ARC 088/ABC 083 解説

DEGwer

2017/12/23

A: Libra

整数 A, B, C, D を読み込み、$A + B$ と $C + D$ の大小比較をすればよいです。

```
#include<stdio.h>
int main()
{
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        if (a + b > c + d)printf("Left\n");
        else if (a + b == c + d)printf("Balanced\n");
        else printf("Right\n");
}
```

B: Some Sums

N 以下の整数の各桁の和を求め、適切に場合分けをして足し合わせてやればよいです。

```
#include<stdio.h>
int main()
{
        int n, a, b;
        scanf("%d%d%d", &n, &a, &b);
        int r = 0;
        for (int i = 1; i <= n; i++)
        {
                int c = 0, t = i;
                for (int j = 0; j < 5; j++)c += t % 10, t /= 10;
                if (a <= c&&c <= b)r += i;
        }
        printf("%d\n", r);
}
```

C: Multiple Gift

作る数列を $A_1, ..., A_{|A|}$ とします。条件より $X \leq A_1$ であり、またすべての i に対し $2A_i \leq A_{i+1}$ となります。よって、$A_i \geq X \times 2^{i-1}$ がわかります。

逆に、$A_i = X \times 2^{i-1}$ とすれば、これは条件を満たします。すべての i に対し A_i の最小値を達成できているので、条件を満たす数列のうちこのようにして作られるもののみを考慮すればよく、時間計算量 $O(\log \frac{B}{A})$ でこの問題を解くことができます。

D: Wide Flip

与えられる文字列の長さを n とします。

k 文字目と $k+1$ 文字目が異なるとします。このとき、すべての文字を 0 にするためには、k 文字目と $k+1$ 文字目の片方のみを書き換えるような操作を 1 回以上行わなければなりません。すなわち、$max(k, n-k)$ 文字以下の連続する部分を書き換えるような操作が必要となります。

S の、k 文字目と $k+1$ 文字目が異なるような箇所のうちの、$max(k, n-k)$ の最小値を $T(S)$ とします。$K \geq T(S)$ のとき、T の値の最小値をとる k をとり、$k = max(k, n-k)$ なら最初 k 文字を、そうでないなら最後 $n-k$ 文字を書きかえれば、連続する文字が異なるような箇所の個数を、$T(S)$ を減らさずに 1 減らすことができます。また、$T(S) = 0$ のとき、もし全文字が 0 ならこれが目的の文字列であり、そうでないなら全体に対して操作を行えば目的の文字列を作ることができます。

よって答えは最初の文字列 S における $T(S)$ の値で、これは $O(n)$ 時間で求められます。

E: Papple Sort

まず、奇数回しか出てこない文字が 2 つ以上ある場合、回文を作ることはできません。以下、それ以外の場合を考えます。

最終的に (最小の交換回数で) 作る回文を T とし、S で添え字 i の文字が T では添え字 p_i となるとします。

同じ文字同士を入れ替えるのは無意味なので、各文字に対してその文字の出現回数を求めれば、文字列中のすべての添え字の ($|S|$ 個の) 文字に対し、その文字が最終的に T の左半分に属するか、ちょうど真ん中の文字となるか、右半分に属するかは一意に定めることができます。

上述の 3 つの添え字集合をそれぞれ A, B, C とします。A の添え字の文字を順序を保ったまま左半分に、B の添え字の文字を真ん中に、C の添え字の文字を順序を保ったまま右半分に移動して作った文字列を X とします。このとき、これら 3 種類の添え字の文字同士の swap が起こりますが、T に関する上述の条件より、$i < j$ かつ $p_i > p_j$ となるような文字 S_i, S_j の swap のみが起こることがわかります。よって (バブルソートの交換回数とはそのような添え字 (i, j) の組の個数に他ならないので) S から T への変換での交換回数は、S から X への変換での交換回数と X から T への変換での交換回数の合計と等しくなることが分かります。

さて、X から T に変換するとき、A 内の要素同士、または C 内の要素同士の swap のみが起こります。もし A 内の要素同士の swap が起きる場合、その代わりに C 内での対応する要素同士を swap することにしても、最終的な文字列が回文になるという条件は依然満たされます。よって、A 内の要素同士は swap しないとして問題ありません。

A 内の要素の順番が定まれば C 内の要素の順番も定まるので、結局 p_i の値がすべての i に対して一意に定められたことになります。あとは、順列 p のバブルソートの交換回数を、BIT などを用いて数えてやればよいです。時間計算量は $O(|S| \log |S|)$ です。

F: Christmas Tree

この問題は、以下の条件を満たす辞書順最小の (A, B) を求める問題だと言い換えることができます。

- 木の辺たちを A 色で塗り分ける。このとき、同じ色が塗られた辺の集合は、長さ B 以下のパスをなしている必要がある。

まず、A の最小値を求めましょう。次数奇数の頂点に対し、その点を端点とするようなパスが 1 つ以上必要です。逆に、次数奇数の頂点間のパスをとって取り除いていくことを考えれば、次数奇数の頂点の個数の半分の個数のパスがあれば十分であることもわかります。よって、A の最小値は、次数奇数の頂点の個数の半分です。

B の最小値を求めるために、二分探索を考えましょう。B の値を固定し、そのような塗りわけが存在するかどうかを求める DP を考えましょう。

まず、次数 1 の頂点を根として入力の木を根付き木にします。便宜上、根からも上向きに辺が出ているものとして扱います。各頂点 v に対し、DP$[v]$ を、v を根とする部分木に属するパスの長さがすべて B 以下となるように塗り分けるときの、v から親方向にのびるパスの長さの最小値 とします。

この DP 値はどのように求めればよいでしょうか。v の子を $u_1, ..., u_k$ とし、DP$[u_i]$ の小さい順に並べ替え、改めて添え字を ($u_1, ..., u_k$ と) つけなおしておきます。条件より、

- k が奇数のとき、u_i たちから親方向に伸びるパスからペアを $\frac{k-1}{2}$ 個作ってつなぎ合わせ、残ったパスを v から親方向に伸ばすことにする
- k が偶数の時、u_i たちから親方向に伸びるパスからからペアを $\frac{k}{2}$ 個作ってつなぎ合わせ、v から新しいパスを親方向に伸ばすことにする、またはペアを $\frac{k}{2}-1$ 個作ってつなぎ合わせ、残った 2 本のパスのうち 1 本の端点を v にし、もう 1 本のパスを v から親方向に伸ばすことにする

する必要があります。

パスたちをペアにするときは、各ペアについてその合計長さが B を超えないようにする必要があります。すなわち、合計の長さの最大値を最小にするようなペアの組み方を考え、そのときの最大長が B 以下かどうか判定すればよいです。このとき、greedy な考察により、$2m$ 本のパスを m 個のペアに分けるとき、すべての x に対し、x 番目に長いパスと x 番目に短いパスを組み合わせるのが最適であることがわかるので、この判定は $O(m)$ 時間で行うことができます。

上の場合分けでの前者の場合、1 本のパスを選んで取り除いた後に、パスをペアにしていく必要があります。取り除くパスは、DP 値の定義より、なるべく短いものを選びたいです。これは、「どのパスを取り除くべきか」で二分探索を行うことで最適化することができます。

後者の場合、ペアを $\frac{k}{2}$ 個作る場合は上述のアルゴリズムが動作します。そうでない場合、端点を v にするパスは最も長いパスを選ぶのが良いです。このように選べば、あとは k が奇数の場合と同じ問題に帰着されます。

以上より、子を k 個もつ頂点での DP 値の更新が $O(k \log k)$ 時間でできることが分かりました。最後に、(次数 1 の頂点を根に選んだことに注意すれば) 根から親方向に伸びるパスの長さを評価することで外側の二分探索の評価関数が正しく動作し、よって全体で $O(N \log^2 N)$ 時間でこの問題を解くことができました。

ARC 088/ABC 083 Editorial

DEGwer

2017/12/23

A: Libra

```
#include<stdio.h>
int main()
{
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        if (a + b > c + d)printf("Left\n");
        else if (a + b == c + d)printf("Balanced\n");
        else printf("Right\n");
}
```

B: Some Sums

```
#include<stdio.h>
int main()
{
        int n, a, b;
        scanf("%d%d%d", &n, &a, &b);
        int r = 0;
        for (int i = 1; i <= n; i++)
        {
                int c = 0, t = i;
                for (int j = 0; j < 5; j++)c += t % 10, t /= 10;
                if (a <= c&&c <= b)r += i;
        }
        printf("%d\n", r);
}
```

C: Multiple Gift

Let $A_1, ..., A_{|A|}$ be the sequence. $X \leq A_1$ must be satisfied, and for each i, $2A_i \leq A_{i+1}$ must be satisfied. Thus, we get $A_i \geq X \times 2^{i-1}$.

Therefore, the maximum length can be achieved when $A_i = X \times 2^{i-1}$. (For all i, the lower bound of A_i is achieved by this sequence.)

This solution works in $O(\log \frac{B}{A})$ time.

D: Wide Flip

Let n be the length of the string.

If the k-th and the $k+1$-th (1-based) characters of the string are different, in order to achieve the goal, we must perform at least one operation that flips exactly one of the two characters. In this operation, we flip at most $max(k, n-k)$ consecutive characters.

Let $T(S)$ be the minimum vaule of $max(k, n-k)$ such that the k-th and the $k+1$-th characters of S are different. As we see above, K must be at most $T(S)$.

On the other hand, we can prove that when $K = T(S)$, we can achieve the goal.

- If $i <= N - K$, we can flip a single character S_i by flipping $[i, N]$ and $[i+1, N]$.
- Similarly, if $i > K$, we can flip a single character S_i.
- By the definition of $T(S)$, all characters between the $N-K+1$-th and the K-th are the same.

Thus, by repeating the first two types of operations, we can make all characters equal, and (possibly by performing the third operation once) we can achieve the goal.

Therefore, the answer is $T(S)$, and it can be computed in $O(n)$ time.

E: Papple Sort

If two or more letters appear odd number of times, we can't make a palindrome. For simplicity, suppose that each letter appears even number of times (when a letter appears odd number of times, we know which letter will go to the center of the final string, and the solution doesn't change a lot.)

Let n be the length of the string. Let T be the final palindrome we make (by performing the minimum possible number of operations), and let T_{p_i} be the destination of S_i.

Consider a particular letter c, and suppose that it appears as i_1-th, i_2-th, ..., i_{2k}-th letters of S. Since it makes no sense to swap the same letters, the relative positions of these letters won't change. Thus, we know that i_1 and i_{2k} will be paired in T (i.e., $p_{i_1} + p_{i_{2k}} = N + 1$), i_2 and i_{2k-1} will be paired, and so on. Therefore, we know which characters will be paired in T, and we can assume that S contains $N/2$ different types of characters, and each character appears exactly twice.

Now, consider two particular types of letters in S (call it 'A', 'B').

There are six possibilities for their relative positions. For example, they can appear as

...A...A...B...B...

If the pair of A becomes "outer" pair and the pair of B becomes "inner" pair in T, we need two swaps among these characters. If the pair of B becomes "outer" pair and the pair of A becomes "inner" pair in T, we need two swaps among these characters. Thus, it doesn't matter.

By similar observations, it turns out that when they appear as

...A...B...B...A...

A should be "outer" and B should be "inner", and if they appear as

...B...A...A...B...

B should be "outer" and A should be "inner". In other cases it doesn't matter.

Thus, when we sort the pairs, we should make sure that these properties hold. For example, when we compare two pairs at positions (l_1, r_1) and (l_2, r_2), we make (l_1, r_1) "outer" if $l_1 < l_2$.

Now, we know which characters in A will go to which position in T (i.e. we know the values of p_i), and we can compute the inversion number in $O(|S| \log |S|)$.

F: Christmas Tree

We want to paint the edges of the tree using A colors. The set of edges of a specific color must form a path of length at most B.

What is the minimum possible value of A? Suppose that the tree has O vertices with odd degree. Since an odd vertex must be an endpoint of at least one path, we need at least $O/2$ paths. On the other hand, by repeatedly removing a path that connects two odd vertices, we can achieve $A = O/2$.

Now, we want to minimize B under the condition that $A = O/2$. Let's do a binary search on B. Now we want to check if a valid coloring exists for a fixed B.

Choose an arbitrary leaf of the tree, and make it the root. For convenience, we attach an edge to the root (an edge between the root and "the root's parent").

Let's define $DP[v]$ as follows. Consider a subtree rooted at v, plus an edge between v and v's parent (we call it e_v). We want to color each edge inside this subtree, such that

- The length of each color must be at most B (of course, each color must form a path).
- We should minimize the number of paths: each odd vertex is an endpoint of exactly one path, and each even vertex is an endpoint of no path.
- Under these conditions, we want to minimize the length of the color that contains e_v. Let $DP[v]$ be this length.

How do we compute $DP[v]$? Suppose that a vertex v has k children $u_1, ..., u_k$, and assume that $DP[u_1] \leq DP[u_2] \leq ....$

If k is odd, we need to make $\frac{k-1}{2}$ pairs among the children, and pair the only remaining child (call it u_r) with e_v. Obviously, we want to minimize r. Let's do a binary search again on r. For a fixed r, we want to check if the values $DP[u_1], \ldots, DP[u_{r-1}], DP[u_{r+1}], \ldots, DP[u_k]$ can be divided into pairs whose sums are at most B. This can be done greedily by pairing the x-th largest value and the x-th smallest value. Then, $DP[v] = DP[u_r] + 1$.

If k is even, add an imaginary child u_0 such that $DP[u_0] = 0$, and it reduces to the case with odd k.

If we don't get contradictions and $dp[root] \leq B + 1$, the chosen value of B is valid.

Since we need $O(k \log k)$ time to handle a vertex with k children, we need $O(N \log N)$ per step, and in total this solution works in $O(N \log^2 N)$ time.