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 3-2）
解説

廣田悠輔
y-hirota@u-fukui.ac.jp

2021 年 6 月 24 日

目次

1 問題 A：合同実験

解説

問題文のとおりに 2 人組を作り，その 2 人組の数を足し合わせれば回答可能である．しかしながら，**もっと簡単に回答することもできる**．出力すべき値は作られる 2 人組の総数であるので，学生数の内訳（M および N）に関係なく 2 人組の数は学生総数 $M + N$ を 2 で割った値を切り上げた整数となる．

競技プログラミングではこの問題のように回答に必要のない情報が問題文で説明されることがある．回答の際はよく考え，本当に必要な情報を使ってプログラムを作成するように努められたい．現実のプログラム設計においては設計のために必要最低限の情報だけが与えられるというようなことはなく，むしろ不要な情報が数多くある中から本質的に必要な部分を取り出して用いることが多いのである．

回答例

```
1 #include <stdio.h>
2
3 int main(void) {
4   int m, n;
5   scanf("%d", &m);
```

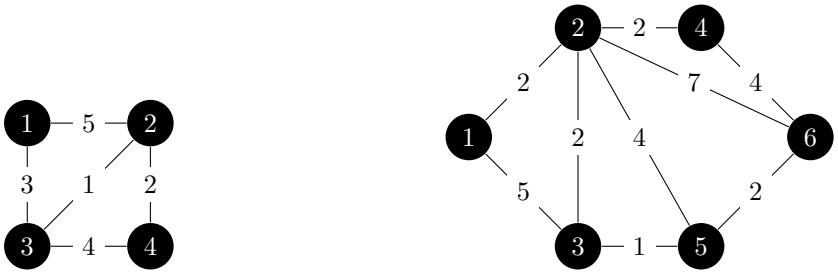


図 1: 入力例のグラフによる表現.

```
6    scanf("%d", &n);
7
8    printf("%d\n", (m + n + 1) / 2);
9
10   return 0;
11 }
```

2　問題 B：わらしべ長者

解説

問題文では物々交換について説明しているが，実のところは単なる最短経路問題である．最短経路問題の地点が物質であり，地点間の距離が交換コストに相当する．例えば，入力例 1，2 は図 1 のようにグラフで表現できる．最短経路問題は Dijkstra 法などを使えば効率的に解ける．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  #include <assert.h>
6
7  enum {
8    MAX_N = 100,
9  };
10
11 int edge[MAX_N][MAX_N]; // Initialized by zeros.
12
13 int dijkstra(int const n, int const s, int const e)
```

```
14 {
15   int i;
16   int fixed[n]; // If the shortest distance from the node s has been
         computed, 1.
17   int nodes[n]; // Distance from the node s.
18   int pivot; // Currently checking node.
19
20   for (i = 0; i < n; ++i) {
21     fixed[i] = 0;
22   }
23   for (i = 0; i < n; ++i) {
24     nodes[i] = INT_MAX;
25   }
26
27   pivot = s;
28   nodes[s] = 0;
29   fixed[s] = 1;
30   while (pivot != e) {
31     int next_pivot;
32     int next_pivot_distance;
33
34     // Update nodes that connects to the pivot node.
35     for (i = 0; i < n; ++i) {
36       if (edge[pivot][i] <= 0) continue; // No route between the nodes
             pivot and i.
37       if (!fixed[i] && nodes[i] > nodes[pivot] + edge[pivot][i]) {
38         nodes[i] = nodes[pivot] + edge[pivot][i];
39       }
40     }
41
42     // Find next pivot node (non-fixed smallest node).
43     next_pivot = -1;
44     next_pivot_distance = INT_MAX;
45     for (i = 0; i < n; ++i) {
46       if (!fixed[i] && nodes[i] <= next_pivot_distance) {
47         next_pivot = i;
48         next_pivot_distance = nodes[i];
49       }
50     }
51     assert(next_pivot >= 0);
```

```
52      pivot = next_pivot;
53      fixed[next_pivot] = 1;
54    }
55
56    return nodes[e];
57  }
58
59  int main(void)
60  {
61    int n, s, e;
62    int p, k;
63
64    scanf("%d", &n);
65    scanf("%d␣%d", &s, &e);
66    s--; // 0-origin indexing.
67    e--; // 0-origin indexing.
68    scanf("%d", &k);
69    for (p = 0; p < k; ++p) {
70      int i, j, c;
71      scanf("%d␣%d␣%d", &i, &j, &c);
72      i--; // 0-origin indexing.
73      j--; // 0-origin indexing.
74      edge[i][j] = c;
75      edge[j][i] = c;
76    }
77
78    printf("%d\n", dijkstra(n, s, e));
79
80    return EXIT_SUCCESS;
81  }
```

3　問題 C：正整数の立方数による表現

解説

まず愚直な方法を考えてみよう．n を正整数の立方数の和として

$$n = p_1^3 + p_2^3 + \cdots + p_r \quad (r \leq m)$$

の形で表現するのであるから

$$1 \leq p_i \leq q \quad (1 \leq i \leq r), \qquad q = \lfloor \sqrt[3]{n} \rfloor$$

でなければならない．ただし，$\lfloor a \rfloor$ は a 以下の最大の整数（a の床関数値）である．したがって n を r 個の正整数の立方数の和として表現する $p_1, p_2, \ldots, p_r$ のパタンは q^r 通りある．したがって，$r = 1, 2, \ldots, m$ として，それぞれ q^r 個のパタンを全探索して問題に回答できる．

もう少し解法を単純化ならびに効率化することを考えよう．まず，n を m 個以下の正整数の立方数の和として表現するのではなく，ちょうど m 個の非負整数の立方数と和として表現することで解法を単純化する．例えば，$n = 30$（$q = 3$），$m = 5$ のとき，n は

$$30 = 3^3 + 1^3 + 1^3 + 1^3$$

と 4 個の正整数の立方数として表現できるが，これを $m = 5$ 個の非負整数の立方数の和

$$30 = 3^3 + 1^3 + 1^3 + 1^3 + 0^3 \tag{1}$$

と考えるようにする．すると常に m 個の変数を用いる場合を考えることになり，各変数の取りうる値の候補が $q + 1$ 個あるので $(q + 1)^m$ ケースを調べるというより単純な考え方ができる．

続いて，立方数 $p_1^3, p_2^3, \ldots, p_m^3$ の並び順が回答に関係しないことに注目する．例えば，(1) の右辺の並び順を

$$30 = 0^3 + 3^3 + 1^3 + 1^3 + 1^3$$

や

$$30 = 0^3 + 1^3 + 1^3 + 1^3 + 3^3$$

と変えても n を表現する正整数の立方数の数は 4 で変わりない．したがって，

$$p_i \geq p_j \qquad (1 \leq i < j \leq n)$$

という制約を加え，並び順だけが違う組合せを同一視して差し支えない．すると調べるべき組合せの数は

$$\binom{(q+1) + m - 1}{m}$$

まで削減できる．あとは，この組合せを再帰関数などにより図 2 のように全探索すれば良さそうに思える．しかしながら，本問題では最大で $n = 5 \times 10^4$，$m = 10$ であるから，調べるべき組合せ数は 2.92×10^{10} であり，制限時間である 2.5 秒以内に探索を完了するのはやや困難である．

制限時間内での探索を可能にするため，分枝限定法により探索対象の組み合わせを削減することを考える．再び，$n = 30$（$q = 3$），$m = 5$ の例を考える．仮に $p_1 = 3$ とした場合，

$$30 = 3^3 + p_2^3 + p_3^3 + p_4^3 + p_5^3 \tag{2}$$

を満たす p_2, p_3, p_4, p_5 を探索することになる．ここで $p_2 = 3$ とすると，

$$30 < 3^3 + 3^3$$

であるので，p_3, p_4, p_5 の値によらず (2) の等号が成り立たないと分かる．同様に，$p_2 = 2$ とした場合も (2) の等号は成立しない．したがって，$p_1 = 3, p_2 = 3$ および $p_1 = 3, p_2 = 2$ を満たす組合せは探索の必要がないことがわかる．このように確実に条件を満たさない場合を探索しないことで実際に探索する組合せの数を大きく削減できる（分枝限定法）．図 3 に分枝限定法を用いた場合の探索対象を示す．

分枝限定法では木のできるだけ上層で枝切りが生じるように探索を設計することが重要なポイントである．本解説では $p_i \geq p_j$ $(1 \leq i < j \leq n)$ という制約をつけて $p_1, p_2, \ldots$ の順に，すなわち大きな値の側から探索を行っている．この探索方法では立方数の部分和が早期に n より大きな値となるため，早い段階で部分木の枝切りが可能となる．逆に小さい値の側から順に探索すると木の下層に到達するまで n を超えないケースが多く，枝切りの効率が低下してしまう．

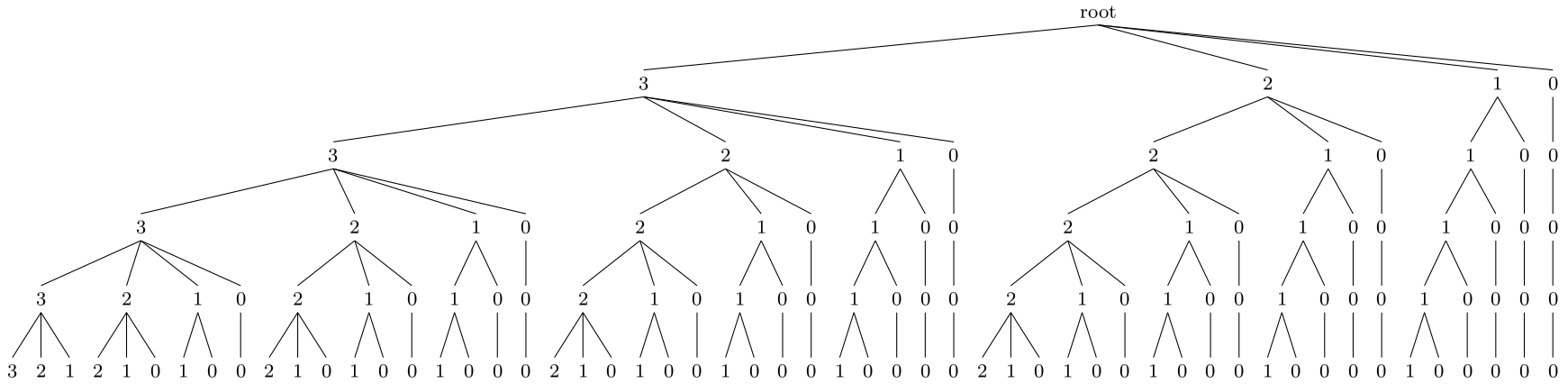


図 2: $n = 30$（$q = 3$），$m = 5$ の例．$p_i \geq p_j$ $(1 \leq i < j \leq n)$ を満たすケースの探索．root を除く各頂点は上の階層から順に $p_1, p_2, \ldots, p_m$ を表す．

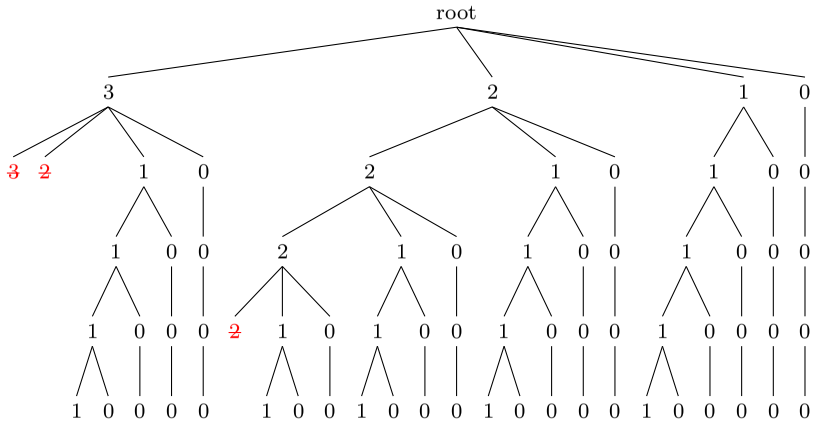


図 3: $n = 30$（$q = 3$），$m = 5$ の例に対して分枝限定法を用いた場合の様子．打消し線付き赤数字はその値を選択した場合に合計が n を超えることを意味する．打消し線付き赤数字以下の部分木を探索しないことで探索数を大幅に削減している．

ノート

類似問題が ACM ICPC 1999 年度京都大会に出題されている [2].

回答例

```
1  #include <stdio.h>
2  #include <assert.h>
3  #include <limits.h>
4
5
6  int dfs(int const n, int const m, int const sum, int const prev, int
       const depth)
7  {
8    int i;
9    int min_ans;
10
11   assert(m > 0);
12   assert(n > 0);
13   assert(sum >= 0);
14   assert(0 <= depth && depth <= m);
15
16   if (sum == n) {
17     return depth;
18   }
19   if (depth == m) {
20     return INT_MAX;
21   }
22
23   min_ans = INT_MAX;
24   for (i = prev; sum + i * i * i <= n; ++i) {
25     int const this_ans = dfs(n, m, sum + i * i * i, i, depth + 1);
26     if (this_ans < min_ans) {
27       min_ans = this_ans;
28     }
29   }
30   return min_ans;
31 }
32
33 int main()
```

```
34 {
35   int n, m;
36   int ans;
37
38   scanf("%d␣%d", &n, &m);
39   ans = dfs(n, m, 0, 1, 0);
40
41   if (ans <= m) {
42     printf("%d\n", ans);
43   } else {
44     printf("%d\n", -1);
45   }
46
47   return 0;
48 }
```

4 問題 D：電報料金最小化

解説

支社（本社を含む）を頂点，支社間の電報経路を辺（辺の重みは電報料金）とみなすと，支社間の電報経路情報は重み付き無向グラフとして表現できる．例えば入力例 2 は図 4 のように表現される．

この問題における全体連絡はグラフの全頂点が接続されるように辺を選択することに相当する．無論，そのような辺の選択方法は多数存在する．求めるべき値は最小電報料金であるので，そのような辺の選択の中から辺の重みの総和が最小となるもの（最小全域木，Minimum Spanning Tree）を求めることになる．例えば入力例 2 の電報料金を最小にする電報の送信方法

- 支社 0（本社）が支社 1 に電報を送信（料金 3），
- 支社 1 が支社 3 に電報を送信（料金 3），
- 支社 0（本社）が支社 4 に電報を送信（料金 2），
- 支社 4 が支社 2 に電報を送信（料金 2）

は，図 5 の最小全域木として表される[*1]．

図 4 のような非負の重み付きグラフが与えられるときに図 5 のような最小全域木を求めるアルゴリズムの一つに Prim 法がある．ここでは Prim 法の手順のみを説明する[*2]．Prim 法は手順は以下の通りである[*3]．

1. 1 つの頂点を選択して，その頂点のみからなる木 T を作る．

[*1] 入力例 2 に対しては偶然に一筆書き状の最小全域木が得られているが，最小全域木は一筆書き状である必要はない．

[*2] 詳細については [1, 第 2.5 節] などを参照のこと．特に貪欲的に頂点を追加していく操作でなぜ辺の重みの総和が最小値となる木が得られるかという点については感覚的には納得しがたい部分であろうと思うので，様々な文献を読んだり具体例を紙上で追跡するなどして理解されたい．

[*3] 見ての通りこの手順は最短経路を求める Dijkstra 法に似ている．

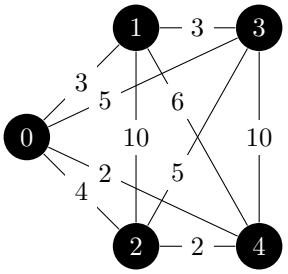


図 4: 入力例 2 のグラフによる表現．頂点が支社番号，辺の重みが支社間の電報料金を表す．

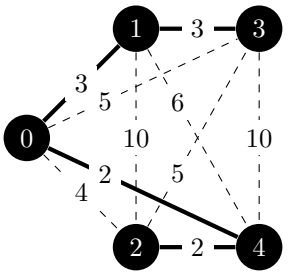


図 5: 入力例 2 のグラフに対する最小全域木．太い辺が選択された辺である．支社間の電報の送受信は太い辺による頂点間の接続に対応する．

2. 木 T の頂点と辺で接続され木 T に含まれない頂点のうち，接続する辺の重みが最小である頂点とその辺を木 T に加える．
3. すべての頂点が木 T に加わるまで手順 2. を繰り返す．

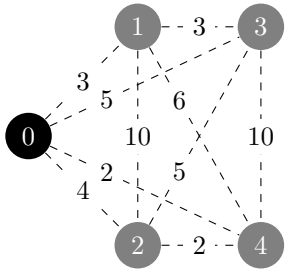
以上の手順を適用すれば最小全域木が得られるので，その辺の重みの総和が出力すべき値である．参考までに入力例 2 に Prim 法を適用したときの木 T の変化の様子を図 6 に示す．

ノート

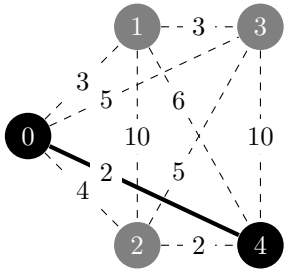
最小全域木を求める方法には Prim 法の他に Kruskal 法 [1, 第 2.5 節] も有名である．

回答例

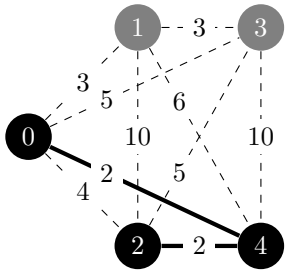
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4  #include <assert.h>
5
6  enum {
7    N_MAX = 20
8  };
9
10 int edges[N_MAX + 1][N_MAX + 1]; // i-to-j telegram fee.
```



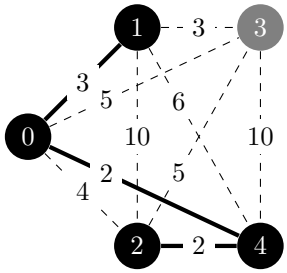
(a) 頂点 0 のみからなる木 T.



(b) 木 T に頂点 4 を追加.



(c) 木 T に頂点 2 を追加.



(d) 木 T に頂点 1 を追加.

図 6: 入力例 2 に対する Prim 法の手順．(d) の次に頂点 3 が頂点 1 に接続され図 5 の最小全域木が得られる．

```
11
12 // Prim's algorithm.
13 // Accept undirected graph.
14 // m: #nodes, s: starting node (arbitrary in [0, m).
15 int mst_prim_weight(int const m, int const s)
16 {
17   int i, k;
18   int queue_i[m * (m - 1) / 2]; // Queued edges (i-index).
19   int queue_j[m * (m - 1) / 2]; // Queued edges (j-index).
20   int queue_used[m * (m - 1) / 2];
21   int queue_end = -1;
22   int visited[m]; // Visited nodes.
23   int weight = 0L;
24
25   for (k = 0; k < m * (m - 1) / 2; ++k) {
26     queue_used[k] = 0;
27   }
28   for (i = 0; i < m; ++i) {
29     visited[i] = 0;
30   }
31
```

```
32    // Visit the node s and queue the edges from s.
33    visited[s] = 1;
34    for (i = 0; i < m; ++i) {
35      if (s == i) continue;
36      queue_end++;
37      queue_i[queue_end] = s;
38      queue_j[queue_end] = i;
39    }
40
41    while (1) {
42      int n_visited_nodes = 0;
43      int min_fee = INT_MAX;
44      int min_k = -9999;
45      int new_node_id = -9999;
46
47      // Check if all the nodes have been visited.
48      for (i = 0; i < m; ++i) {
49        if (visited[i] == 1) n_visited_nodes++;
50      }
51      if (n_visited_nodes == m) return weight; // Return the ans.
52
53      // Find the lightest edge to a node out of the current sub-graph.
54      for (k = 0; k <= queue_end; ++k) {
55        int const ii = queue_i[k];
56        int const jj = queue_j[k];
57
58        // If the edge has been used.
59        if (queue_used[k] == 1) continue;
60
61        // If the edge connects between the visited nodes, skip it.
62        if (visited[ii] == 1 && visited[jj] == 1) continue;
63
64        if (min_fee > edges[ii][jj]) {
65          min_fee = edges[ii][jj];
66          min_k = k;
67        }
68      }
69      assert(min_fee != INT_MAX);
70      queue_used[min_k] = 1;
71
```

```
72      weight += edges[queue_i[min_k]][queue_j[min_k]];
73      if (visited[queue_i[min_k]] == 0) {
74        new_node_id = queue_i[min_k];
75      } else {
76        new_node_id = queue_j[min_k];
77      }
78      visited[new_node_id] = 1;
79      // Update the queue.
80      for (i = 0; i < m; ++i) {
81        if (visited[i] == 1) continue; // The edge is to a visited node.
82        queue_end++;
83        queue_i[queue_end] = new_node_id;
84        queue_j[queue_end] = i;
85      }
86    }
87
88    return -2; // Cannot happen.
89  }
90
91  int main(void)
92  {
93    int n;
94    int i, j;
95
96    scanf("%d", &n);
97
98    for (i = 0; i <= n; ++i) {
99      for (j = i + 1; j <= n; ++j) {
100       scanf("%d", &edges[i][j]);
101       edges[j][i] = edges[i][j];
102     }
103   }
104
105   printf("%d\n", mst_prim_weight(n + 1, 0));
106
107   return EXIT_SUCCESS;
108 }
```

参考文献

[1] 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.

[2] ACM International Collegiate Programming Contest Asia RegionalContest, Problem B “Square Coins”, https://icpc.iisf.or.jp/past-icpc/regional1999/1999regional.pdf (Accessed, 2021-06-04).