ARC 091/ABC 090 解説

Kohei Morita(yosupo)

平成 30 年 3 月 19 日

For International Readers: English editorial starts on page 7.

A: Two Coins

A, B, C を入力し，$A + B \geq C$ を判定すれば良いです。
[https://beta.atcoder.jp/contests/abc091/submissions/2218039]

B: Two Colors Card Game

どのカードにも書かれていない，めちゃくちゃな文字列を言えば 0 点を取ることが出来ます。そして，1 点以上を取るためには，少なくとも青いカードのうちどれかに書かれた文字列を言う必要があります。

よって，青いカードに書かれた文字列それぞれについて，その文字列を言ったら何点貰えるかを計算し，その最大値を計算すれば良いです。

[https://beta.atcoder.jp/contests/abc091/submissions/2218376]

C: 2D Plane 2N Points

青い点のうち，最も x 座標が小さいもの (bA とします) に注目します。この点と仲良しペアになれる点が存在した場合，そのような点のうち最も y 座標が大きいもの (rA とします) と仲良しペアにしてよいです。

証明ですが，bA がどの点ともペアにならなかった場合と，rA 以外とペアになった場合それぞれについて，bA と rA をペアにしても同じ個数，もしくはそれ以上の個数の仲良しペアが作れることを示します。

bA がどの点ともペアにならなかった場合　rA がどの点ともペアになっていなかったら，bA と rA をペアにすれば良いです。rA が他の青い点 (bB とします) とペアになっていたら，そのペアを解消し，bA と rA をペアにすれば良いです。

bA が rA 以外とペアになった場合 (bA とペアになっている点を rB とします)　rA がどの点ともペアになっていなかったら，bA の仲良しペアを解消し，bA と rA をペアにすれば良いです。

rA が他の青い点 (bB とします) とペアになっていた場合，つまり bA-rB, rA-bB というペアが出来ていた場合を考えます。この場合は，ペアの相手を交換し，必ず bA-rA, rB-bB というペアに出来ます。これは，以下の 2 つから示せます。

- bA が青い点のなかで一番 x 座標が小さいので，bB の x 座標は必ず bA より大きい，つまり rB よりも大きいです。
- rB は (bA とマッチング出来ていたので，) 必ず rA より y 座標が小さいです。また，bB は (rA とマッチング出来ていたので，) 必ず rA より y 座標が大きいです。よって，bB の y 座標は，必ず rB より大きいです。

以上より，bA はかならず rA とペアにしていいことが示せます。
この性質より，青い点を x 座標の小さい順に見ていき，

- 青い点より x, y 座標が小さく，まだ仲良しペアになっていない赤い点を探す
- なかったらなにもしない
- あったら，その中で最も y 座標が大きいものを探し，仲良しペアにする

というアルゴリズムで最適解が求まります。

余談　この問題は，二部グラフの最大マッチング問題としても考えることが可能で，これは最大流アルゴリズムを使用すれば解くことが出来ます。

D: Two Sequences

まず，この問題では bit は 0-indexed とします。つまり，2 進数で一番下の桁から，0-bit 目，1-bit 目, ..., と呼びます。

答えを各 bit ごとに求めます。k-bit 目を求めたいとします。

xor の定義より，N^2 個の $a_i + b_j$ のうち，k-bit 目が 1 のものが偶数個あるか奇数個あるか判定できれば良いです。

ここで重要な考察として，$a_i + b_j$ の k-bit 目を考えているので，a_i, b_j の $k+1$-bit 目以降は無視してよい，つまり a_i, b_j は 2^{k+1} で mod を取ってしまって良いというのがあります。

すると，$a_i + b_j$ はたかだか $4T (T = 2^k)$ 未満です。そして，k-bit 目が 1 の範囲は，$[T, 2T), [3T, 4T), [5T, 6T), ...$ なので，$a_i + b_j$ のうち，$[T, 2T)$, $[3T, 4T)$ の範囲にあるものの個数を求めればよいことがわかります。

これは，a_i を固定してしまえば，b_j のうち $[T - a_i, 2T - a_i), [3T - a_i, 4T - a_i)$ の範囲にあるものの個数を求めれば良くなり，これは b_j を sort しておけば二分探索で $O(logN)$ で求められます。

余談　この問題の $O(N^2)$ 解は非常に単純なうえ，SIMD で高速化が効き，とてつもなく高速に動作します。

writer が素直な $O(N^2)$ を書き，ブロック化を行ったら 3770ms となったので，SIMD が得意な人ならばもしかすると愚直解で通せてしまうのではとも考えています。

ぜひ SIMD が得意な人は挑戦してみてください。

E: Both Sides Merger

最終的な数列の要素は，当然 a_i をいくつか足したものになります。

ここで，この i の集合に注目します。例えばサンプル 1 なら最終的な要素は $a_2 + a_4$ なので，$\{2, 4\}$ という集合に注目します。

この集合にはどのような性質があるでしょうか？実は，必ずこの集合の要素は，偶奇は全て等しくなります。

つまり，偶奇の違うペア，例えば a_1, a_2 や a_4, a_7 は，絶対に一つの要素に合わせられないです。

これは，どのような操作をしてもこのペアの要素は偶奇が異なったまま (なので，一つの要素にまとめることは不可能) であることから示せます。

実は，これが必要十分です。つまり，すべての要素が偶奇が等しい集合が与えれば，最終的な要素がそれに対応した要素の和になるような手順が必ず存在します。

構築の方法ですが，両端のいらない要素をまず全てカットします。その後

- 両端がどちらも残したい要素，かつ自分自身は残したくない要素 (A)
- 両端がどちらも残したくない，かつ自分自身も残したくない要素 (B)

この A, B を満たす要素のうちどちらかは必ず取れるので，これを取っていけばかならず要素数が少なくなっていき，最終的に 1 つになります。

なぜこのどちらかは必ず存在するのかというと，集合の要素の偶奇が等しいということは，oxxxoxo，のように残したい要素同士の間には，残したくない要素が奇数個挟まっています。

もし 1 個挟まっていれば，その要素は A の条件を満たし，3 個以上挟まっていれば，(そのうち両端以外は)B の条件を満たします。よって，必ずどちらかを満たす要素が存在します。

以上より，この問題は，「数列から添字の偶奇が同じように要素を取って，総和を最大化してください」という問題になります。これは，偶奇を固定し，0 以上の要素をすべて取るだけでよいです。

コーナーケースは全部の要素が負の場合です。十分注意してください。

F: Two Faced Edges

各辺 i に対し，以下の条件を満たすかを調べたいです。

- 1. もし辺 (a_i, b_i) を消したとしても，a_i から b_i に到達可能か？
- 2. b_i から a_i に到達可能か？

この 2 つの条件を両方達成するか，両方達成しない場合，辺の反転で SCC の成分数が変わらないことが容易に示せます。

条件 2 は $O(N+M)$ や $O(N(N+M))$ で容易に調べられます。ですので，この問題の本質は条件 1 を高速に求めることです。

方針としては，頂点を固定し，そこから出ていく辺すべてについての条件 1 を $O(N+M)$ で求めます。つまり，a_i を固定します。

まず，a_i から出てくる辺に，適当に連番を振っておきます。

当然 (a_i, b_i) を使わずに a_i から b_i に到達可能ならば，a_i から出ていく他の辺を使う必要があります。そしてその辺の番号は (a_i, b_i) より小さいかまたは大きいです。

よって，まずは小さい辺を使った場合のみ，つまり以下の条件を考えます。当然小さい辺を使った場合のみに限定して解ければ，辺に降った番号を逆順にしてもう一度同じ問題を解けば良いので，この問題が解けます。

- 3. もし辺 (a_i, b_i) を消したとしても，a_i から (a_i, b_i) より小さい番号の辺を使い，b_i に到達可能か？

条件 3 は，辺の番号が小さい順に辺を追加していき，追加する直前に今まで追加した辺を利用して b_i に到達できるか，を調べられれば良いです。

これは，辺を追加するたびに dfs を走らせ，二度同じ頂点は辿らないようにするだけで，まとめて $O(N+M)$ となります。

よってこの問題はまとめて $O(N(N+M))$ で解けました。

A: Two Coins

[https://beta.atcoder.jp/contests/abc091/submissions/2218039]

B: Two Colors Card Game

[https://beta.atcoder.jp/contests/abc091/submissions/2218376]

C: 2D Plane 2N Points

Let A be a blue point with the minimum x-coordinate. If A can be paired with red points, we should choose a red point B such that B can be paired with A, and among all such points the y-coordinate is the maximum possible.

(If B is paired with other point, for example if there are pairs $B-C$ and $A-D$, we can rearrange them because $A-C$ and $B-D$ are also valid pairs).

By doing this, we can reduce the number of points by 1 (if A can't be paired at all) or 2 (if we find a pair $A-B$).

In summary, we first sort all $2N$ points together by x-coordinates, and process points in this order. We also keep a set of points (initially this is empty). Then, for each point, do the following:

- If a red point appears, add its y-coordinate to the set.
- If a blue point appears, let p be its y-coordinate. If all elements in the set are greater than p, do nothing (we can't do anything with this blue point). Otherwise, choose the maximum element q in the set such that $q < p$, remove it, and increase the answer by one (the current blue point and the point that corresponds to q are paired).

D: Two Sequences

Since we can handle each bit independently, let's consider only the k-th bit. We want to count the number of pairs (i, j) such that the k-th bit of $a_i + b_j$ is one. (And if this number is add, we should add 2^k to the answer.)

Let $T = 2^k$. An important observation is that, we are only interested in the values of a_i, b_i in modulo $2T$. Thus, let's replace a_i with $a_i\%(2T)$ and b_i with $b_i\%(2T)$, and assume that $a_i, b_i < 2T$.

Then, there are two cases when the k-th bit of $a_i + b_j$ is one:

- $T \leq a_i + b_j < 2T$
- $3T \leq a_i + b_j < 4T$

Let's sort b in an increasing order. For a fixed i, the set of j that satisfies $T \leq a_i + b_j < 2T$ forms an interval. Thus, we can count the number of such js by binary search (or two-pointers). Similarly, we can handle the second case.

This solution works in $O(NlogNlogMAX)$ time, where MAX is the maximum number that can appear in the input.

E: Both Sides Merger

The answer of this problem will be the sum of a subset of $\{a_i\}$.

For example, consider example 1. In this sample, the answer is $a_2 + a_4$, so the set of indices is $\{2, 4\}$

Suppose that we are given a set of indices, I. Let's check if we can perform operations such that the final integer corresponds to the sum of all elements with indices in I.

It turns out that the following are neccesary and sufficient conditions:

- The parity of all indices in I are the same.
- I is non-empty.

It's easy to prove that these conditions are necessary. Suppose that initially, x is in an odd-indexed position and y is in an even-indexed position. Then, no matter what operations we perform, the parities of the indices of x, y never become the same.

These are also sufficient conditions because we can construct a sequence of operations, as we describe below.

To compute the answer, fix a parity, and take all non-negative elements in the positions with chosen parity. When all elements are negative, you need to be careful not to make it empty.

To construct a sequence of operations, we first remove all unnecessary elements from both ends. Then, we repeat choosing elements while we have more than one elements. We can choose an element x if one of the following holds:

- x is adjacent to two elements (call it y, z), and we want to keep both y and z in the final element.
- x is adjacent to two elements (call it y, z), and we want to remove both y and z from the array.

It's easy to see that such elements always exist.

F: Two Faced Edges

For each edge i, we want to determine the following two things:

- 1. If we remove an edge from a_i to b_i, is b_i still reachable from a_i (by using other edges)?
- 2. Is a_i reachable from b_i?

It's easy to see that the number of SCCs won't change after the reversion of edge i if and only if the answers to the two questions above are the same (i.e., both are yes or both are no). Since the second question can be solved easily in $O(NM)$ by running dfs from all vertices, we will focus on the first question.

Consider a particular vertex x. We will solve the first question for all edges such that $a_i = x$, in $O(M)$ time. Then, we can solve the entire problem in $O(NM)$.

Suppose that there are k edges that goes out of the vertex x, and let's name the destinations $y_1, \ldots, y_k$. We will do the following:

- By running a dfs from y_1, we mark all vertices that are reachable from y_1 without visiting x.
- By running a dfs from y_2, we mark all unmarked vertices that are reachable from y_2 without visiting x.
- By running a dfs from y_3, we mark all unmarked vertices that are reachable from y_3 without visiting x.
- And so on.

Then, for each vertex z, we know the minimum integer $p(z)$ such that z is reachable from x by using an edge $x- > y_{p(z)}$ (and without visiting the vertex x again). Similarly, by running dfs in the reverse order, for each vertex z, we know the maximum integer $q(z)$ such that z is reachable from x by using an edge $x- > y_{q(z)}$ (and without visiting the vertex x again). The question 1 for an edge $x- > y_i$ is "yes" if and only if at least one of $p(y_i) \neq i$ or $q(y_i) \neq i$ holds.

This solution works in $O(NM)$ time.