ABC 110 解説

writer: drafear

2018 年 9 月 23 日

A: Maximize the Formula

数式 $XY+Z$ と $Z+XY$ の値は同じなので、$XY+Z$ の形のみを考えることにします。さらに、$XY+Z$ と $XZ+Y$ の値は変わりません。従って、作れる数式の値は $10\times X+Y+Z, X+10\times Y+Z, X+Y+10\times Z$ の 3 通りです。これらの最大値が答えになるので、C++ での実装例は次のようになります。

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int A, B, C; cin >> A >> B >> C;
  int ans = max<int>({
    10 * A + B + C,
    A + 10 * B + C,
    A + B + 10 * C,
  });
  cout << ans << endl;
}
```

さらに考察すると、X, Y, Z のうち最大のものに 10 をかける場合が最大になります。これを C++ で実装すると、例えば次のようになります。

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int A, B, C; cin >> A >> B >> C;
  int mx = max({A, B, C});
```

```
    int ans = A + B + C + 9 * mx;
    cout << ans << endl;
}
```

B: 1 Dimensional World's Tale

X と x_i, Y と y_i は同一視できるので、$x_{N+1} = X, y_{M+1} = Y$ とします。すると、Z が満たすべき条件は次のようになります。

- $x_1, x_2, ..., x_{N+1} < Z$
- $y_1, y_2, ..., y_{M+1} \geq Z$

このような Z は存在するなら x_1 (または -100) より大きく、y_1 (または 100) 以下のはずなので、$Z = -100, -99, ..., 100$ と順に条件を満たすものがあるか調べて解くことができます。これを C++ で実装すると、例えば次のようになります。

```
#include <iostream>
#include <vector>

using namespace std;

bool is_war(const vector<int>& x, const vector<int>& y) {
  for (int Z = -100; Z <= 100; ++Z) {
    bool is_ok = true;
    for (int i = 0; i < x.size(); ++i) {
      if (x[i] >= Z) is_ok = false;
    }
    for (int i = 0; i < y.size(); ++i) {
      if (y[i] < Z) is_ok = false;
    }
    if (is_ok) return false;
  }
  return true;
}

int main() {
  int N, M; cin >> N >> M;
  vector<int> x(N+1), y(M+1);
  cin >> x[0] >> y[0];
  for (int i = 0; i < N; ++i) {
    cin >> x[i+1];
  }
  for (int i = 0; i < M; ++i) {
    cin >> y[i+1];
```

```
  }
  if (is_war(x, y)) {
    cout << "War" << endl;
  }
  else {
    cout << "No War" << endl;
  }
}
```

他の考え方もあります。$x_i \geq y_j$ なる i,j が存在すれば、条件を満たす Z は存在しません (存在したとすると $x_i < Z \leq y_j$ となって矛盾します)。逆に、全ての i,j について $x_i < y_j$ が成立すれば、$Z = min\{y_1, y_2, ..., y_{M+1}\}$ とすることで条件を満たします。

すなわち、全ての i,j について $x_i < y_j$ であるかを調べる方法や、$Z = min\{y_1, y_2, ..., y_{M+1}\}$ として条件を満たすかどうかを調べる方法があります。前者については、代わりに $max\{x_1, x_2, ..., x_{N+1}\} < min\{y_1, y_2, ..., y_{M+1}\}$ であるかどうか調べてもよく、上の実装例の is_war 関数をこの方法で実装すると次のようになります。

```
#include <algorithm>
bool is_war(const vector<int>& x, const vector<int>& y) {
  return *max_element(x.begin(), x.end()) >= *min_element(y.begin(), y.end());
}
```

C: String Transformation

S に操作を繰り返してできる文字列は、('a' から 'z' の) アルファベットとアルファベットを **1 対 1** 対応させた置換表 (変換表) に従って S の各文字を置換したもののみです。ここで 1 対 1 とは、異なるアルファベットは異なるアルファベットに置換されることを指します。すなわち、例えば 'a' が 'c' に, 'b' が 'c' にといった様に、同じアルファベットに置換されるものが複数存在しません。従って、S から T に変換される 1 対 1 の置換表が作れるかを調べれば良いことになります。置換表の条件は、アルファベット S_i はアルファベット T_i に置換されるといったものなので、

- $S_i = S_j$ ならば、$T_i = T_j$ である (同じアルファベットは同じアルファベットに置換される)
- $T_i = T_j$ ならば、$S_i = S_j$ である (異なるアルファベットは異なるアルファベットに置換される)

を満たせば、各 S_i を T_i に置換するような置換表を作ることができます。ここで、S に出現しない文字が存在する場合について言及すると、それらを $a_1, ..., a_k$ とすれば T に出現しない文字も k 個存在するはずなのでそれを $b_1, ..., b_k$ とすると例えば a_i を b_i に置換する置換表でうまく置換されることがわかります。

これを愚直に実装すると $O(|S|^2)$ かかりますが、例えば置換表を表す 26 要素の配列 R, R' (初期値-1) を持って、順に

- $R[S_i] >= 0$ ならば $R[S_i]$ が T_i に等しいことを検査する
- $R'[T_i] >= 0$ ならば $R'[T_i]$ が S_i に等しいことを検査する
- $R[S_i] < 0$ ならば $R[S_i] := T_i$ とする
- $R'[T_i] < 0$ ならば $R'[T_i] := S_i$ とする

を行えば $O(|S|)$ でこの問題を解くことができます。

D: Factorization

$p_1, p_2, ..., p_k$ を異なる素数として、

$$M = p_1^{b_1} p_2^{b_2} ... p_k^{b_k}$$

と表せたとします。$a_1 a_2 ... a_N = M$ とすると、これらもまた、ある c_{ij} が存在して

$$a_i = p_1^{c_{i1}} p_2^{c_{i2}} ... p_k^{c_{ik}}$$

と表せるはずです。このとき、a_i' と a_i'' が異なることと、ある j が存在して c_{ij}' と c_{ij}'' が異なることは同値です。従って、この問題の答えは、各 b_j を $c_{1j}, c_{2j}, ..., c_{Nj}$ に $c_{1j} + c_{2j} + ... + c_{Nj} = b_j$ を満たすように振り分ける場合の数と一致します。これは j に関して独立に計算でき、各 j についての場合の数は、組み合わせを $comb$ で表すとすれば、$comb(b_j + N - 1, b_j)$ です (直感的には、b_j 個のボールを $N-1$ 個の仕切りでグループ分けするイメージで、ボールか仕切りか定まっていない $b_j + N - 1$ 個の連続したオブジェクトから b_j 個の仕切りを選ぶ場合の数です)。よって、答えは

$$comb(b_1 + N - 1, b_1) \times comb(b_2 + N - 1, b_2) \times ... \times comb(b_k + N - 1, b_k)$$

となります。時間計算量は、素因数分解に $O(\sqrt{M})$ かかり、各 b_i は $b_i = O(logM)$ で抑えられるので $comb(b_j + N - 1, b_j)$ の計算は b_j 回かけたり割ったりすれば $O(logM)$ で計算でき、k も高々 $O(logM)$ なので、全体としては $O(\sqrt{M} + (logM)^2)$、すなわち $O(\sqrt{M})$ になります (実は N によりません)。