ABC 041 解説

writer : sugim48

A : 添字

文字列 s の i 文字目を出力する問題です．多くのプログラミング言語では，`s[i]` で文字列 s の i 文字目を参照できます．ただし，以下の点に注意してください．

- 多くのプログラミング言語では，長さ N の文字列は先頭文字から順に 0，1，...，$N-1$ と番号が振られています．これを 0-indexed などといいます．
- この問題では，長さ N の文字列は先頭文字から順に 1，2，...，N と番号が振られています．これを 1-indexed などといいます．

入力される i は 1-indexed なので，あらかじめ i の値を 1 だけ減らして 0-indexed へ変換してから，`s[i]` と参照しなければなりません．

問題文中の番号が 0-indexed か 1-indexed かは問題によるので，問題文をよく読む必要があります．入力される番号が 1-indexed である場合，すぐに番号の値を 1 だけ減らして 0-indexed へ変換するとよいです．

B : 直方体

整数 A，B，C $(1 \leq A, B, C \leq 10^9)$ が与えられるので，$(A \times B \times C)$ % $(10^9 + 7)$ を求める問題です．ここで，% は剰余の演算子です．

いくつかのプログラミング言語では，整数型が表せる値に上限があります．たとえば，C 言語の `long long` 型が表せる値の上限はおよそ 9×10^{18} です．$A \times B \times C$ は最大で 10^{27} なので，`long long` 型では正しく表せません．そのため，$A \times B \times C$ を計算した後 $10^9 + 7$ で剰余をとる方法では，正しい答えが求まりません．

正しい答えを求めるためには，計算の途中結果が常に 9×10^{18} 以下に収まるようにしなければなりません．そのために，よく知られた剰余の性質

$$(A \times B) \% M = \{(A \% M) \times (B \% M)\} \% M$$

を使います．この性質を使うと，$(A \times B \times C)$ % $(10^9 + 7)$ は

```
(long long) A * B % 1000000007 * C % 1000000007
```

という式で計算できます．計算の途中結果が常に 9×10^{18} 以下に収まることが確認できます．

他の方法として，多倍長整数型（表せる値に上限がない整数型）を使って計算することもできます．実行時間に余裕があれば有用な方法です．

C：背の順

長さ N の distinct な整数列 a が与えられるので，a_i が大きい方から順に i を出力する問題です．このように，各 i に対応するある値 a_i に着目して i の列をソートする処理は，問題を解くための前処理として頻出です．この処理は以下のように行います．

まず，各 $1 \leq i \leq N$ について，a_i と i をこの順でペア $(a_i,\ i)$ としてまとめます．その後，$(a_i,\ i)$ の列を a_i の降順にソートします．ソートされた $(a_i,\ i)$ の列の i を参照すると，それらが欲しかった i の列となっています．2 つの値をペアとしてまとめるには，たとえば C++ では `std::pair` がよく使われます．

$(a_i,\ i)$ の列を普通にソートすると，a_i の昇順にソートされてしまいます．これを a_i の降順にソートするには，たとえば以下の方法があります．

- `std::sort` 関数の引数に `std::greater<T>()` を渡す．
- a_i の昇順にソートした後，`std::reverse` 関数で逆順にする．
- $(-a_i,\ i)$ の列を $-a_i$ の昇順にソートする．

どの方法もよく使われます．

D：徒競走

問題は次のように言い換えられます．

> N 個の頂点と M 本の有向辺からなるグラフがあります．辺 i は頂点 x_i から頂点 y_i へ張られています．N 個の頂点を横一列に並べます．このとき，各有向辺は左から右へ向かうようにします．頂点の並べ方は何通りでしょうか？

有向グラフが与えられたとき，上の条件を満たすように頂点を並べることを，**トポロジカルソート** といいます．よって，この問題はトポロジカルソートの方法の数え上げということになります．この問題は難しい問題として知られていますが，N が小さければ解くことができます．

部分点解法

$N \leq 8$ なので，N 個の頂点の並べ方を全探索することができます．それぞれの並べ方について，M 本の有向辺が左から右へ向かっているか判定すればよいです．最大のステップ数を見積もると，$N! \times M \approx 1.1 \times 10^6$ と十分小さいことが分かります．なお，順列を全探索するには，たとえば

C++ では `std::next_permutation` 関数が便利です．

満点解法

$N \leq 16$ なので，N 個の頂点の並べ方を全探索することはできません．そこで，**動的計画法 (DP)** によって解くことを考えます．

全頂点集合を $U := \{0, 1, \ldots, N-1\}$ とします．それぞれの頂点集合 $S \subseteq U$ に対して，

$$f(S) := (\text{頂点集合 } S \text{ をトポロジカルソートする方法の通り数})$$

と定義します．$f(\phi) = 1$ であり，また $f(U)$ が求めたい答えです．これらの間を繋ぐ漸化式を立ててみましょう．頂点集合 S をトポロジカルソートするとき，頂点 $v \in S$ が最も右に置かれるための必要十分条件を考えてみます．これは，頂点 v から頂点集合 $S - \{v\}$ へ向かう有向辺がないという条件になります．この条件を満たす頂点の集合を X_S $(\subseteq S)$ と書くことにします．つまり，

$$X_S := \{v \in S \quad | \quad v \text{ から } S - \{v\} \text{ へ向かう有向辺がない } \}$$

と定義します．すると，$f(S)$ の漸化式は次のようになります．

$$f(S) = \begin{cases} 1 & (S = \phi) \\ \sum_{v \in X_S} f(S - \{v\}) & (\text{otherwise}) \end{cases}$$

この漸化式に従って各 $f(S)$ を計算すれば，$f(U)$ が答えです．

以降は，実装上のテクニックを紹介します．プログラム上で頂点集合 S を表現する際は，二進数を使うのが便利です．頂点集合 S が頂点 i を含むならば第 i ビットを 1 とし，頂点 i を含まないならば第 i ビットを 0 とします．たとえば，$N = 5$ で $S = \{0, 1, 3\}$ ならば，対応する二進数は 01011 となります．こうすることで，頂点集合 S と整数のインデックスを対応付けることができ，$f(S)$ の値を配列に格納することができます．さらに，各 $f(S)$ を計算する際には，二進数の値が小さい方から順に計算していけばよいことが分かります．以上のテクニックを使った DP は特に **bit DP** と呼ばれています．

最後に，計算時間を見積もりましょう．最大のステップ数を見積もると，$2^N \times M \approx 7.9 \times 10^6$ と十分小さいことが分かります．このように，$N!$ は大きすぎるが 2^N は小さい場合，bit DP が想定解であることはよくあります．