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

目次

1 問題 A：電報字数削減

解説

やさしい文字列処理の問題である．目的の出力を得るには様々な方法があるが，回答例では以下の手順を採っている．

1. 出力の 1 文字目は必ず入力の 1 文字目と一致するので，入力の 1 文字目をそのまま出力する（回答例 17 行目）．また，入力文字列中の読み取った最新の文字が何回連続で出現しているかを記憶する変数 `n_repeat` に 1 をセットする（回答例 16 行目）．
2. 入力 2 文字目以降は，現在注目している文字（$i+1$ 文字目）が一つ前の文字（i 文字目）と一致するか否かにより分岐する（回答例 21 行目）．
   - 現在注目している文字が一つ前の文字と異なっているならば，一つ前の文字まで繰り返されていた回数（`n_repeat`）の値を十進表現で出力する．ただし，繰り返しの回数が 1 であれば回数は出力しない（回答例 23–25 行目）．また，新たな文字が出現したらその文字は必ず出力する必要があるので，現在注目している文字を出力する．また，繰り返し回数を 1 にリセットする（回答例 26–27 行目）．
   - 現在注目している文字が一つ前の文字と同じであれば，繰り返しの途中であるので何も出力をせず，繰り返しの回数を一つ増やす（回答例 29 行目）．
3. 入力文字列の終端に到達したら，現在の繰り返し回数の値を十進表現で出力する．ただし，繰り返しの

回数が 1 であれば，やはり回数は出力しない（回答例 34–36 行目）．

なお，入力は 1 行目が整数値，2 行目が大文字アルファベットのみからなる文字列であるので，（回答例は `fgets` と `sscanf` を使っているが）読み取りは `scanf` のみを使って行っても良い．

コメント

このような符号化手法をランレングス符号化（run-length encoding）という．

この問題は情報工学分野の大学 3 年次生を対象とした問題としてはやさしい．時間内に解けなかった者は基本的なプログラミング能力を身につけるられるよう努力すること．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_LEN (1000 + 2)
5
6  int main(void)
7  {
8    char str[MAX_LEN];
9    int n, n_repeat, i;
10
11   fgets(str, MAX_LEN, stdin);
12   sscanf(str, "%d", &n);
13   fgets(str, MAX_LEN, stdin);
14
15   /* First character */
16   n_repeat = 1;
17   printf("%c", str[0]);
18
19   /* Second to last */
20   for (i = 1; i < n; ++i) {
21     if (str[i - 1] != str[i]) {
22       /* New character found. Flush. */
23       if (n_repeat != 1) {
24         printf("%d", n_repeat);
25       }
26       n_repeat = 1;
27       printf("%c", str[i]);
28     } else {
29       n_repeat++;
```

```
30      }
31    }
32
33    /* Flush with a line break. */
34    if (n_repeat != 1) {
35      printf("%d", n_repeat);
36    }
37    printf("\n");
38
39    return EXIT_SUCCESS;
40  }
```

2　問題 B：電報復元

解説

ランレンスグ符号化の結果から，元の文字列を復号せよという問題である．入力文字列内に部分文字列として現れる数値を `int` 型等の整数値に変換する方法の設計が回答のポイントとなる．

復号の方法は唯一ではないが，回答例では次に示す方法を採用している．

- 入力を先頭から 1 文字ずつ順に読み取り，何文字繰り返すか確定した時点でまとめて出力する．例えば `A234BC5DE6` という文字列であれば，`A` を 234 回繰り返すことが確定した時点で `AAA...A`（`A` の 234 回繰り返し）を出力し，その後に読み取りを再開して `B` を 1 回だけ繰り返すことが確定した時点で `B` と出力，再び読み取りを再開して `C` を 5 回繰り返すことが確定した時点で `CCCCC` と出力，... という具合に，文字の繰り返しごとにまとめて出力を行う．
- 繰り返し回数確定の判定は，一つ前の文字（`str[i - 1]`）と現在注目している文字（`str[i]`）がそれぞれ，数字であるかアルファベットであるにより，4 パタンに分けて実施している（回答例 23–42 行目）．
  - 一つ前の文字と現在注目している文字がともにアルファベットである場合，一つ前の文字の繰り返し回数は 1 回であることに確定するので，一つ前の文字を出力する．
  - 一つ前の文字がアルファベットで現在注目している文字が数字である場合，現在注目している文字は繰り返し回数を表す数値の 1 桁目である．例えば入力が `A234BC5DE6`，一つ前の文字が `A` で現在注目している文字が 2 のとき，`A` というアルファベットの繰り返し回数を表す数値の 1 桁目 2 が新たに現れている．このときは，まだ文字の繰り返し回数は確定していないので，一つ前の文字を文字として，現在注目している文字の示す数字を数値に変換して記憶する．
  - 一つ前の文字が数字で現在注目している文字がアルファベットである場合，これは繰り返し回数を表す数値が一つ前の文字で終了し，現在の文字は新しいアルファベットであると判定できる．例えば入力が `A234BC5DE6` のとき，一つ前の文字が 4 で現在注目している文字が `B` のとき，繰り返し回数 234 の最後の桁が一つ前の文字 4 であり，現在注目している文字が `B` は新たなアルファベットである．このときに繰り返しの回数が確定するので，求めた繰り返し回数だけ文字を出力する．
  - 一つ前の文字と現在注目している文字がともに数字である場合，これは繰り返し回数を表す数値の新しい桁が出現したと判定できる．このとき，繰り返し回数は，既に判明している上位桁を 10 倍

して新しい桁の数値を足しこむことで更新する（回答例 40 行目）．
例えば入力が A234BC5DE6，一つ前の文字が 3 で現在注目している文字が 4 のときを考える．このとき，A23 までで A の上位桁が 23 であることが既に判明していて，さらに新たな桁に 4 という数字が加わることになるので，$234 = 23 \times 10 + 4$ という計算で A の繰り返し回数を更新できる．

- 入力文字列の終端では専用の処理を行う必要がある（回答例 45–53 行目）．最終文字がアルファベットであるか数字であるかにより異なるが，基本的な考え方は文字列の途中の処理と同じである．

なお，回答例では一文字ずつ読み取って数値を復元しているが，atoi を使って数値を復元することも可能である．

回答例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4
5  #define MAX_LEN (1000 + 2)
6
7  int main(void)
8  {
9    char str[MAX_LEN];
10   int m, n_repeat, i, j;
11   char repeating_char;
12
13   fgets(str, MAX_LEN, stdin);
14   sscanf(str, "%d", &m);
15   fgets(str, MAX_LEN, stdin);
16
17   /* First character */
18   repeating_char = str[0];
19   n_repeat = 1;
20
21   /* Second to last */
22   for (i = 1; i < m; ++i) {
23     if (isalpha((unsigned char) str[i - 1])) {
24       if (isalpha((unsigned char) str[i])) {
25         /* Previous char: alphabet, this char: alphabet. */
26         printf("%c", str[i - 1]);
27       } else {
28         /* Previous char: alphabet, this char: digit. */
29         repeating_char = str[i - 1];
```

```
30          n_repeat = str[i] - '0';
31        }
32      } else {
33        if (isalpha((unsigned char) str[i])) {
34          /* Previous char: digit, this char: alphabet. */
35          for (j = 0; j < n_repeat; ++j) {
36            printf("%c", repeating_char);
37          }
38        } else {
39          /* Previous char: digit, this char: digit. */
40          n_repeat = n_repeat * 10 + (str[i] - '0');
41        }
42      }
43    }
44
45    /* Flush with a line break. */
46    if (isalpha((unsigned char) str[m - 1])) {
47      printf("%c", str[m - 1]);
48    } else {
49      for (j = 0; j < n_repeat; ++j) {
50        printf("%c", repeating_char);
51      }
52    }
53    printf("\n");
54
55    return EXIT_SUCCESS;
56  }
```

3 問題 C：君は牛を一頭持っている

解説

この問題は動的計画法（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（得られる生乳量の最大値）をモデル化する．残り期間が n 日で，その期間の最初の日に得ら

れる生乳量が p リットルであるときに，得られる生乳総量の最大値を関数 $f(n,p)$ で表す．我々が求めなければならないのは残り期間が m 日で最初の日に得られる生乳量が 10 リットルのときの得られる生乳総量の最大値 $f(m,10)$ である．ところで，残り n 日で得られる生乳総量の最大値 $f(n,p)$ は，残り $n-1$ 日での生乳総量の最大値を使って次のように再帰的に表現できる．

$$f(n,p) = \max\{p + f(n-1,q),\ f(n-1,10)\}, \tag{1}$$

$$q = \max\{p-1, 0\}. \tag{2}$$

ただし，

$$f(0,r) = 0 \qquad (r = 0,1,\ldots,10) \tag{3}$$

である．式 (1) の max 関数は，以下の 2 つの選択肢からより多い生乳総量となる側の選択を意味する．

- その日の生乳生産量 p と，翌日以降 $n-1$ 日間の生乳総量の最大値 $f(n-1,q)$ の合計を得る．ただし，q は (2) で表される生乳量であり，その日の生乳量 1 リットルを引いた量，または 1 リットル引いた量が 0 リットル未満となるときは 0 リットルである．
- その日に牛を休ませ，翌日以降 $n-1$ 日間の生乳総量の最大値 $f(n-1,10)$ を得る．

式 (1)，(2)，(3) にしたがって再帰的に $f(m,10)$ を計算すれば答えが求まる．

再帰的に $f(m,10)$ を求める際には，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（メモ化という）．

回答例では `milk_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 日の基底部では (3) の通りゼロを返している（15–18 行目）．それ以外の場合は，(1)，(2) のとおり，その日に牛を休ませる場合と搾乳を行う場合の生乳総量をそれぞれ再帰的に計算し，それらの中でより多い側を選択している（25–31 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（20–23 行目）．

コメント

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．

回答例

```
1 #include <stdio.h>
2
3 enum {
4   MILK_UNIT = 1,
5   MILK_INITIAL = 10,
6   PERIOD_MAX = 100
7 };
8
```

```
 9 int memo[PERIOD_MAX][MILK_INITIAL + 1];
10
11 int milk_dp(int const duration, int const current_milk)
12 {
13   int yasumi, shiboru, next_milk;
14
15   if (duration == 0) {
16     /* Base case. */
17     return 0;
18   }
19
20   if (memo[duration - 1][current_milk] >= 0) {
21     /* This sub-problem is already solved. */
22     return memo[duration - 1][current_milk];
23   }
24
25   yasumi = milk_dp(duration - 1, MILK_INITIAL);
26
27   next_milk = (current_milk == 0 ? 0 : current_milk - 1);
28   shiboru = current_milk + milk_dp(duration - 1, next_milk);
29
30   memo[duration - 1][current_milk] = (yasumi > shiboru ? yasumi :
       shiboru);
31   return memo[duration - 1][current_milk];
32 }
33
34 int main(void)
35 {
36   int i, j;
37   int result;
38   int m;
39   scanf("%d", &m);
40
41   for (i = 0; i < m; ++i) {
42     for (j = 0; j <= MILK_INITIAL; ++j) {
43       memo[i][j] = -1;
44     }
45   }
46
47   result = milk_dp(m, MILK_INITIAL);
```

```
48    printf("%d\n", result * MILK_UNIT);
49
50    return 0;
51  }
```

4　問題 D：ファイル送信料金最小化

解説

学生を頂点，学生間の通信経路を辺（辺の重みは通信料金）とみなすと，学生間の通信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 1 のように表現される．

この問題における全学生間のファイル共有のための通信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通信料金であるので，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 の通信料金を最小にするファイルの送信方法

- 学生 0 が学生 1 にファイルを送信（料金 3），
- 学生 1 が学生 3 にファイルを送信（料金 3），
- 学生 0 が学生 4 にファイルを送信（料金 2），
- 学生 4 が学生 2 にファイルを送信（料金 2）

は，図 2 の最小全域木として表される[*1]．

図 1 のような非負の重み付きグラフが与えられるときに図 2 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．
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 3 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 [3, 第 2.5 節] も有名である．

回答例

[*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 [3, 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．

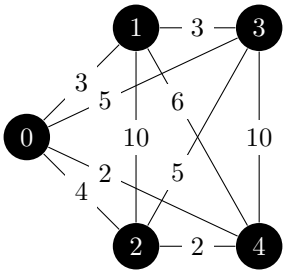


図 1: 入力例 2 のグラフによる表現．頂点が学生の番号，辺の重みが学生間の通信料金を表す．

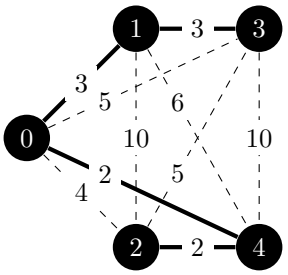
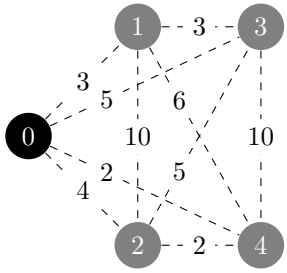
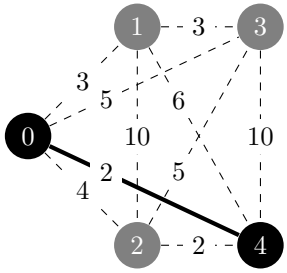


図 2: 入力例 2 のグラフに対する最小全域木．太い辺が選択された辺である．学生間のファイルの通信は太い辺による頂点間の接続に対応する．

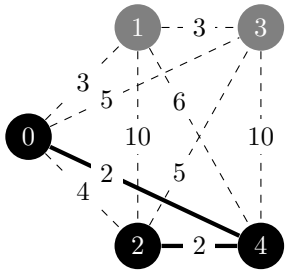
```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <assert.h>

enum {
  N_MAX = 20
};

int edges[N_MAX + 1][N_MAX + 1]; // i-to-j telegram fee.

// Prim's algorithm.
// Accept undirected graph.
// m: #nodes, s: starting node (arbitrary in [0, m)).
int mst_prim_weight(int const m, int const s)
{
  int i, k;
  int queue_i[m * (m - 1) / 2]; // Queued edges (i-index).
  int queue_j[m * (m - 1) / 2]; // Queued edges (j-index).
  int queue_used[m * (m - 1) / 2];
  int queue_end = -1;
```



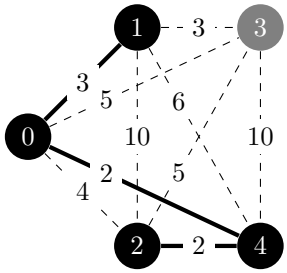
(a) 頂点 0 のみからなる木 T.



(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 3: 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 2 の最小全域木が得られる．

```
22  int visited[m]; // Visited nodes.
23  for (k = 0; k < m * (m - 1) / 2; ++k) {
24    queue_used[k] = 0;
25  }
26  for (i = 0; i < m; ++i) {
27    visited[i] = 0;
28  }
29
30  // Visit the node s and queue the edges from s.
31  visited[s] = 1;
32  for (i = 0; i < m; ++i) {
33    if (s == i) continue;
34    queue_end++;
35    queue_i[queue_end] = s;
36    queue_j[queue_end] = i;
37  }
38
39  int weight = 0L;
40  while (1) {
41    int n_visited_nodes = 0;
42    int min_fee = INT_MAX;
```

```
43      int min_k = -9999;
44      int new_node_id = -9999;
45
46      // Check if all the nodes have been visited.
47      for (i = 0; i < m; ++i) {
48        if (visited[i] == 1) n_visited_nodes++;
49      }
50      if (n_visited_nodes == m) return weight; // Return the ans.
51
52      // Find the lightest edge to a node out of the current sub-graph.
53      for (k = 0; k <= queue_end; ++k) {
54        int const ii = queue_i[k];
55        int const jj = queue_j[k];
56
57        // If the edge has been used.
58        if (queue_used[k] == 1) continue;
59
60        // If the edge connects between the visited nodes, skip it.
61        if (visited[ii] == 1 && visited[jj] == 1) continue;
62
63        if (min_fee > edges[ii][jj]) {
64          min_fee = edges[ii][jj];
65          min_k = k;
66        }
67      }
68      assert(min_fee != INT_MAX);
69      queue_used[min_k] = 1;
70
71      weight += edges[queue_i[min_k]][queue_j[min_k]];
72      if (visited[queue_i[min_k]] == 0) {
73        new_node_id = queue_i[min_k];
74      } else {
75        new_node_id = queue_j[min_k];
76      }
77      visited[new_node_id] = 1;
78      // Update the queue.
79      for (i = 0; i < m; ++i) {
80        if (visited[i] == 1) continue; // The edge is to a visited node.
81        queue_end++;
82        queue_i[queue_end] = new_node_id;
```

```
83        queue_j[queue_end] = i;
84      }
85    }
86
87    return -2; // Cannot happen.
88  }
89
90  int main(void)
91  {
92    int n;
93    int i, j;
94
95    scanf("%d", &n);
96
97    for (i = 0; i <= n; ++i) {
98      for (j = i + 1; j <= n; ++j) {
99        scanf("%d", &edges[i][j]);
100       edges[j][i] = edges[i][j];
101     }
102   }
103
104   printf("%d\n", mst_prim_weight(n + 1, 0));
105
106   return EXIT_SUCCESS;
107 }
```

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.

[2] PKU JudgeOnline, “Face The Right Way”, http://poj.org/problem?id=3276 (accessed on 2023-05-24).

[3] 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.