NOMURA プログラミングコンテスト 2020 解説

gazelle, kobae964, satashun

2020 年 5 月 30 日

For International Readers: English editorial starts on page 8.

A: Study Scheduling

高橋君が起きている時間は $60(H_2 - H_1) + (M_2 - M_1)$ 分間です。寝る前に勉強が終わるためには、寝る時刻の K 分前までに勉強を始める必要があります。よって求める答えは $60(H_2 - H_1) + (M_2 - M_1) - K$ 分です。

Listing 1　解答例 (Python3)

```
import sys
readline = sys.stdin.buffer.readline

h1, m1, h2, m2, k = map(int, readline().split())
print(60 * (h2 - h1) + m2 - m1 - k)
```

B: Postdocs

T に含まれる ? を全て D で置き換えてできる文字列が，常に博士・PD 指数を最大にします。

というのも，T の ? を P で置き換えている箇所があったとします。その箇所を代わりに D で置き換えたとき，博士・PD 指数がどのように増減するかを考えます。すると，まず元々 P のあとに D が続いていた場合，連続部分文字列として PD が 1 個なくなるので博士・PD 指数は 1 減りますが，新しく D で置き換えたことにより連続部分文字列として D が 1 個増えるので，結果として博士・PD 指数は変わりません。元々 P のあとに P が続いていた場合は，博士・PD 指数は新しく増えた D の分 1 増えます。

よって，T に含まれる ? を全て D で置き換えてできる文字列が，その他の置き換えによってできる文字列より博士・PD 指数が小さくないことが分かります。

Listing 2　C++ による実装例

```
#include<bits/stdc++.h>
using namespace std;

int main() {
  string t;
  cin >> t;
  for(int i = 0; i < t.length(); i++) if(t[i] == '?') t[i] = 'D';
  cout << t << endl;
  return 0;
}
```

C: Folia[*1]

基本方針

$d = 0, 1, \ldots, N$ について、深さ d の頂点数の上界が以下の 2 通りの方法で定まります。

- 深さ $d-1$ の頂点のうち葉ではないものの個数の 2 倍
- $A_{d+1} + A_{d+2} + \cdots A_N$

この 2 種類を満たすように、深さ 0 の頂点からなるべく多くなるように頂点の個数を決めれば、最大頂点数が達成できそうです。以下ではそれを厳密に証明します。

証明

深さ d の頂点のうち葉ではないものの個数を B_d と置きます。全体で木になっているため、以下が成立します。

$$A_0 + B_0 = 1 \tag{1}$$

深さ $d+1$ の頂点は深さ d の葉でない頂点から辺が伸びているため、以下が成立します。

$$B_d \leq A_{d+1} + B_{d+1} \leq 2B_d \tag{2}$$

また、深さ d の葉でない頂点の子孫には必ず深さ $d+1$ 以上の葉が存在するため、以下が成立します。

$$B_d \leq A_{d+1} + A_{d+2} + \cdots + A_N \tag{3}$$

これらの条件を満たすように長さ $N+1$ の数列 $B_0, \ldots, B_N$ を先頭から貪欲に決めるのが最善です。これを示すために、まず以下の定義を行います。

定義. N, k を非負整数とする。非負整数列のペア $((A_i)_{i=0,\ldots,N}, (B_i)_{i=0,\ldots k})$ に深さ N 以下の二分木 T が対応するとは、以下の条件を満たすことをいう。

- $d = 0, \ldots, k$ に対して、T の深さ d の葉はちょうど A_d 個、T の葉以外の深さ d の頂点はちょうど B_d 個存在する

上の戦略の妥当性を示すには以下の命題を証明すれば良いです。

命題. k を $0 \leq k \leq N$ を満たす整数、$A_0, \ldots, A_N, B_0, \ldots, B_{k-1}, B_k, B'_0, \ldots, B'_{k-1}, B'_k$ を非負整数列とする。以下を仮定する。

- $((A_i)_{i=0,\ldots,k}, (B_i)_{i=0,\ldots,k})$ と $((A_i)_{i=0,\ldots,k}, (B'_i)_{i=0,\ldots,k})$ はともに (2) と (3) を満たす

[*1] folia はラテン語で「葉」を意味する単語 folium の複数形です。

- $B_k \leq B'_k$
- $((A_i)_{i=0,\ldots,N}, (B_i)_{i=0,\ldots,k})$ には対応する深さ N 以下の二分木が存在する
- $((A_0, \ldots, A_{k-1}, A_k + B'_k), (B'_0, \ldots, B'_{k-1}, 0))$ には対応する深さ k 以下の二分木が存在する

この場合、$((A_i)_{i=0,\ldots,N}, (B'_i)_{i=0,\ldots,k})$ には対応する深さ N 以下の二分木が存在する。

Proof. k の上の (逆順の) 数学的帰納法を用いる。$k = N$ のとき、(3) から $B_N = B'_N = 0$ である。よって仮定から、$((A_i)_{i=0,\ldots,N}, (B'_i)_{i=0,\ldots,N})$ に対応する二分木が存在することは自明である。

$0 \leq k < N$ のとき、仮定から存在が保証されている二分木の、葉でない頂点の個数を深さごとに $B_0, \ldots, B_N$ とし、$B'_{k+1} = \min(2B'_k - A_{k+1}, A_{k+2} + \cdots + A_N)$ と置く。$2B'_k - A_{k+1} \geq 2B_k - A_{k+1} \geq B_{k+1}$ かつ (3) より $B_{k+1} \leq A_{k+2} + \cdots + A_N$ であるから $B_{k+1} \leq B'_{k+1}$ である。$((A_i)_{i=0,\ldots,N}, (B'_i)_{i=0,\ldots,k+1})$ に対応する深さ N 以下の二分木が存在することを示す。

- 非負整数列のペア $((A_i)_{i=0,\ldots,k+1}, (B'_i)_{i=0,\ldots,k+1})$ は明らかに (2)、(3) を満たす。
- $B_{k+1} \leq B'_{k+1}$ は明らか
- $((A_i)_{i=0,\ldots,N}, (B_i)_{i=0,\ldots,k+1})$ に対応する二分木の存在は仮定されている
- $((A_0, \ldots, A_k, A_{k+1} + B'_{k+1}), (B'_0, \ldots, B'_k, 0))$ に対応する深さ $k+1$ 以下の二分木が存在することを示す。$((A_0, \ldots, A_{k-1}, A_k + B'_k), (B'_0, \ldots, B'_{k-1}, 0))$ に対応する深さ k 以下の二分木を一つ取り、それを T とする。T の深さ k の頂点は全て葉であるが、そのうち B'_k を葉でない頂点に変更し、その下に $A_{k+1} + B'_{k+1}$ 個の葉をつけた二分木が条件を満たすことを示す。この二分木が構成可能であれば $((A_0, \ldots, A_k, A_{k+1} + B'_{k+1}), (B'_0, \ldots, B'_k, 0))$ に対応することは明らか。構成可能であることを示す。$B_k \leq A_{k+1} + B'_{k+1} \leq 2B_k$ が満たされればよい。$A_{k+1} + B'_{k+1} \leq 2B_k$ は B'_{k+1} の作り方から明らか。$B_k \leq A_{k+1} + B'_{k+1}$ は $B_k \leq 2B'_k$ と $B_k \leq A_{k+1} + \cdots + A_N$ の双方が示されれば示されるが、前者は $B_k \leq B'_k$ という仮定から明らかで、後者は $((A_i)_{i=0,\ldots,k}, (B_i)_{i=0,\ldots,k})$ が (3) を満たすことから明らか。

以上より、帰納法の仮定から $((A_i)_{i=0,\ldots,N}, (B'_i)_{i=0,\ldots,k+1})$ に対応する深さ N 以下の二分木が存在する。この二分木が元の条件も満たす。 □

以上から、先頭から B_d を (1),(2) および (3) を満たすようになるべく大きくするのが最善で、証明の方法からこのやり方で全ての B_d が最大化できることも言えます。答えは最大で

$$\sum_{d=0}^{N}(A_d + B_d) \leq \sum_{d=0}^{N}(A_d + \cdots + A_N) \leq \sum_{d=0}^{N} 10^8 \times (N - d + 1) = \frac{10^8(N+1)(N+2)}{2} \simeq 5 \times 10^{17}$$

程度になり得るため、32bit 整数の範囲には収まりません。

D: Urban Planning

まず、全ての町について要請先が決まっている場合に必要な道の本数について考えましょう。(多重辺を許して) (i, P_i) に無向辺を張ったグラフを考えると、全域森を取れば各連結成分ごとに頂点数 -1 本の辺で条件を満たせることがわかり、また連結であるためには明らかにこれが下界です。

ここで、グラフの作り方より各連結成分については頂点数と辺数が等しく、(i, P_i) を有向辺と見た時閉路が 1 つずつ存在するという特徴があり、各閉路ごとに 1 本道が削減できるという見方をすると性質が良さそうです。削減できる道の数の和を求めて全体から引くことにして、次の問題を考えます。

> $P_1, P_2, \cdots, P_N$ のうちいくつかが -1 である。全ての要請方法について、有向辺 (i, P_i) を張ってできる (有向) 閉路の個数の和は何個であるか。

以下ではある閉路を構成する頂点を順番も含めて固定したときに何回カウントされるかという見方で考えていきます。

ひとまず既に決まっている辺を張ってしまうと、何個かの (弱) 連結成分に分かれます。ある連結成分に既に閉路がある場合、これは全ての -1 の決め方において数えられるので別で数えておきます。

未決定の頂点の行き先を決めたことで新しくできる閉路については、各連結成分について先端の頂点と (その頂点も含めて) 連結成分内のもう 1 頂点の間のパスが閉路の一部となる可能性があります。

-1 のみからなる孤立点もサイズ 1 の連結成分と見なしておくと楽に扱えます。

連結成分からサイズ $A_1, A_2, \cdots, A_k$ である k 個の連結成分を選び、これらのパスを好きな順番で繋げてできる閉路は以上の考察より $A_1 \times A_2 \times \cdots \times A_k \times (k-1)!$ 通りあり、この閉路上にない町の決め方とは独立です。 (残りの自由度も、k ごとに同じです)

よって連結成分数の配列を作っておき $dp_{i,j} := i$ 番目の連結成分までみて j 個選んだ時の係数の和と定義するとこの配列は $O(N^2)$ で計算できます。(実際は $(1 + A_1 x)(1 + A_2 x) \cdots (1 + A_n x)$ の x^k の係数です)

この情報を用いて ($k = 1$ の時は $P_i \neq i$ の条件を考慮する必要があることに注意してください) 答えを計算すると全体でも $O(N^2)$ で解くことができ、十分高速です。

E: Binary Programming

操作列を逆順に見ます。すると T に対して，

- 奇数番目の文字の総和を x に加算する
- 好きな文字を 1 個削除する

という操作を繰り返して空文字列にするとき，最終的な x の最大値はいくつか？ という問題になります。

まず，1 を削除するのは 0 を全て削除してからでいいです。というのも 0 がまだ残っているなら，1 を削除する代わりにその 1 に最も近い 0 を削除することで，1 になっている箇所がより多い文字列が得られるからです。

T が 0 を含まなくなってからの操作は自明なので，それまでの x を最大化する操作を考えます。T の中で 1 が偶数長連続している部分については，0 を削除するとき常にその半分が x に加算されます。よって無視して考えても問題ありません。同様に 1 が奇数長連続している部分については，1 つだけ残してあとを無視して構いません。すると，T に含まれる 1 は連続していないと仮定することができます。

T が k 個の 1 を含んでいるとして，T が，

$$(a_0\text{個の }0)\ 1\ (a_1\text{個の }0)\ 1\ \ldots\ 1\ (a_k\text{個の }0)$$

という形になっているとします。各 1 がどのタイミングで x に加算されうるかを考えると，i 個目の 1 は $\sum_{j=0}^{i-1} a_j$ の約半分（1 の位置の偶奇で変化）の 0 を削除するときに必ず x に加算され，$\sum_{j=i}^{k} a_j$ 個の 0 を削除するときに x に加算されうることが分かります。実はこの上界が達成可能で，

a_0個の 0 を全て削除 $\to$ a_1個の 0 を 1 個残してあとは削除 $\to \ldots \to$ a_k個の 0 を 1 個残してあとは削除

としたあとに，残っている 0 を右から順に削除していけばいいです。

F: Sorting Game

すぬけくんが可能な操作を操作 A，高橋くんが可能な操作を操作 B と呼ぶことにします。

操作 B を繰り返して数列 a を昇順に並び替えることができるための必要十分条件は，任意の $a_i > a_j$ $(i < j)$ について a_i, a_j が 2 進数でちょうど 1 桁だけ異なることです。必要性は $a_i > a_j$ $(i < j)$ となる a_i, a_j を必ずどこかでスワップする必要があることから分かります。十分性は最小要素から左に詰めていけば帰納法で示せます。以下この必要十分条件を条件 P と呼ぶことにします。

すぬけくんが操作 A を繰り返して 0 にする桁の組合せについては，「ある x 桁目以上の桁を全て選ぶ」パターンだけを考えれば十分です。というのも，このパターンで条件 P が成り立つと仮定した上で，ある桁の集合を 1 つ固定して，それらの桁を 0 にしたときに $a_i > a_j$ $(i < j)$ になっている a_i, a_j に注目します。a_i, a_j とで異なる最上位の桁が d 桁目であるとすると，$d + 1$ 桁目以上の桁を全て 0 にしたときに条件 P が（要素ペアレベルで）成り立つという仮定から，この場合も条件 P が（要素ペアレベルで）成り立つことが分かります。ある x 桁目以上の桁を全て 0 にするパターンの操作を行ってもなお条件 P が成り立つという性質を，性質 Q と呼ぶことにします。

2^i 未満の整数からなる長さ j の数列のうち，性質 Q を満たすものを考えます。そのような数列の 2 進数で i 桁目を左から見ていったときにどうなっているかを考えると，「1 のあとに 0 が現れない」「1 のあとに 0 が現れる部分がある」のいずれかです。いずれの場合も，性質 Q の定義を考えると，この数列は最上位桁を全て 0 にしてもなお性質 Q を満たしています。前者については他に i 桁目未満の桁について要請はありません。後者については，条件 P を考えると，最初に 1 になる要素および最後に 0 になる要素の間の要素は i 桁目を除いて等しいはずです。

以上の性質を踏まえて，動的計画法で解を計算することにします。2^i 未満の整数からなる長さ j の数列のうち，性質 Q を満たすものの個数を dp_{ij} とします。dp_{ij} の値を dp_{i-1*} の各値から計算することを考えます。上で述べた「1 のあとに 0 が現れない」パターンについては，$dp_{(i-1)j}$ に数えられている数列の，最初のいくつかの要素の i 桁目を 0 にして残りの i 桁目を 1 にしたものと対応付けられます。また「1 のあとに 0 が現れる部分がある」パターンについては，$k < j$ なる k について，$dp_{(i-1)k}$ に数えられている数列の適当な要素を選んで，それより左の要素は i 桁目を 0 に，それより右の要素は i 桁目を 1 にして，選んだ要素は $j - k$ 個増やして，その左端の i 桁目は 1 に，右端の i 桁目は 0 にして残りの i 桁目は任意として作れる数列と対応付けられます。j の小さい順にテーブルを埋めていけば全体で $O(NM)$ で計算できるので，dp_{NM} を答えとしてこの問題を解くことができました。

A: Study Scheduling

Takahashi is up for $60(H_2 - H_1) + (M_2 - M_1)$ minutes. To finish studying before he sleeps, he has to start it at least K minutes before the moment when he sleeps. Thus, the answer is $60(H_2 - H_1) + (M_2 - M_1) - K$ minutes.

Listing 3 Sample Implementation (Python3)

```
import sys
readline = sys.stdin.buffer.readline

h1, m1, h2, m2, k = map(int, readline().split())
print(60 * (h2 - h1) + m2 - m1 - k)
```

B: Postdocs

The string obtained by replacing every '?' with a D always maximizes the quotient.

We will prove this. Assume that one of the '?'s in T is replaced by a P. Let us consider how the quotient changes when we replace that '?' with a D instead. If a D followed the P, T loses one occurrence of PD but gains one occurrence of D, so the quotient does not change. If a P followed the P in question, the quotient increases by 1 from gaining one occurrence of D.

Thus, the quotient of the string obtained by replacing every '?' with a D is not smaller than those of strings obtained otherwise.

Listing 4 Sample Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

int main() {
 string t;
 cin >> t;
 for(int i = 0; i < t.length(); i++) if(t[i] == '?') t[i] = 'D';
 cout << t << endl;
 return 0;
}
```

C: Folia[*2]

Overview

For each $d = 0, 1, \ldots, N$, we can see the following upper bounds to the number of leaves at depth d:

- the number of non-leaf vertices at depth $d-1$, multiplied by 2
- $A_{d+1} + A_{d+2} + \cdots A_N$

It seems we can maximize the total number of vertices by maximizing the number of vertices at depth 0, at depth 1, and so on in this order. Below, we will formally prove this.

Proof

(Will be ready within a day.)

[*2] The Latin word for leaves (singular: folium).

D: Urban Planning

Will be ready within a day.

E: Binary Programming

Consider the sequence of operations in reverse order. Now the problem becomes the following: maximize the final value of x when we have a string T and make it empty by repeating the operations below:

- add the sum of the digits in the odd positions
- delete one character of your choice

First, we can assume that we only delete 1s after deleting all 0s. This is because if the string still has 0s, instead of deleting a 1, we can delete the 0 nearest to that 1 and obtain a string with more 1s.

What we can do when T no longer contains 0 is obvious, so let us consider how to maximize x before that. For a segment in T formed by an even number of consecutive 1s, x will always increase by half of that number, so we can ignore it. Similarly, for a segment in T formed by an odd number of consecutive 1s, we can just keep one of those 1s and ignore the rest. Now, we can assume that T does not contain consecutive 1s.

Assume that T contains k 1s, and T is in the following form:

$$(a_0\text{0s})\ 1\ (a_1\text{0s})\ 1\ \ldots\ 1\ (a_k\text{0s})$$

Let us consider when these 1s contribute to x. The i-th 1 is always contributed to x when approximately half (depending on the parity of the position of 1) of the $\sum_{j=0}^{i-1} a_j$ 0s are deleted, and is potentially contributed to x when the $\sum_{j=i}^{k} a_j$ 0s are deleted. This upper bound is actually achievable by doing as follows:

Delete all of the $a_0$0s $\to$ a_1Delete all but one of the $a_1$0s $\to \ldots \to$ a_kDelete all but one of the a_k0s

and then deleting the remaining 0s from right to left.

F: Sorting Game

Will be ready within a day.