ABC 050 / ARC 066 解説

writer : maroonrk

2016 年 12 月 17 日

A : Addition and Subtraction Easy

整数、文字列、整数を空白区切りで読み込み、読んだ文字列が “+” か “−” で条件分岐をします。

C++ のコード例

```
int a,b;
string op;
cin>> a >> op >> b;
if(op == "+")
        cout << a + b << endl;
else
        cout << a - b << endl;
```

B : Contest with Drinks Easy

それぞれの問題にかかる時間を配列に保存しておきます。ドリンクごとに、配列を一時的に変更し、その総和を求めて出力すればよいです。なお、予めかかる時間の総和を求めておいて、ドリンクが与えられた時に差分だけ計算する方法だと、より高速に答えが求まります。

C : Lining Up

N が偶数である場合を考えます。N 人が並んだ場合、「自分の左に並んでいた人数と自分の右に並んでいた人数の差の絶対値」は、$N-1, N-3, N-5, ...3, 1, 1, 3, ...N-5, N-3, N-1$ となるはずです。この配列を B_i とします。

するとこの問題は、A_i を並び替えて B_i にする方法は何通りか、という問題になります。まず、A_i と B_i で、ある値の登場する回数が等しくなかった場合、明らかに答えは 0 です。

では、等しい場合はどうなるでしょうか。この配列には、N/2 種類の値が、それぞれ 2 回ずつ登場しているはずです。それぞれの値について、並び替える方法は、「A_i で前にあったものが B_i で前にある」と、「A_i で前にあったものが B_i で後ろにある」の 2 通り考えられます。

$N/2$ 種類の値で独立に並び替えがあるので、求める答えは $2^{(N/2)}$ になります。N が奇数である場合も同様

に解けますが、その場合 Ai や Bi の中に 0 という値が 1 度だけ登場することに注意してください。

D : Xor Sum

説明のため、and をビットごとの論理積、$<<$ を左シフト演算、$>>$ を右シフト演算とします。

方針としては、a と b のビットを上から決めていく桁 DP を考えます。$u = a\ xor\ b$ としている限り、$0 \leq u$ は常に成り立ちます。また、$v = a + b = ((a\ and\ b) << 1) + (a\ xor\ b) = ((a\ and\ b) << 1) + u \geq u$ なので、$v \leq N$ という条件のみに注意すればよいです。そこで、次のような DP を考えます。

$dp[i][j] = a$ と b の i ビット目以上が決定していて、その段階で $v = j$ となる通り数。

この DP では、以下の 3 パターンの遷移が考えられます。

・a の i ビット目と b の i ビット目が共に 1 の場合

・a の i ビット目と b の i ビット目のどちらか片方のみが 1 の場合

・a の i ビット目と b の i ビット目が共に 0 の場合

しかし、このままではこの DP は $O(NlogN)$ となってとても間に合いません。そこで、この DP を少し修正して、以下のようにしてみます。

$dp[i][j] = a$ と b の i ビット目以上が決定していて、その段階で $(v >> i) = (N >> i) - j$ となる通り数。

これは一見、先程より複雑になっただけで、計算量が改善していないように見えます。しかし、$j \geq 2$ のパターンは、その後どのように a, b のビットを選んでも、v が N を超えることが起きません。なので、それらのパターンを全て $j = 2$ として扱ってよくなります。よって j の範囲は 0〜2 だけを考えればよくなり、この DP が $O(logN)$ で行えるようになりました。

E : Addition and Subtraction Hard

簡単のため、最初の数の前に “+” という記号があるものとして扱います。まず、“+” のあとで開き括弧を挿入しても意味がないことがわかります。また、“−” のあとに開き括弧を必ず挿入しても問題ありません。なぜなら、そこで開いた括弧を、次の数の直後で閉じれば、括弧をつけていないのと変わらないからです。また、同じ箇所に開き括弧を 2 個以上挿入するのも無意味です。ここで、次のような DP を考えます。

$dp[i][j] = i$ 項目の直後で、j 個括弧が開いている時の、そこまでの式の値の最大値。

これをどのように更新すればいいのでしょうか。上の議論より、開き括弧の直前には必ず “−” があるはずです。つまり、x 個括弧が開いている中にある数は、最終的に $(-1)^x$ 倍された値として計算されるということです。これがわかれば更新するのは簡単です。この DP で答えが求まりましたが、これでは $O(N^2)$ となり、時間制限に間に合いません。

しかし実は、括弧は 3 個以上開く必要がありません。なぜなら、括弧を 2 個開いたあとは、“−” の前で必ず一つ括弧を閉じるようにすることで、残りの数を全て “+” として扱うことができるからです。よって j の範囲は 0〜2 だけを考えればよくなり、この DP が $O(N)$ で行えるようになりました。

F : Contest with Drinks Hard

最初に、ドリンクがない状態で、最大スコアを求めましょう。この問題は、

「いくつか区間を選んで、

$\sum_{\text{選んだ区間}}$(区間の長さ) $*$ (区間の長さ $+1$)$/2-$区間に含まれる問題にかかる時間の合計
を最大化せよ」
と言い換えることができます。まず、次のような DP を考えます。
$dp[i]=$問題 i まで考えた時の最大スコア。
この DP の遷移は、以下のようになります。
$dp[i]=\max(\max_{\{0\leq j<i\}}\{dp[j]+(i-j)*(i-j+1)/2-(sum(i)-sum(j))\},dp[i-1])$
ここで、$sum(x)$ は、問題 1〜x までを解くのにかかる時間の総和です。さらに、式を整理すると、以下のようになります。
$dp[i]=\max(\max_{\{0\leq j<i\}}\{-j*i+j*(j-1)/2+dp[j]+sum(j)\}+i*(i+1)/2-sum(i),dp[i-1])$
これは、内側の max の中が、i に関する 1 次式になっているので、ConvexHullTrick を使って高速化することができます。よって、この DP は $O(N)$ で計算することができます。
では、クエリに答えるにはどうすればいいのでしょうか。そのためには、各問題について、それを解く時の最大スコア、解かない時の最大スコアを前計算しておけばよいです。
では、前計算はどうすればいいのでしょうか。上記の DP を、左から進めていったものと、右から進めていったものを用意すると、ある問題を解かない時の最大スコアはすぐに求まります。
次に、ある問題を解くときの最大スコアについて考えます。まず、$M=N/2$ として、問題 $i(i\geq M)$ と M を含む区間を解くときの最大スコアを考えます。これは、さっきの DP を少し変えると求まり、先ほどと同様に ConvexHullTrick で $O(N)$ で求めることができます。同様に、問題 $i(i\leq M-1)$ と M を含む区間を解くときの最大スコアも求められます。すると、ほかに考えるべきものは、問題 $i(1<M)$ を含んで、M を含まない区間と、問題 $i(M<i)$ を含んで、M を含まない区間だけです。問題を、M の左の部分と右の部分に分けることができたので、分割統治をすると良いとわかります。分割統治の 1 ステップが $O(N)$ なので、合計 $O(NlogN)$ で求めることが出来ました。