全国統一プログラミング王決定戦本戦 解説

writer : maroonrk

平成 31 年 2 月 17 日

1 A : Abundant Resources

区間の左端 i を固定して、右端 j を一つずつ伸ばしていくと、各 (i, j) について、区間 $[i, j]$ の総和を求められます。あとはそれぞれの (i, j) について、長さ $j - i$ に対する答えを更新すればよいです。以上でこの問題は $O(N^2)$ で解けます。

回答例

B : Big Integers

K 進数の数の大小を比較する問題です。

$N \neq M$ のときは、N と M の大小関係から答えがわかります。そうでない場合は、$(A_1, A_2, \cdots, A_N)$ と $(B_1, B_2, \cdots, B_N)$ を辞書順で比較すればよいです。よってこの問題は $O(N+M)$ で解けます。

回答例

C : Come Together

行方向の移動と列方向の移動を完全に分離して考えることができます。まず、すべての駒を同じ行に集めるために必要な最小の操作回数を求めます。同様にすべての駒を同じ列に集めるために必要な最小の操作回数を求めれば、求めた 2 つの値の和が答えです。

すべての駒を同じ行に集める際には、すべての駒の行番号の中央値に集まるようにすれば良いです。すべての駒の行番号を列挙すると時間がかかりすぎますが、W_i を i 行目にある駒の数として、数列 $W_1, W_2, \cdots, W_R$ から中央値を求めることができます。これは、駒の個数の半分を H として、累積和が始めて H 以上になる場所が中央値になることから求められます。

以上でこの問題は $O(H + W + K)$ で解けます。

D : Deforestation

ある竹が最後に伐採されたのが時刻 x だとすれば、それ以前に伐採されたタイミングに関係なく、その竹を伐採することで得られた得点の総和は x です。よって、各竹について、最後に伐採された時刻がわかれば良いです。

各イベントにおいて、そのイベントで伐採される竹であって、以降のイベントで伐採されないものを列挙することができればよいです。伐採するイベントを時刻の逆順に見ます。整数の集合を持って、イベントごとに、伐採される区間に属する整数を列挙し、それを集合から消す、という操作ができれば良いです。これは、例えば std::set を用いて高速に実装することができます。よってこの問題は $O(NlogN)$ で解けます。

E : Erasure

問題は、端点が 0 以上 N 以下の整数で長さが $K+1$ 以上の区間を用いて $[0,N]$ を被覆する方法は何通りか、ということです。DP をします。

$dp[i][j]$ = 左端が i 以下の区間のみを用いて、$[0,j]$ を被覆しており、$j+1$ は被覆されていないようにする方法の数

とします。

次に遷移を考えます。まず、左端が $i+1$ の区間を使用しない場合の遷移は、$dp[i+1][j]+=dp[i][j]$ です。次に、左端が $i+1$ の区間を使用する場合を考えます。左端が $i+1$ の区間の右端の値の最大値が L $((i+1)+(K+1) \leq L)$ だとします。このとき、左端が $i+1$ の区間の選び方は、$2^{L-(i+1)-(K+1)}$ 通りです。よって遷移は、$dp[i+1][max(L,j)]+=dp[i][j] \times 2^{L-(i+1)-(K+1)}$ となります。

このままでは各 i,j に対して各 L を試す必要があり、$O(N^3)$ 時間かかります。ですが、各 L に対して遷移を行う操作でボトルネックになるのは、数列のある範囲に対して、$w, w\times 2, w\times 2^2, \cdots$ と足し込む操作です。これは、累積和と同じように処理できます。（累積和との違いは、直前の値を 2 倍して足し合わせることです。）よってこの問題は $O(N^2)$ で解けます。

F : Flights

S から T へ向かうのも T から S に向かうのも同じなので、一般性を失わずに $Y_S \geq Y_T$ とします。

$X_T \leq X_S$ のパターンを考えます。このとき、最適な動きは以下のいずれかです。

- S から T へ直接向かう。コスト C_S がかかる。
- $X_S \leq X_w, Y_S \leq Y_w$ を満たす w を選び、$S \to w \to T$ と向かう。コスト $2C_w$ かかる。

これ以外の動きが、上記のいずれかの形より良くならないことはすぐにわかります。このパターンは明らかに $O(N)$ で解けます。

$X_S \leq X_T$ のパターンを考えます。最適な移動において、$X_i \leq X_j \leq X_k, Y_i \leq Y_j \leq Y_k$ を満たす i,j,k について、$i \to j \to k$ または $k \to j \to i$ となる動きは存在しません。なぜなら、i と k の間を直接移動したほうが明らかにコストが小さいからです。$a \to b$ という動きがあって、$X_a \leq X_b, Y_a \leq Y_b$ を満たすとき、b を、右上の点、と呼ぶことにします。ここで、最適な移動において登場する右上の点は、x 座標昇順で登場することが言えます。x 座標昇順に違反する最適解があったとすると、うまく組み替えるとよりコストの小さい経路が得られることからこれはわかります。

右上の点が x 座標昇順という形のみを考えて、S からの各点への最小コストを求めることにします。点 i への最小コストを $D(i)$ と書くことにします。到達不能な i については $D(i) = \infty$ としておきます。より具体的には、x 座標がある値以下の点のみを使ったときの最小コストを考え、x 座標の上限を段階的に増加させます。なお、各点の x 座標を y 座標 $\times eps$ だけ増加させて考えることで、全点の x 座標が異なるようにしておきます。まず、$X_i \leq X_s, Y_i \geq Y_S$ となる点 i を消去します。なぜなら、そのような点 i を使うと、右上の点が x 座標昇順のもとで T にたどり着けないからです。まず、$X_i \leq X_S$　をみたす i について、x 座標の上限 X_S での点 i への最小コストを求めます。明らかに、$i \neq S$ ならば $D(i) = C_S$ で、$i = S$ なら $D(i) = 0$ です。次に、現在の x 座標の上限よりも x 座標の大きい頂点のうち、x 座標最小のものを点 w とします。x 座標の上限を X_w にすることを考えます。まず、$D(w) = min\{D(i) + C_w : X_i \leq X_w, Y_i \leq Y_w\}$ となります。点 w 以外の最小コストを更新することを考えます。当然 w を使う経路のみを考えて更新すれば良いですが、そのような経路で w は必ず、右上の点になっています。右上の点の x 座標が昇順なので、考えるべきは、w から一度だけ左下へ向かうような動きのみです。つまり、$X_i \leq X_w, Y_i \leq Y_w$ をみたす i について、$D(i) = min(D(i), D(w) + C_w)$ とすれば良いです。

x 座標の昇順で頂点を追加していることから、$X_i \leq X_w$ の条件は、今まで追加した頂点である、と言い換えられます。よって、必要な操作は、y 座標がある区間にある点の最小コストの min をとる、y 座標がある区間にある点の最小コストを与えられた値で min をとって更新する、y 座標と最小コストが与えられた点を追加する、の 3 つです。これは、Segment Tree を用いて実装することができます。しかし、操作を行う区間が必ず $y \leq hoge$ の形でかけることから、実はこれは std::set を用いて実装することもできます。どちらにせよ、1 回の操作が $O(logN)$ で行えるため、全体で $O(NlogN)$ でこの問題は解けます、

G : Greatest Journey

最適な移動の方法は、ある頂点まで最短距離で移動し、その後同じ辺を通り続けることです。

重心分解をします。重心 G を取って、解の形を G を通る経路に限定して問題を解きます。頂点 V_i から G へ向かう部分は自明なので、G から始めて d 回動いたときに得られる得点の総和の最大値はいくらか、という問題を様々な d に対して求めれば良いです。

G を根にした根付き木を考え、頂点 v の親を $p(v)$ とします。解の形は、G からある頂点 $v(v \neq G)$ へ向かい、その後辺 $p(v)-v$ を往復し続ける形として良いです。辺 $p(v)-v$ の得点を A、$G-V$ パスの長さを L、$G-V$ パスの得点の総和を B とすれば、G から d 回動く場合の得点は、$A \times (d-L)+B = A \times d + (B - A \times L)$ となります。これは d の一次式になっています。この一次式を以後 $f_v(d)$ と呼びます。ここで、$d < L$ の場合、頂点 v にそもそもたどり着けないため、頂点 v を使うパターンは考えられません。よって、d が与えられたときには、$L \leq d$ を満たす v について、$f_v(d)$ の最大値を求めれば良いです。ところで、辺 $p(v)-v$ の得点が、パス $G-v$ 上の辺の得点の最大値でない場合、v を使うパターンは考える必要がありません。そして、そのような v を除外すると、$d < L$ の場合に $f_v(d)$ を評価しても、実際に可能なものよりも良い値になることはありません。なので、$d < L$ の制約を考える必要がなくなり、単に $f_v(d)$ を最大化することだけを考えれば良いです。そしてこの問題は Convex Hull Trick を使うと解くことができます。辺を重みでソートするのに $O(NlogN)$、クエリを d でソートするのに $O(QlogQ)$ かかるので、重心を一つ決めたときの処理は $O(NlogN+QlogQ)$ で可能です。よって、全体でこの問題は $O((NlogN+QlogQ)logN)$ で解けます。

H : Homework Scheduling

一般性を失わず、$A_1 \leq A_2 \leq \cdots \leq A_N$ とします。

問題を最小費用流を用いて定式化します。

まず、$2N+3$ 個の頂点を用意し、S, $P_1, P_2, \cdots, P_N$, $Q_1, Q_2, \cdots, Q_{N+1}$, T とおきます。次に、各 i $(1 \leq i \leq N)$ に対して、以下のように辺を張ります。これらの辺は宿題 i に対応する辺です。

- S から P_i へ向かう、容量 1、コスト 0 の辺を張る。
- P_i から Q_i へ向かう、容量 1、コスト $-X_i$ の辺を張る。
- P_i から Q_{N+1} へ向かう、容量 1、コスト $-Y_i$ の辺を張る。

また、各 i $(1 \leq i \leq N)$ に対して、以下のように辺を張ります。これらの辺は Day i に対応する辺です。

- $i \leq A_j$ となる最小の j をとる。そのような j が存在しない場合は $j = N+1$ とする。Q_j から T へ向かう、容量 1、コスト 0 の辺を張る。

さらに、各 i $(1 \leq i \leq N)$ に対して、以下のように辺を張ります。

- Q_{i+1} から Q_i へ向かう、容量 ∞、コスト 0 の辺を張る。

このグラフ上で、S から T へ流量 k の最小費用流を流すと、そのコストは Day k までに得られる得点の総和の最大値の符号を反転させたものになっています。そこで、このグラフの最小費用流を高速に求めることを考えます。

今、流量 k の最小費用流が求まっているとします。ここで、現在求まっている残余グラフが、次の条件を満たすと仮定します。

- 辺 $Q_i \to T$ の容量が正であるすべての i に対して、辺 $Q_{i-1} \to Q_i$ の容量が 0 である。

この仮定のもとで、流量を 1 増やします。上記の仮定より、残余グラフ上の $S \to T$ 最短路は、次のいずれかの形をしていると考えて良いです。

- $S \to P_i \to Q_i \to Q_j \to T$
  ただし j は、$Q_j \to T$ の容量が正である j であって、$j \leq i$ を満たす中での最大値。
- $S \to P_i \to Q_{N+1} \to Q_j \to T$
  ただし j は、$Q_j \to T$ の容量が正である j の最大値。
- $S \to P_i \to Q_i \to Q_j \to P_j \to Q_{N+1} \to Q_h \to T$
  ただし h は、$Q_h \to T$ の容量が正である h の最大値。

詳細な証明は省略しますが、これは次のような方針で示すことができます。まず、現在求まっているのが容量 k の最小費用流なので、残余グラフに負閉路は存在しません。ゆえに、同じ頂点を複数回通るパスは考えなくてよいです。とくに、Q_{N+1} を通る回数は高々 1 回です。通る回数、及びそこに至るまでの経路で場合分けをして、仮定した条件と $X_i > Y_i$ を使うと、上記のパターンを考えれば十分であることがわかります。

このいずれのパスに沿って流量を 1 増やしても、上記で仮定した条件は保たれたままになります。$k = 0$ のときに明らかに仮定した条件は成立しているので、上記のいずれかの形のパスに沿って流量を増やし続けることにすれば、仮定した条件は常に成り立っているとしてよいです。

あとは、上記の形のパスの最短路を高速に求められればよいです。最初の 2 個の形は簡単に求められます。$Q_j \to T$ の容量が正の j を set 等で管理しながら、$P_i \to Q_i$ または $P_i \to Q_{N+1}$ のコストが最小のもの（ただし増加パスが存在するものに限る）使うようにすればよいです。

最後の形のパスは、次のような問題が高速に解ければ最短路を高速に求められます。

- 配列 C, D, E を用意する。C は、$S \to P_i$ の容量が正の i に対しては、$C_i = -X_i$、それ以外の i に対しては $C_i = \infty$ となる配列。D は、$S \to P_i \to Q_i$ に沿って 1 の流量がある i に対しては $X_i - Y_i$、それ以外の i に対しては $D_i = \infty$ となる配列。E は、$E_i = (Q_i \to Q_{i+1}$ の容量) となる配列。
- C の 1 要素を変更するクエリを処理する。
- D の 1 要素を変更するクエリを処理する。
- E の ある区間に $+1$ または -1 を加算するクエリを処理する。
- $i < j$ の組であって、$D_i + C_j$ が最小となるものを答える。
- $i < j$ の組であって、$E_i, E_{i+1}, \cdots, E_{j-1}$ がすべて正であるようなものの中で、$C_i + D_j$ が最小となるものを答える。

この問題は Segment Tree を用いることで高速に解けます。具体的には、Segment Tree の各ノードに、以下の情報を持たせると、ノードのマージが可能で、かつ必要な情報を答えることができます。

- ノードが対応する区間を半開区間 $[l, r)$ とする。
- E の $[l, r)$ 全体に足すべき値
- E の $[l, r)$ 内の最小値
- E の $[l, r)$ 内の最小値を m、最小値ををとる最大の index を u、最小の index を v とおく。
- C の $[l, u]$ 内の最小値
- C の $[u + 1, r)$ 内の最小値
- D の $[l, v]$ 内の最小値
- D の $[v + 1, r)$ 内の最小値
- $[l, r)$ 内の $i < j$ であって、$D_i + C_j$ が最小のもの
- $[l, r)$ 内の $i < j$ であって、$[i, j)$ 内の E の最小値が m であるようなもののうち、$C_i + D_j$ が最小のもの
- $[l, r)$ 内の $i < j$ であって、$[i, j)$ 内の E の最小値が m でないようなもののうち、$C_i + D_j$ が最小のもの

E の要素が 0 を下回らないことから、上記の情報から正しく答えを導くことができることがわかります。また、このノードのマージは定数時間で行なえます。

すべて合わせて、この問題は $O(NlogN)$ で解くことができます。