ABC 172 解説

beet, evima, kyopro_friends, sheyasutaka, tozangezan, ynymxiaolongbao

2020 年 6 月 27 日

For International Readers: English editorial starts on page 9.

A: Calc

(解説: evima)

プログラミングの学習を始めたばかりで何から手をつけるべきかわからない方は、まずは「practice contest」(`https://atcoder.jp/contests/practice/`) の問題 A「はじめてのあっとこーだー」をお試しください。言語ごとに解答例が掲載されています。

今回の問題に含まれる要素であって「はじめての〜」に含まれないものは、a^2, a^3 といった値の計算のみです。言語を問わずに使える確実な方法は、`a * a * a` などとして計算してしまうことです。言語に `**`, `^` といった累乗演算子が用意されていればそれを使うこともできますが、結果が実数ではなく整数として計算されることを確認してからの方が安全かもしれません。

`pow()` といった名前の標準ライブラリも多くの言語に用意されていますが、それらは実数の実数乗を計算するものであることが多く、整数の整数乗の計算に用いるには適さない可能性があります。$2^{53}-1$ を超える整数は 64 ビット実数型では正しく表現できないため、計算結果がその範囲にあると正しい結果が得られません[*1]。また、正しい計算結果が得られても、`cout << a + pow(a, 2) + pow(a, 3)` などと実数のまま出力して `1.0101e+06` などとなってしまうと不正解と判定されます (この問題に限らず、AtCoder では、整数での出力が期待される問題で実数形式の数値を出力すると不正解と判定されます)。ただし、今回の $1 \leq a \leq 10$ という制約下では計算結果が誤っていることはないはずで、出力形式に関しても都合よく処理する言語が多いようです。

[*1] 今回の制約には反しますが、例えば $a = 1000000$ の場合、$a + a^2 + a^3$ を 64 ビット実数を用いて計算した結果を整数にキャストすると 1000001000000999936 となります

C++ での実装例:

```cpp
#include <iostream>
using namespace std;
int main(){
    int a;
    cin >> a;
    cout << a + a * a + a * a * a << endl;
}
```

B: Minor Change

(解説: evima)

問題文は厳密を期してやや堅苦しく書かれていますが、平たくいえば「S と T は何か所が異なるか？」という内容に過ぎません。

主な課題は、二つの文字列の長さが一定でない上に大きいことです。「if 文」の扱いは Beginner Contest の問題 A でも要求されますが、今回の問題も

```
if length(S) >= 100 and S[99] != T[99] then ans += 1
```

といった行を 20 万行並べて理論上は解くことができます。しかし、ソースコードが数百万文字以上の長さになり、AtCoder に提出できません (提出可能な最大長は 52 万文字程度です)。

20 万行の if 文を不要にするには、「for 文」と呼ばれるループ構造を使うのが最も素直です。`if S[i] != T[i] then ans += 1` という処理を for ループ内で $i = 0, i = 1, \ldots, i = (S \text{ の長さ}) - 1$ のそれぞれに対して実行させれば、一個の if 文で済みます。「for 文」の言語ごとの詳細は検索エンジンで「[言語名] for」などと検索することで得られるはずです。

C++, Python での実装例 (後者は「for 文」以外のアプローチの例として):

```cpp
#include <iostream>
using namespace std;

int main(){
    string S, T;
    cin >> S >> T;
    int N = S.size(), ans = 0;
    for(int i = 0; i < N; ++i){
        if(S[i] != T[i]) ++ans;
    }
    cout << ans << endl;
}
```

```python
S, T = input(), input()
print(len(list(filter(lambda i: S[i] != T[i], range(len(S))))))
```

C: Tsundoku

(解説: evima)

結局のところ、机 A, B からそれぞれ何冊の本を読むか以外に行うべき選択はありません。$a_0 = 0, a_1 = A_1, a_2 = A_1+A_2, a_3 = A_1+A_2+A_3, \ldots, a_N = A_1+\cdots+A_N$ とします。$b_0, b_1, \ldots, b_M$ も同様に定義します。これらの値をプログラムで求める際は、$a_1 = a_0+A_1, a_2 = a_1+A_2, a_3 = a_2+A_3, \ldots$ などとします (毎回 A_1 から足していくと時間切れになります)。すると、机 A から i 冊、机 B から j 冊読むことを考えれば、問題は次のように言い換えられます。

> 整数 i, j $(0 \leq i \leq N, 0 \leq j \leq M)$ を $a_i + b_j \leq K$ となるように選ぶとき、$i + j$ がとりうる最大の値を求めよ。

N, M がある程度小さければ、$(i, j) = (0, 0), (0, 1), \ldots, (0, M), (1, 0), \ldots, (N, M)$ という $(N + 1) \times (M + 1)$ 通りすべての選択を実際に試すことでこの問題を解くことができます。しかし、$N = M = 200000$ の場合には少なくとも数十秒の実行時間が必要で、間に合いません。

そこで、i についてのみ $i = 0, 1, \ldots, N$ というすべての選択を試し、このそれぞれについて選択可能な j の最大値、つまり $b_j \leq K - a_i$ であるような j の最大値を求めることにします。

まず、$i = 0$ のときに選択可能な j の最大値を求めます。これは、$j = M, M - 1, \ldots$ と降順に走査すると、$b_j \leq K - a_0$ が最初に成立した j として求まります。この j の値を $best_0$ とします。

次に、$i = 1$ のときに選択可能な j の最大値を求めます。ここで、本一冊を読むのにかかる時間が正であること (a, b がともに単調増加数列であること) から、この最大値は $best_0$ 以下です。よって、降順の走査を $j = M$ から始める必要はなく、$j = best_0$ から始めることができます。

この要領で $i = N$ まで計算を続ける (ただし $a_i > K$ となったら打ち切る) ことで、計算量を大幅に削減することができます。次ページの実装例 (Python のコードではありますが、主要部である 11 行目から 18 行目まではほぼ疑似コードといえるでしょう) で考えると、15, 16 行目の while ループの内部は合計で M 回以下しか実行されないため、時間計算量は $\mathrm{O}(N + M)$ となります。

Python での実装例:

```python
N, M, K = map(int, input().split())
A = list(map(int, input().split()))
B = list(map(int, input().split()))

a, b = [0], [0]
for i in range(N):
    a.append(a[i] + A[i])
for i in range(M):
    b.append(b[i] + B[i])

ans, j = 0, M
for i in range(N + 1):
    if a[i] > K:
        break
    while b[j] > K - a[i]:
        j -= 1
    ans = max(ans, i + j)
print(ans)
```

D: Sum of Divisors

(解説: evima)

一般に、正整数 X に対して、$f(X)$ つまり X の約数の個数は $\mathrm{O}(\sqrt{X})$ 時間で求められます[*2]。しかし、$X = 1, \ldots, 10^7$ のすべてに対してこれを行う時間はありません。

そこで、$1, \ldots, N$ の約数の個数を求めることなく問題で要求された値を計算することを考えます。要求された値は、以下の疑似コードが出力する値であると考えられます。

```
1 ans = 0
2 for i = 1, ..., N:
3     for j = 1, ..., N:
4         if i % j == 0 then ans += i
5 print ans
```

ここで、4 行目は結局のところ $(i, j) = (1, 1), \ldots, (1, N), (2, 1), \ldots, (N, N)$ の N^2 通りのペアに対して実行されるため、次のように 2, 3 行目の for 文を入れ替えても結果は変わりません。

```
1 ans = 0
2 for j = 1, ..., N:
3     for i = 1, ..., N:
4         if i % j == 0 then ans += i
5 print ans
```

このコードの挙動を考えると、問題は次のように言い換えられます。

> 正整数 j に対し、j の倍数であって N 以下であるものの総和を $g(j)$ とする。$\sum_{j=1}^{N} g(j)$ を求めよ。

この $g(X)$ は等差数列の和であり、$f(X)$ より高速に求めることができます。具体的には、$Y = \lfloor N/X \rfloor$ (N/X 以下の最大の整数) とすると $g(X) = X + 2X + \ldots + YX = (1 + 2 + \ldots + Y)X = Y(Y+1)X/2$ であり、この式を用いれば $\mathrm{O}(N)$ 時間で問題が解けます。

(一部の言語では、上の式を用いても実行制限時間に間に合わない可能性があります。その場合は、心苦しいですが、他の言語を用いてくださいというほかありません。制約における N の上限を小さくしてしまうと、$X = 1, \ldots, N$ のすべてに対して $\mathrm{O}(\sqrt{X})$ 時間で約数を列挙する方針を C++ などの言語で実装すれば間に合いかねないため、このような制約となりました。)

[*2] $i = 1, \ldots, \lfloor \sqrt{X} \rfloor$ ($\sqrt{X}$ 以下の最大の整数) のそれぞれについて X を割り切るか確かめ、割り切るなら i と X/i は X の約数である、とすることで X のすべての約数が得られます

E. NEQ

(解説: ynymxiaolongbao)

$1 \leq i \leq N$ なる任意の i について $A_i \neq B_i$ であるという条件を一旦無視します。すなわち、1 以上 M 以下の整数からなる長さ N の数列 A と B の組であって、$1 \leq i < j \leq N$ なる任意の (i, j) について $A_i \neq A_j$ かつ $B_i \neq B_j$ であるようなもの全体を考えます。

各要素が 1 以上 N 以下の整数である集合 S について、「$i \in S$ なるすべての i について $A_i = B_i$ である」という条件を満たす A と B の組の個数は ${}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$ です。この式は、1 以上 M 以下の整数から $|S|$ 個を並べて $i \in S$ なる i の $A_i (= B_i)$ の値とし、残りの $M - |S|$ 個の整数から $N - |S|$ 個を並べて $i \notin S$ なる i の A_i の値とし、同様に $M - |S|$ 個の整数から $N - |S|$ 個を並べて $i \notin S$ なる i の B_i の値とすることを考えることで得られます。

包除原理より、すべての 1 以上 N 以下の整数の集合 S について、「$i \in S$ なるすべての i について $A_i = B_i$ である」という条件を満たす A と B の組の個数を求め、それに $(-1)^{|S|}$ を掛けた値の総和が答えになります。すなわち $(-1)^{|S|} \times {}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$ の総和が答えになります。

ここで、$(-1)^{|S|} \times {}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$ の値を 2^N 通りすべての S について計算していては間に合いませんが、この値が $|S|$ のみに依存することに注目すると、$|S|$ に対する値に $|S|$ がその値になるような S の個数、すなわち ${}_NC_{|S|}$ を掛けたものを足し合わせることでも答えを求めることができます。つまり、答えは $\sum_{K=0}^{N}({}_NC_K \times (-1)^K \times {}_MP_K \times ({}_{M-K}P_{N-K})^2)$ と表現できます。

コンビネーションや順列は、フェルマーの小定理を用いる方法や、拡張ユークリッドの互除法を用いる方法で高速に求めることができます。前者を用いれば、計算量は $O(M + N)$ になります。

F: Unfair Nim

(解説: evima)

タイトルで示唆されているように、このゲームは Nim と呼ばれる有名なもので、$A_1 \oplus A_2 \oplus \ldots \oplus A_N = 0$ ($\oplus$ は XOR: ビットごとの排他的論理和) であれば後手が必勝法を持ち、そうでなければ先手が必勝法を持つことが知られています。この問題を解く上では、Nim についてはこの知識を持ってさえいれば十分であり、詳細には立ち入りません。

上記より、$A_1 + A_2$ を S、$A_3 \oplus A_4 \oplus \ldots \oplus A_N$ を X として、問題は次のように言い換えられます。

> $a + b = S, a \oplus b = X, a \leq A_1$ であるような非負整数のペア (a, b) が存在するか判定し、存在する場合はそのようなペアで a がとりうる最大の値を求めよ。

(元の問題で第 1 の山からすべての石を移すことが禁じられていることが反映されていませんが、答えが 0 であった場合に代わりに「存在せず」と報告することで対処できます)

解法の一つは、問題をさらに言い換えて「A_1 以下の非負整数 a であって $a + (a \oplus X) = S$ を満たす最大のものを求めよ」として、動的計画法により実質的に a の値の全探索を行うことです。

しかし、$a + b = (a \oplus b) + 2 \times (a \ \texttt{\&}\ b)$ (`&` は AND: ビットごとの論理積。この式が成り立つことは、XOR が「繰り上がりを無視した足し算」であることを考えればわかります) であることを利用すると、探索の必要はなくなります。条件を満たすような (a, b) について、先ほどの式より $a \ \texttt{\&}\ b = (S - X)/2$ (この値を D とします) が成り立ちます。D が非負整数でなかったり、D と X で共通して立っているビットが一か所でもある場合、答えは「存在せず」です。

そうでなければ、$a \leq A_1$ という条件を一旦無視すると、X で立っているビットの集合を二分割して得られる整数を Y, Z としたとき ($Y \ \texttt{\&}\ Z = 0, Y \oplus Z = X$)、$a = D \oplus Y, b = D \oplus Z$ というペアは残りの条件を満たします。また、この方法で得られるペアの他に残りの条件を満たすペアはありません。このようなペアにおける a の値のうち、$a \leq A_1$ を満たす最大のものが求める答えです。

まず、このようなペアにおける a の最小の値は $Y = 0$ としたときの $a = D$ で、この値が $a \leq A_1$ を満たさなければ答えは「存在せず」です。そうでなければ、$a \leq A_1$ が満たされる範囲で $a = D \oplus Y$ をできるだけ大きくするように Y を選択する必要があります。結論から述べると、まず $Y = 0$ として、X で立っているビットを一つずつ降順に取り上げ、「このビットを Y に入れても $a = D \oplus Y \leq A_1$ は成立するか？」と問い、答えが Yes なら入れる、という貪欲な戦略が最適です。これは、数の大小の定義 (任意の桁は、それより下のすべての桁を合わせたものより重要) から示せます。よって、言い換え後の問題は $O(\log X)$ 時間で解けます。

A: Calc

(Editorial: evima)

If you are a beginner of studying programming and have no idea what to start with, then try solving "Welcome to AtCoder", Problem A from the "practice contest" (`https://atcoder.jp/contests/practice/`). There you can find sample codes for each language.

The only element in this problem that is not covered by "Welcome to —" is the calculation of a^2, a^3 and so on. The most certain way that is available regardless of languages is calculating like `a * a * a`. If the language has operators like `**` or `^`, then you can also use them, but it might be safer to ascertain that the result is calculated as integers, not as real numbers.

Most languages provides with standard libraries named `pow()` or similar, but they tend to aim at calculating a real-number power of real number, and not be suitable for calculating an integer-power of integer. Since integers exceeding $2^{53} - 1$ cannot be represented accurately by 64-bit real numbers, so if the result is not within the range, you cannot obtain accurate result[*3]. Also, even if you obtained an accurate result, if you output as real number like `cout << a + pow(a, 2) + pow(a, 3)`, you will obtain outputs like `1.0101e+06`, which will be judges as wrong answer (not only in this problem; in any AtCoder problems which expects integer outputs, if you output in a real-number style, it will be judged as being wrong answer). However, under the constraints $1 \leq a \leq 10$ this time, the calculation result must be accurate, and also output format seems to be processed suitably in many languages.

[*3] If $a = 1000000$, though this is not within the constraints this time, calculating $a + a^2 + a^3$ with 64-bit real numbers results in 1000001000000999936 when casted to integer

Sample code in C++:

```cpp
#include <iostream>
using namespace std;
int main(){
    int a;
    cin >> a;
    cout << a + a * a + a * a * a << endl;
}
```

B: Minor Change

(Editorial: evima)

The problem statement is written rather formally for the sake of strictness, but in fact the problem simply asks "how many letters are different between S and T?"

The main issue is that the lengths of the strings are not constant, and large. The treatment of "if statements" is required in Problem A of Beginner Contests, and this problem can be theoretically solved by repeating 200 thousands lines of if statements like

```
if length(S) >= 100 and S[99] != T[99] then ans += 1
```

However, the source code would have millions of characters, which is not submittable to AtCoder (The possible maximum length is about 520 thousands).

To avoid 200 thousand if statements, most straightforward way is to use the structure called "for statements." If you execute the operation of `if S[i] != T[i] then ans += 1` for each $i = 0, i = 1, \ldots, i =$ (`length of` S) $- 1$ in the for statement, you just have to write if statement once. You can obtain more details of for statement by searching like "[language name] for".

Sample code in C++ and Python (the latter of which is as an example of an approach other than "for statement")

```cpp
#include <iostream>
using namespace std;

int main(){
    string S, T;
    cin >> S >> T;
    int N = S.size(), ans = 0;
    for(int i = 0; i < N; ++i){
        if(S[i] != T[i]) ++ans;
    }
    cout << ans << endl;
}
```

```python
S, T = input(), input()
print(len(list(filter(lambda i: S[i] != T[i], range(len(S))))))
```

C: Tsundoku

(Editorial: evima)

After all, the only choice you can make is how many books to read from desk A and B. Let $a_0 = 0, a_1 = A_1, a_2 = A_1 + A_2, a_3 = A_1 + A_2 + A_3, \ldots, a_N = A_1 + \cdots + A_N$. We also define $b_0, b_1, \ldots, b_M$ similarly. When calculating those value in programs, we calculating in the way like $a_1 = a_0 + A_1, a_2 = a_1 + A_2, a_3 = a_2 + A_3, \ldots$ (if you sum up from A_1 every time, it will exceed time limit). Then, considering reading i books from desk A and j books from desk B, the problem can be rephrased as follows:

> For integers i, j $(0 \leq i \leq N, 0 \leq j \leq M)$ such that $a_i + b_j \leq K$, find the maximum possible value of $i + j$.

If N and M are fairly small, we can solve this problem by trying all possible $(N+1) \times (N+1)$ combinations of $(i, j) = (0, 0), (0, 1), \ldots, (0, M), (1, 0), \ldots, (N, M)$ However, when $N = M = 200000$, we need at least tens of seconds of execution time, which will exceed the time limit.

Then, let us try exhaustive search for only $i = 0, 1, \ldots, N$, and for each i, find the maximum choosable j, that is, maximum j such that $b_j \leq K - a_i$.

First, let us find the maximum choosable j when $i = 0$. This can be found as the first j such that $b_j \leq K - a_0$ when scanning in the order of $j = M, M - 1, \ldots$. Let such j be $best_0$.

Next, we will find the maximum choosable j when $i = 1$. Since the time needed to read a book is positive (therefore a and b are both monotonic increasing sequence), the maximum value no more than $best_0$. Therefore, you don't have to start scanning in the decreasing order from $j = M$; instead, you can start it from $j = best_0$.

By continuing calculation until $i = N$ in the similar way (stopping when $a_i > K$ though), you can dramatically decrease the execution time. In the sample code on the next page (although it is written in Python, the cardinal part of line 11 to 18 is readable as pseudocode), the while loop on line 15 and 16 is at most executed M times, so the total time complexity is $O(N + M)$.

Sample code in Python:

```python
N, M, K = map(int, input().split())
A = list(map(int, input().split()))
B = list(map(int, input().split()))

a, b = [0], [0]
for i in range(N):
    a.append(a[i] + A[i])
for i in range(M):
    b.append(b[i] + B[i])

ans, j = 0, M
for i in range(N + 1):
    if a[i] > K:
        break
    while b[j] > K - a[i]:
        j -= 1
    ans = max(ans, i + j)
print(ans)
```

D: Sum of Divisors

(Editorial: evima)

Generally, for a positive integer X, the number of divisors of X, $f(X)$, can be calculated in a total of $\mathrm{O}(\sqrt{X})$ time. [*4]. However, the time limit is not long enough to execute this for all $K = 1, \ldots, 10^7$.

Then, let us consider finding the value desired in the problem statement without finding the number of divisors of $1, \ldots, N$. The desired answer can be considered as the value that the following pseudocode outputs.

```
1 ans = 0
2 for i = 1, ..., N:
3     for j = 1, ..., N:
4         if i % j == 0 then ans += i
5 print ans
```

Here, line 4 is executed for all N^2 pairs of $(i, j) = (1, 1), \ldots, (1, N), (2, 1), \ldots, (N, N)$, so you can swap the for statements in the lines 2 and 3 without changing the result.

```
1 ans = 0
2 for j = 1, ..., N:
3     for i = 1, ..., N:
4         if i % j == 0 then ans += i
5 print ans
```

Considering the behavior of the code, the problem can be reworded as follows:

> For a positive integer j, we define $g(j)$ as the sum of multiples of j that does not exceed N. Find $\sum_{j=1}^{N} g(j)$,

This $g(X)$ is a sum of an arithmetic progression, and can be calculated faster than $f(X)$. Specifically, let $Y = \lfloor N/X \rfloor$ (the maximum integer that does not exceed N/X), then $g(X) = X + 2X + \ldots + YX = (1 + 2 + \ldots + Y)X = Y(Y+1)X/2$, so by using this formula, the problem can be solved in a total of $\mathrm{O}(N)$ time.

(In some language, even if you use the equations above, your program may not finish in the time limit. In such case, regretfully, there is no way but using other languages. If the upper bound of N is smaller, iterating all the divisors of all $X = 1, \ldots, N$ in $\mathrm{O}(\sqrt{X})$ time each might

[*4] For each $i = 1, \ldots, \lfloor \sqrt{X} \rfloor$ (maximum integer not exceeding $\sqrt{X}$), check if it divides X, and if it does, then i and X/i are the divisors of X; by doing so you can obtain all the divisors of X

be finish in the time limit when implemented in languages like C++, and this is the reason for such constraints.)

E. NEQ

(Editorial: ynymxiaolongbao)

We first ignore the conditions that $A_i \neq B_i$ for any i such that $1 \leq i \leq N$. In other words, we consider all the pairs A and B of sequences of length N consisting of integers between 1 and M (inclusive; the same applies hereinafter) such that, for all (i, j) such that $1 \leq i < j \leq N$, $A_i \neq A_j$ and $B_i \neq B_j$.

For a set of integers whose each element is between between 1 and N, the number of pairs A and B of sequences such that "for all $i \in S$, $A_i = B_i$" is ${}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$. This expression can be obtained by considering the $|S|$-permutation of integers between 1 and M assigned to $A_i (= B_i)$ for each i such that $i \in S$, $N - |S|$-permutation of the remaining $M - |S|$ integers assigned to A_i for each i such that $i \notin S$, and similarly $N - |S|$ permutation of the $M - |S|$ integers assigned B_i for each i such that $i \notin S$.

By inclusion-exclusion principle, the answer is the sum of the number of pairs of sets A and B such that "for all $i \in S$, $A_i = B_i$", multiplied by $(-1)^{|S|}$, for all sets of integers between 1 and N. In other words, the answer is the sum of $(-1)^{|S|} \times {}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$.

Here, calculating $(-1)^{|S|} \times {}_MP_{|S|} \times ({}_{M-|S|}P_{N-|S|})^2$ for all 2^N possibilities of S, but since this value only depends on $|S|$, you can also calculate the sum by finding the value for $|S|$ multiplied by the number of S such that $|S|$ is that value, that is, multiplied by ${}_NC_{|S|}$. Therefore, the answer is can be represented as $\sum_{K=0}^{N} ({}_NC_K \times (-1)^K \times {}_MP_K \times ({}_{M-K}P_{N-K})^2)$.

Combinations and permutations can be calculated by making use of the little theorem of Fermat or extended Euclidean algorithm. By using the former, the total time complexity is $O(M + N)$.

F: Unfair Nim

(Editorial: evima)

As the title implies, this game is a famous one called Nim; it is known that the player who moves second has a strategy to always win if and only if $A_1 \oplus A_2 \oplus \ldots \oplus A_N = 0$ (where $\oplus$ denotes XOR: bitwise logical exclusive sum), and otherwise the player who moves first has a strategy to always win. When solving this problem, what you only have to know about Nim is this knowledge, so we won't explain in detail.

By above, let S be $A_1 + A_2$ and X be $A_3 \oplus A_4 \oplus \ldots \oplus A_N$, then the problem can be rephrased as follows:

> Judge if there exists a pair of non-negative integers (a, b) such that $a+b = S, a \oplus b = X, a \leq A_1$, and if so, find the maximum value of a.

(In the problem above, you are allowed to move all the stones for the first pile, which you are not in the original problem, but you can handle with it by answering "impossible" if the answer is 0.

One of the solutions is to rephrase the problem further as "find the maximum non-negative integer a such that $a+(a \oplus X) = S$, and use dynamic programming to virtually perform exhaustive search for all a.

However, since $a + b = (a \oplus b) + 2 \times (a \ \texttt{\&}\ b)$ (where `&` denotes AND: bitwise logical product. This formula can be justified by considering that XOR is operation of "calculating the sum while ignoring carryovers), you don't have to perform searching. For each (a, b) that satisfies the conditions, by the equation above, it holds that $a \ \texttt{\&}\ b = (S - X)/2$ (let this value be D). If D is not a non-negative integer, or D and X has one or more bit in common, then the answer is "impossible".

Otherwise, if we ignore the condition that $a \leq A_1$, then for all pairs of integers Y and Z which can be obtained by partitioning the set of bits 1 in X ($Y \ \texttt{\&}\ Z = 0, Y \oplus Z = X$), the pair $a = D \oplus Y, b = D \oplus Z$ satisfies the remaining constraints. Also, no other pair satisfies such conditions. Among each a of such pairs, the maximum a such that $a \leq A_1$ is the desired answer.

First, the minimum a is $a = D$, which can be obtained when $Y = 0$, and if this value does not satisfy $a \leq A_1$, then the answer is "impossible." Otherwise, we want to choose as large Y

as possible, subject to $a \leq A_1$, so that $a = D \oplus Y$ is maximum possible. Come to the point, the optimal is a greedy strategy of, first let $Y = 0$, then iterate each bit of X whose value is 1, checking if "after adding this bit to Y, does $a = D \oplus Y \leq A_1$ still holds?", and if the answer is Yes, then add it. This can be proven by definition of ordering of numbers (that any digit is significant that all lower digits combined). Therefore, the rephrased problem can be solved in a total of $\mathrm{O}(\log X)$ time.