ABC #062 / ARC #074 Editorial

writer : sugim48

2017/05/20

For International Readers: English editorial starts on page 5.

A: Grouping

すべての (x, y) のペアについて `Yes` / `No` を記録しておく方法では，$\binom{12}{2} = 66$ 通りの場合分けが必要になり，大変です．代わりに，グループに $1, 2, 3$ と番号を振っておき，各 $1 \leq x \leq 12$ ごとに x が属するグループの番号を記録しておけば，x, y が属するグループの番号を比較することで `Yes` / `No` は簡単に判定できます．

C++ のコード例

```
int main() {
        int a[] = {0, 1, 3, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1};
        // 配列の添字は 0 から始まるので，a[0] は 0 にしている．
        int x, y; cin >> x >> y;
        cout << (a[x] == a[y] ? "Yes" : "No") << endl;
}
```

B: Picture Frame

最初に，縦 $H + 2$ 行，横 $W + 2$ 列の文字配列 b を用意し，`#` で埋めておきます．次に，各 $1 \leq i \leq H$, $1 \leq j \leq W$ ごとに $a_{i,j}$ を $b_{i+1,j+1}$ へコピーします．最後に，配列 b の内容を出力すればよいです．

ちなみに，迷路などが入力として与えられたとき，この問題のようにあらかじめ外壁で囲んでおくと，その後の実装が簡単になることがあります．

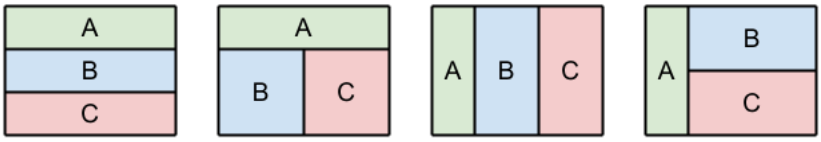
C++ のコード例

```
char a[100][101], b[102][102];

int main() {
        int H, W; cin >> H >> W;
        for (int i = 0; i < H; i++)
                scanf("%s", a[i]);
```

```
        for (int i = 0; i < H + 2; i++)
                for (int j = 0; j < W + 2; j++)
                        b[i][j] = '#';

        for (int i = 0; i < H; i++)
                for (int j = 0; j < W; j++)
                        b[i + 1][j + 1] = a[i][j];

        for (int i = 0; i < H + 2; i++) {
                for (int j = 0; j < W + 2; j++)
                        cout << b[i][j];
                cout << endl;
        }
}
```

C : Chocolate Bar

分割の方法は次図の 4 ケースを考えれば十分です．



左の 2 ケースについて答えを求める方法があれば，H, W を入れ替えて同じ方法を用いることで右の 2 ケースについても答えを求められます．これら 2 通りの答えの最小値が全体の答えとなります．

では，左の 2 ケースについて答えを求める方法を考えましょう．まず，長方形 A の縦幅 h $(1 \leq h \leq H-1)$ を全探索します．10^5 オーダーの計算回数ならば，TLE の心配はありません．これで，長方形 A の面積 S_A は $S_A = hW$ と決まります．あとは，下半分の長方形を長方形 B, C へ分割する方法を決めればよいです．

下半分の長方形を分割する方法は，縦 2 つに分割するケースと，横 2 つに分割するケースがあります．これらはどちらも試すことにしましょう．ここでは，横 2 つに分割するケースを説明します．先程と同様に，長方形 B の横幅を全探索したいところですが，全体で 10^{10} オーダーの計算回数となり，TLE してしまいます．実は，$S_{max} - S_{min}$ をできるだけ小さくするためには，下半分の長方形もできるだけ均等に分割するのがよいことが示せます．できるだけ均等に分割しようとすると，長方形 B の横幅 w は $w = \lfloor W/2 \rfloor$ となります．すると，長方形 B の面積 S_B は $S_B = (H-h)w$，長方形 C の面積 S_C は $S_C = (H-h)(W-w)$ と決まります．S_A, S_B, S_C がすべて決まったので，この分割における $S_{max} - S_{min}$ が求まります．以上すべての分割における $S_{max} - S_{min}$ の最小値が，左の 2 ケースについての答えです．

D : 3N Numbers

数列 a のうち，取り除く N 要素を黒色で，a' の前半 N 要素を赤色で，a' の後半 N 要素を青色で表示することにします．例えば，数列 $(8,2,2,7,4,6,5,3,8)$ の最適解は $(8,2,2,7,4,6,5,3,8)$ となります．

次の条件が成り立つような整数 k を $N \leq k \leq 2N$ の範囲で全探索することにします．

- a の前半 k 要素は黒色または赤色のみで，a の後半 $3N-k$ 要素は黒色または青色のみである．

各 k について答えを求められれば，それらの最大値が全体の答えとなります．では，各 k について答えを求める方法を考えていきましょう．

k をひとつ固定します．このとき，a の前半 k 要素のうち，ちょうど N 要素が赤色で，残りの要素が黒色です．同様に，a の後半 $3N-k$ 要素のうち，ちょうど N 要素が青色で，残りの要素が黒色です．ここで，赤い要素の選び方と青い要素の選び方は独立であることに注意してください．答えは (赤い要素の総和) − (青い要素の総和) の最大値なので，(赤い要素の総和) をできるだけ大きくし，(青い要素の総和) をできるだけ小さくすればよいことが分かります．以降は，(赤い要素の総和) の最大値の求め方のみを考えることにします．

(赤い要素の総和) をできるだけ大きくするためには，a の前半 k 要素のうち大きい方から N 要素を赤くすればよいです．これは，a の前半 k 要素を大きい順にソートし，前半 N 要素の総和を求めれば，可能ではあります．しかし，この方法をすべての k $(N \leq k \leq 2N)$ について行うと，全体の計算量は $\mathrm{O}(N^2 \log N)$ となって TLE してしまいます．すべての k について，a の前半 k 要素のうち大きい方から N 要素の総和を求める効率的な方法はないでしょうか？

これは，優先度付きキューと呼ばれるデータ構造を用いて簡単に行うことができます．優先度付きキューは，「要素の push」「最小要素の pop」がともに $\mathrm{O}(\log N)$ 時間で行えるキューです．あらかじめ，キューに a の前半 N 要素を push しておきます．また，現在キューに入っている要素の総和を保持しておきます．以降，k を N から $2N$ までインクリメントしていきますが，a の前半 k 要素のうち大きい方から N 要素がキューに入っている状態を常に保つことにします．これは，要素 a_k をキューに push した後，キューの最小要素を pop すれば可能です．また，現在キューに入っている要素の総和も，push / pop された要素の分だけ足し引きすれば，保持できます．以上の方法で，すべての k について (赤い要素の総和) が $\mathrm{O}(N \log N)$ 時間で求まります．

同様にして，すべての k について (青い要素の総和) を求めておけば，あとは各 k について (赤い要素の総和) − (青い要素の総和) を計算し，それらの最大値を取れば全体の答えが求まります．

E : RGB Sequence

この問題は $\mathrm{O}(N^3 + N^2 M)$ 時間の DP で解けます．

左から順番にマスの色を塗っていくことにします．既に色を塗ったマスのうち，最も右にある赤いマスの index (1-origin) を r とします．ただし，赤いマスが存在しなければ $r = 0$ とします．同様に，緑のマス，青いマスについてもそれぞれ g, b を定義します．この状態に対応する色の履歴の通り数を $dp[r][g][b]$ とします．

まず，DP の遷移を考えます．$k = \max\{r, g, b\}$ とすると，次に色を塗るマスの index は $k+1$ であることが分かります．このマスに 赤 / 緑 / 青 のどれを塗るかによって，それぞれ r / g / b が $k+1$ へ変わります．例えば，このマスに赤を塗ることに対応する遷移は，`dp[k+1][g][b] += dp[r][g][b]` となります．

次に，条件が成り立たない配色を数え上げない方法について考えます．区間 $[l_i, r_i]$ に対する条件のチェックは，$\max\{r, g, b\} = r_i$ のタイミングで行うことにします．説明のため，$r < g < b (= r_i)$ とすると，「区間 $[l_i, r_i]$ にちょうど x_i 色含まれる」という条件は，l_i と r, g の大小関係の条件で表せます．例えば，$x_i = 2$ の場合，$r < l_i$ かつ $l_i \leq g$ でなければなりません．このような条件が成り立たないような $dp[r][g][b]$ は 0 にしてしまえばよいです．

F : Lotus Leaves

この問題は，グラフの最小カット問題に帰着できます．以降は，グラフの作り方を説明します．

グラフの頂点は，始点 s，終点 t，赤い頂点 1, 2, ..., H，青い頂点 1, 2, ..., W です．赤い頂点 y $(1 \leq y \leq H)$ は，カエルが行 y のどこかのマスにいる，という状態を表します．同様に，青い頂点 x $(1 \leq x \leq W)$ は，カエルが列 x のどこかのマスにいる，という状態を表します．よって，マス S が (y_s, x_s) のとき，始点 s から赤い頂点 y_s と青い頂点 x_s へ，それぞれ容量 ∞ の有向辺を張ります．同様に，マス T が (y_t, x_t) のとき，赤い頂点 y_t と青い頂点 x_t から終点 t へ，それぞれ容量 ∞ の有向辺を張ります．さらに，それぞれの蓮の葉 (y, x) について，赤い頂点 y と青い頂点 x の間に，容量 1 の無向辺を張ります．この無向辺は，カエルが葉 (y, x) で行 y 内の移動と列 x 内の移動を切り替えられる，ということを表します．

以上のグラフにおいて，$s \to t$ 間の最小カットが問題の答えとなります．これは，$s \to t$ 間の最大フローと一致するので，Ford-Fulkerson のアルゴリズムなどで求まります．

ABC #062 / ARC #074 Editorial

writer : sugim48

2017/05/20

A: Grouping

C++ Code Example

```
int main() {
        int a[] = {0, 1, 3, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1};
        int x, y; cin >> x >> y;
        cout << (a[x] == a[y] ? "Yes" : "No") << endl;
}
```

B: Picture Frame

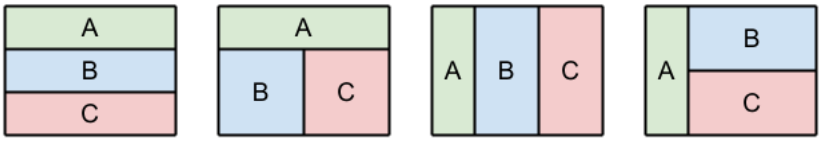
C++ Code Example

```
char a[100][101], b[102][102];

int main() {
        int H, W; cin >> H >> W;
        for (int i = 0; i < H; i++)
                scanf("%s", a[i]);

        for (int i = 0; i < H + 2; i++)
                for (int j = 0; j < W + 2; j++)
                        b[i][j] = '#';

        for (int i = 0; i < H; i++)
                for (int j = 0; j < W; j++)
                        b[i + 1][j + 1] = a[i][j];
```

```
        for (int i = 0; i < H + 2; i++) {
                for (int j = 0; j < W + 2; j++)
                        cout << b[i][j];
                cout << endl;
        }
}
```

C : Chocolate Bar

There are four cases:



In the first two cases, we fix the height h of the rectangle A (and try all possible values of h).

The difference between B and C should be as small as possible. In the first case, the heights of B and C should be $floor((H-h)/2)$ and $ceil((H-h)/2)$, and in the second case, the widths of B and C should be $floor(W/2)$ and $ceil(W/2)$.

The last two cases are similar to the first two cases.

D : 3N Numbers

Let's color the removed elements black, the first N elements of a' red, and the last N elements of a' blue. For example, one optimal solution for the sequence $(8,2,2,7,4,6,5,3,8)$ can be represented as $(8,2,2,7,4,6,5,3,8)$.

There exists an integer $k(N \leq k \leq 2N)$ such that:

- The first k elements of a are either red or black.
- The last $3N-k$ elements of a are either blue or black.

Fix such an integer k. Obviously, we should choose the largest N integers from the first k elements as red elements, and the smallest N integers from the last $3N-k$ elements as blue elements. For each $k(N \leq k \leq 2N)$, we compute the sum of the largest N integers among $a_1, \ldots, a_k$. This is a well-known task, and it can be done in $O(NlogN)$ using priority queue. Similarly, we can compute the sum of blue elements for each k, and the answer is the maximum of red minus blue.

E : RGB Sequence

We color the squares from left to right. Define $dp[r][g][b]$ as the number of ways to color the first $k = \max\{r, g, b\}$ squares, such that:

- The last square we painted red is the r-th square (1-based). $r = 0$ if no square was painted red.
- The last square we painted green is the g-th square.
- The last square we painted blue is the b-th square.
- There's no contradiction in the first k squares: if $r_i \leq k$, the number of different colors in the range $[l_i, r_i]$ must be exactly x_i.

The transitions will be like "`dp[k+1][g][b] += dp[r][g][b]`, `dp[r][k+1][b] += dp[r][g][b]`, `dp[r][g][k+1] += dp[r][g][b]`". However, when $dp[r][g][b]$ is contradictory, we should do "$dp[r][g][b] = 0$;" instead.

This solution works in $O(N^3 + N^2M)$.

F : Lotus Leaves

Convert the grid into a bipartite graph. We should see the cell (i, j) as an edge that connects the i-th red vertex and the j-th blue vertex. Then, a path of the frog is a path in this bipartite graph.

We want to cut S and T by removing leaves (i.e., edges in the bipartite graph). The answer is the minimum s-t cut of the following graph:

- The vertices are s, t, red vertices 1, 2, ..., H, and blue vertices 1, 2, ..., W.
- If $S = (x_s, y_s)$, add an edge between s and the x_s-th red vertex, and an edge between s and the y_s-th blue vertex, each with infinite capacity.
- If $T = (x_t, y_t)$, add an edge between t and the x_t-th red vertex, and an edge between t and the y_t-th blue vertex, each with infinite capacity.
- For each leaf (x, y), add an edge between the x-th red vertex and the y-th blue vertex with capacity 1.