ABC 049 / ARC 065 解説

writer : hogloid

2016 年 12 月 10 日

English Editorial starts on page 5.

A : 居合を終え、青い絵を覆う

文字を入力し、それが a, e, i, o, u のいずれかであるか判定すればよいです。
if の中に 5 つの条件を連ねる以外にも、以下のように母音全てを含む文字列が c を含むか判定することもできます。

C++ のコード例

```
char c;
cin >> c;
const string vowel = "aeiou";
cout << vowel.find(c) == string::npos ? "vowel" : "consonant" << endl;
```

B : たてなが

この問題では、出力する画像をピクセルごとに求める方針でも解くことができますが、出力する画像の 1, 2 行目は元の画像の 1 行目、3, 4 行目は元の画像の 2 行目…のように、上の行から順に 2 回出力を繰り返せばよいです。

C++ のコード例

```
int h, w;
cin >> h >> w;
char buf[105];
for (int i = 0; i < h; ++i) {
  scanf("%s", buf);
  printf("%s\n%s\n", buf, buf);
}
```

C：白昼夢

文字列 S を dream, dreamer, erase, eraser に分解していくことを考えます。先頭から分解していこうとすると、例えば dreamer まで読んだとき、dream で切るべきなのか、dreamer で切るべきなのか判定することができません。(dreameraser は dream eraser と切らなければならないので、dreamer まで読んだときに dream で切らなければいけない場合が存在することが分かります)

逆に、後ろから読んでみましょう。4 つの単語を後ろから読むと、それぞれ maerd, remaerd, esare, resare となります。この 4 つの文字列は、ある文字列が他の文字列の接頭辞 (prefix) になっていないため、後ろから読んで当てはまるものが見つかれば即座に分解するしかありません。(参考: 語頭符号) S を最終的に分解することができなかった場合 `NO` を、そうでない場合 `YES` を出力します。

D：連結

配列 ar で、都市 i と都市 j が道路で連結しているとき $ar[i] = ar[j]$、そうでないとき $ar[i] \neq ar[j]$ となるようなものを取ります。これは、道路の辺を使い深さ優先探索をしたり、道路で直接結ばれた都市をつないだ Union-Find での根の都市の番号を取ることにより実現できます。

同様に、配列 br で、都市 i と都市 j が鉄道で連結しているとき $br[i] = br[j]$、そうでないとき $br[i] \neq br[j]$ となるようなものを取ります。

このとき、i と j が鉄道・道路の両方で連結している $\Leftrightarrow ar[i] = ar[j] \wedge br[i] = br[j]$ です。よって、各 i について、$(ar[i], br[i])$ のようにペアとし、同じペアがいくつ出てくるかを配列のソートや連想配列などを用い求めることで解くことができます。$\mathrm{O}(K + L + NlogN)$ です。

E：へんなコンパス

まず、$(x_i, y_i) \mapsto (x_i + y_i, x_i - y_i)$ と穴の座標を置き換えます。これは、平面を 45 度回転させる操作に対応します。

これにより、$d(i, j) = max(|x_i - x_j|, |y_i - y_j|)$ となります。

必須ではありませんが、この方が何かと実装が平易で済みます。

コンパスにより指される点の組を求めますが、コンパスの指す 2 穴の間の距離 D は常に不変なので、コンパスが指すことのできる穴の集合を求めれば、そこから距離 D の頂点の数を求め総和を取り 2 で割ることで答えが求まります。

コンパスが指すことのできる穴の集合 S は、以下のように再帰的に求めることができます。

擬似コード

```
S := empty
dfs(int v){ // 穴 v をコンパスが指すことができると分かった
  S に v を追加

  S に追加されておらず、v から距離の穴D u それぞれについて{
    dfs(u)
  }
}
dfs(a)
```

ここで、

- S に追加されておらず
- v から距離 D の穴

の 2 の条件をともに満たす穴のみを列挙することで O(N) 回の操作になります。
一つ目の条件が欠けていると、例えば変換後に $x_i = i/2, y_i = (i \bmod 2) * 1000000000$ (割り算は切り捨て) といった場合に、距離 D の穴の組は O(N^2) 個あるため間に合いません。
また、まだ追加されていない全ての頂点について調べる、などを行うと、これもやはり O(N^2) となってしまいます。
「まだ S に追加されておらず、v から距離 D の穴」を 4 つの部分に分けてみましょう。

- $(x_v + D, y_v - D)...(x_v + D, y_v + D)$ の間の点 (v の右側)
- $(x_v - D, y_v - D)...(x_v - D, y_v + D)$ の間の点 (v の左側)
- $(x_v - D, y_v + D)...(x_v + D, y_v + D)$ の間の点 (v の上側)
- $(x_v - D, y_v - D)...(x_v + D, y_v - D)$ の間の点 (v の下側)

さらに、順序集合 (C++ なら std::set) を使ってみましょう。順序を x での比較、同じなら y での比較とします。v の右側の穴を列挙するためには、$(x_v + D, y_v - D)$ 以上 $(x_v + D, y_v + D)$ 以下の穴のみが条件を満たします。これらを 1 つずつ走査していけばよいです。S に v が追加されたときに、この順序集合からも穴の座標を取り除いておけば、一つ目の条件も満たすことができます。左側も同様です。上側・下側の穴は、x, y の比較の順番を逆にした順序集合を持てば同様に実現できます。

コンパスが指すことのできる点から距離 D の穴の数が求める処理が残っています。これは、穴を順序集合で持つ代わりに配列で持ち、ソートし、条件を満たす穴の範囲を二分探索で求め、範囲の長さについて加算すれば実現できます。先ほどの操作とは違い、条件を満たすものの数を重複なく数えなければいけないため、四隅の条件に注意してください。

順序集合からの削除や、二分探索があるため、計算量は O($NlogN$) となります。

おまけ: ソートと O(N) 回の Union-Find の操作の他は線形で解くこともできます。

F：シャッフル

まず、$r_i \geq r_j, i < j$ なる j については、j 番目の操作を行っても行わなくても答えは変わりません。
条件を満たす i のうち最大のものを取ると、$k(i < k \leq j)$ 番目の操作によりシャッフルされる範囲は全て i 番目のシャッフルにより自由に並べ替えることができるためです。
また、$l_i = l_j, r_i > r_j$ なる j も、同様に j 番目の操作を行っても行わなくても答えは変わりません。

これにより、答えの変わらない操作を取り除くことで、$i < j$ のとき $l_i < l_j, r_i < r_j$ を満たすよう操作の列を取ることができます。

次に、シャッフルは部分文字列の配置を全て決めてしまう操作で、これを行うと状態が爆発してしまうため、逐次的な操作に置き換えることを考えましょう。文字列を左から順に決めていくことを考えます。k 番目より左の文字列を決めたとき、

- どこまでがシャッフルされることになっているか (すなわち、自由に並べ替えることができるか)
- シャッフルされることになる部分文字列のうち、1 はいくつあるか

を覚えておけば、1 か 0 を置いていくことができます。なぜなら、シャッフルにより並べ方は自由になってしまうため、1 の数さえ覚えておけば大丈夫だからです。どこまでがシャッフルされることになっているか、は k により一意に決めることができます。$l_i < k$ なる最大の i を取ったときの r_i です。これを $rightend_k$ と表します。

ここで、DP を考えてみましょう。
$dp[k][j] :=$ k 番目より左の文字列は決め、$S[k...rightend_k]$ に 1 が j 個あるときの、k 番目までの文字列の決め方の数
とします。シャッフルされることになる文字列の中の 1 の数を one とします。

- $l_i = k$ となる i があるとき、$S[rightend_k...r_i]$ が新たにシャッフルすることになる部分に加わるので、$one = j +$ ($S[rightend_k...r_i]$ の中の 1 の数) とします。
- そうでないとき、$one = j$ とします。

k 番目の文字を決めるとき、

- $one > 0$ なら 1 を置くことに対応する状態遷移を書きます ($dp[k+1][one-1] += dp[k][j]$)。
- 同様に、全てが 1 でないとき、0 を置く状態遷移を書きます ($dp[k+1][one] += dp[k][j]$)

最終的な答えは、$dp[N+1][0]$ となります。

この DP では、ある文字がどの操作によってもシャッフルされないときを考慮していないため、そのような場合は「1 文字の部分をシャッフルする」(=実質的に何も行わない) ダミーの操作を加えたり、初めから文字列を切り分けてしまうことで対応できます。

計算量は $O(N^2)$ です。

ABC 049 / ARC 065 Rough Editorial

writer : hogloid

2016 年 12 月 10 日

A : UOIAUAI

C++ Code Example

```
char c;
cin >> c;
const string vowel = "aeiou";
cout << vowel.find(c) == string::npos ? "vowel" : "consonant" << endl;
```

B : Thin

Repeat each row twice.

C++ Code Example

```
int h, w;
cin >> h >> w;
char buf[105];
for (int i = 0; i < h; ++i) {
  scanf("%s", buf);
  printf("%s\n%s\n", buf, buf);
}
```

C : Daydream

Reverse the strings. Then, the four words will be: maerd, remaerd, esare, resare. This way you can determine the composition greedily.

D : Connectivity

Construct a graph with roads and find the connected components in it. Then, you can assign values a_i to each city i such that the cities p and q are reachable using roads iff $a_p = a_q$. Similarly, define values b_i for railways.

Then, a pair of two cities satisfies the conditions iff their (a, b) values are the same.

E : Manhattan Compass

Let D be the distance between the two points initially pointed by the compass.

We have two things to do:

- Construct a graph with N vertices. Two vertices a and b are connected if the distance between them is D. Find connected components in this graph.
- For each vertex a, count the number of vertices b such that the distance between a and b is D.

The latter is relatively easy, so we describe the former.

First rotate the entire plane by 45 degrees. Let dx, dy be the difference of x, y coordinates. Now we are intersted in a pair of points such that $max(dx, dy) = D$. To do this, we want to find all pairs of points (a, b) such that

- $x_b - x_a = D$
- $-D \leq y_b - y_a \leq D$

Let's sort the points by the pair (x, y) and renumber them. Then, for a fixed a, the set of b that satisfies the conditions above will form an interval. Let $[L, R]$ be this interval (the interval can be found using binary search). Then, we want to add edges a - L, a - $L + 1$, ..., a - R.

Instead, it is equivalent to add edges a - L and L - $L+1$, $L+1$ - $L+2$, ..., $R-1$ - R. This way, except for $O(N)$ edges, each edge connect adjacent points, so if we compute all edges that connect adjacent points in a clever way, there will be $O(N)$ edges in total.

We should do similar things after swapping x and y. After that, we get the desired graph.

F : Shuffle

If $l_i = l_j, r_i > r_j$, we can ignore the pair j. Thus, we can assume that $M = N$ and for each i, $l_i = i$.

Let $dp[L][R][c]$ be the number of strings we can obtain under the following state:

- We've fixed all characters to the left of L.
- The characters in the range $[L, R)$ are shuffled. c of them are 1.
- In the future, we will perform operations with $l_i \geq L$.

We compute the values as follows:

- When $L = N$, we've finished; the value is 1.
- If $a_L > R$, we can extend the "shuffled range"; change R to a_L and increase c by the number of ones in the interval $[R, a_L]$.
- Now consider two cases for the characters at L. If this is '0' (or '1'), there are $dp[L+1][R][c]$ (or $dp[L+1][R][c-1]$) ways to determine the rest. Thus $dp[L][R][c] = dp[L+1][R][c] + dp[L+1][R][c-1]$. (Make sure not to access invalid values)

This looks like an $O(N^3)$ solution, but we can notice that for a fixed L, only one R appears during the computation. Thus, we can ignore the value of R and it turns out to be an $O(N^2)$ solution.