jj] = d;
65    }
```

```
66
67    // Solve
68    warshall_floyd(n, a);
69
70    // Query
71    for (int k = 0; k < l; ++k) {
72      int ii, jj;
73      scanf("%d␣%d", &ii, &jj);
74      printf("%d\n", distance(a, ii, jj));
75    }
76
77    for (i = 0; i < n; ++i) {
78      free(a[i]);
79    }
80    free(a);
81    return EXIT_SUCCESS;
82  }
```

4 問題 D：Keep Distance

解説

本問題は POJ 2456 “Agressive Cow”[1][*1]を参考に作成された類題である．

本問題を解く前に，まず「全員の距離を L 以上離して M 人全員を着席させられるか」というより簡単な問題を考える．例えば，

```
**#**#********#**********#*#*
```

という配置の椅子に対して，6 以上の距離を開けて 4 人が着席可能かとった具合である．この可否は以下の手順により判断可能である．

1. 最初の椅子に 1 名を着席させる．
2. 現在着席している最も右側の椅子よりも右にあり，現在着席している最も右側の椅子から L 以上離れたもっとも左側にある椅子に 1 名を着席させる．
3. 全員が着席するか，着席可能な椅子がなくなるまで手順 2 を繰り返す．
4. 全員が着席出来たら可能であり，着席可能な椅子がなくなったら不可能であると判断できる．

例えば，前述の例であれば，以下の `@` の位置に 3 人が着席させられるが 4 人目が着席可能な椅子が存在せず，そのような配置が不可能だとわかる．

```
**@**#********@***#***@*#*
```

この手順は椅子の位置（座標）をあらかじめ配列に格納しておけば $O(N)$ の計算量で実行可能である．

先述の簡単化された問題が解ければ，元の問題も容易に解ける．簡単化された問題「全員の距離を L 以上離して M 人全員を着席させられるか」の距離を $L = 1, 2, ...$ を順に増大させて解き，全員着席可能な最大の L が求めるべき答えとなる．文字列の長さは $|S|$ であるので，調べるべき距離 L の最大値は $|S| - 1$ である．したがって，この方法で問題を解いたときの計算量は $O(N|S|)$ となる．文字列の長さは $|S| \leq 1000$，椅子の数は $N \leq 1000$ という制約があるので，時間内に回答可能である．

[*1] 日本語訳が [2, Sec. 3.1] に掲載されている．

参考までにより難しいケースとして，例えば文字列の長さが $|S| = 10^6$ のように大きな値で，前述の方法では実行時間に余裕がない場合を考える．前述の方法では，L の値を $1, 2, ...$ と順に増大させ，逐一その L の値で着席が可能か否かを判定していた．しかしながら，この部分は二分探索を使えば探索回数（判定回数）を $O(\log |S|)$ 回に削減できる．例えば，$L = 1$ で着席可能，$L = |S| - 1$ で着席不能であったとする．このとき，着席可能な L の最大値は $1 \leq L \leq |S| - 1$ の範囲に存在する．次にその中間値である $L = |S|/2$ で着席可能であるかを調べ，着席可能ならば着席可能な L の最大値は $|S|/2 \leq L \leq |S| - 1$ の範囲に存在し，着席不可能ならば着席可能な L の最大値は $1 \leq L \leq |S|/2$ の範囲に存在するとわかる．今度は新しい範囲の中間値を L として着席可能であるかを調べ，同様に最大値の存在範囲を半分に絞り込む．これを $\log_2 (|S| - 1)$ 繰り返せば着席可能な最大の L に辿りづける．結果として，この方法で問題を解いたときの計算量は $O(N \log |S|)$ となる．

回答例（二分探索）

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int num_available_seat(int const n, int const k, int const * const pos)
{
  int i;
  int num_available = 1; /* Number of available seats */
  int last_pos = pos[0];
  for (i = 1; i < n; ++i) {
    if (pos[i] - last_pos >= k) {
      last_pos = pos[i];
      num_available++;
    }
  }
  return num_available;
}

int find_max_interval(int const n, int const m, int const * const pos)
{
  /* Binary search search. */
  int left, right, mid;

  left = 1; /* Feasible interval */
  right = pos[n - 1] - pos[0] + 1; /* Infeasible interval */
  for (mid = (left + right) / 2; mid != left; mid = (left + right) / 2) {
    if (num_available_seat(n, mid, pos) >= m) {
      /* Interval mid is feasible. */
      left = mid;
    } else {
      /* Interval mid is infeasible. */
      right = mid;
    }
```

```
  }
  return mid;
}

#define MAX_STRING_LENGTH (1000)

int main(void)
{
  int m, n;
  char s[MAX_STRING_LENGTH + 2]; /* The main string, '\n', and '\0' */
  int pos[MAX_STRING_LENGTH]; /* Only first n elements are used. */
  int i;
  int result;

  fgets(s, MAX_STRING_LENGTH + 2, stdin);
  scanf("%d", &m);

  n = 0;
  for (i = 0; s[i] != '\0'; ++i) {
    if (s[i] == '#') {
      /* n-th chair is placed on the position i. */
      pos[n] = i;
      n++;
    }
  }

  /* Solve */
  result = find_max_interval(n, m, pos);

  printf("%d\n", result);

  return EXIT_SUCCESS;
}
```

参考文献

[1] PKU JudgeOnline, "Aggressive cows", http://poj.org/problem?id=2456 (accessed on 2022-05-22).

[2] 秋葉拓哉，岩田陽一，北川宜稔，「プログラミングコンテストチャレンジブック: 問題解決のアルゴリズム活用力とコーディングテクニックを鍛える」，第 2 版，マイナビ出版，2012.