第三回 アルゴリズム実技検定

writer: camypaper, gazzele, Kmcode, kobae964

2020 年 5 月 23 日 〜 6 月 6 日

A: ケース・センシティブ

まず、大文字小文字を考慮して s と t が一致するかどうかを判定し、その後大文字小文字を考慮しない場合に s と t が一致するかどうかを判定すればよいです。
大文字小文字を考慮しない判定はどちらも小文字に揃えてしまう、などが簡単です。

解答例

```
s = input()
t = input()
if s == t:
    print("same")
elif s.lower() == t.lower():
    print("case-insensitive")
else:
    print("different")
```

B: ダイナミック・スコアリング

以下の 2 つが分かれば、ある人のスコアは $O(M)$ で計算可能です。

- その人がどの問題を解いたか
- どの問題が何人に解かれたのか

上の 2 つの更新は $O(1)$ で可能です。スコアの計算クエリは毎回 $O(M)$ で実行しても $O(QM+N)$ となって十分高速です。

解答例

```
n, m, q = map(int, input().split())

cnt = [0 for _ in range(m)]
a = [[] for _ in range(n)]
for id in range(q):
    query = list(map(int, input().split()))
    if query[0] == 1:
        query[1] -= 1
        ans = 0
        for j in a[query[1]]:
            ans += n-cnt[j]
        print(ans)
    else:
        query[1] -= 1
        query[2] -= 1
        cnt[query[2]] += 1
        a[query[1]].append(query[2])
```

C: 等比数列

求める値はAR^{N-1}です。Rの値によって場合分けします。

- $R = 1$の場合、$AR^{N-1} = A$ですので、常にAを出力します。($A \leq 10^9$という制約に注意してください。)
- $R \geq 2$の場合、Nの値によって更に場合分けを行います。
    - $N \geq 31$の場合、$AR^{N-1} \geq 1 \times 2^{30} > 10^9$ですから、つねに `large` を出力します。
    - $1 \leq N \leq 30$の場合、単にAにRを$N-1$回掛けて、10^9との大小比較を行えばよいです。C++ などで実装を行う場合は、オーバーフローしないように注意して実装しましょう。

解答例

以下は Python3 で書かれた参考実装です。

```python
import sys
readline = sys.stdin.buffer.readline
sys.setrecursionlimit(10 ** 7)

a, r, n = map(int, readline().split())

if r == 1:
    print(a)
    exit()

if n >= 31:
    print('large')
    exit()

value = a * r ** (n - 1)
if value > 10 ** 9:
    print('large')
else:
    print(value)
```

D: 電光掲示板

入力例 1 が `0123456789` なので、これを利用しましょう。

先頭から 4 列ずつ見て、どの文字と一致するかどうかを二重ループなどで調べればよいです。実行時間制限には十分余裕があります。

解答例

```
digits = [
    ".###..#..###.###.#.#.###.###.###.###.###.",
    ".#.#.##....#...#.#.#.#...#.....#.#.#.#.#.",
    ".#.#..#..###.###.###.###.###...#.###.###.",
    ".#.#..#..#.....#...#...#.#.#...#.#.#...#.",
    ".###.###.###.###...#.###.###...#.###.###.",
]
n = int(input())
s = [input() for i in range(5)]
ans = ""
for j in range(n):
  for v in range(10):
    ok = True
    for x in range(5):
      for y  in range(4):
        ok&=s[x][j*4+y] == digits[x][v*4+y]
    if ok:
      ans+=str(v)
print(ans)
```

E: スプリンクラー

無向グラフの扱いを問う問題です。隣接する全ての頂点は隣接リストを使うと簡単に列挙ができます。

頂点の塗り替えクエリは $O(1)$ で、隣接する頂点の塗り替えクエリは $O(N)$ で実行でき、最悪ケースでも $O(QN + M)$ となって十分高速です。

F: Palindromic Matrix

文字列 S が構築できる条件は与えられる行列 a の i 行目と $N-i+1$ 行目の要素が 1 つ以上同じ文字を持つときです。

各対応する行のペアごとに同じ文字があるかどうか判定し、あればそのうち 1 つの文字を答えの文字列 S に採用すればよいです。

同じ文字があるかどうかの判定ですが、 26 種類の小文字アルファベットそれぞれが各行に含まれているかどうか管理しておくと実装が楽になります。

G: グリッド金移動

BFS でできます。計算量は グリッドの一辺の長さを K として $O(K^2)$ 程度で、$K = 403$ であるため余裕を持って間に合います。
$-200 \leq x \leq 200, -200 \leq y \leq 200$ の正方形領域の外側を通るのが最適である場合が存在することに注意してください。このようなケースには、 各辺一つずつ厚くした正方形領域 $-201 \leq x \leq 201, -201 \leq y \leq 201$ を考えることで対応できます。

H: 障害物競走

以下のような状態を持つ動的計画法を行います。

- 状態: DP[i] := 座標 i に接地した状態で到達したとき、かかる時間の最小値
- 状態遷移:
    - 行動 1 を行う。i から $i+1$ へ移動する。
    - 行動 2 を行う。i から $i+2$ へ移動する。
    - 行動 3 を行う。i から $i+4$ へ移動する。

以下のようなコーナーケースに注意してください。

- ジャンプ中にゴールを飛び越す
    - ジャンプ中にゴールに到達した場合は、走った距離とジャンプした距離がどちらも整数にならないので、計算するときに注意する必要があります。

I: Matrix Operations

まず、転置クエリをどう処理するか考えましょう。

今まで偶数回の転置クエリがあった場合、これは実質転置されていない状態とみなせます。

逆に、奇数回の転置クエリがあった場合、行の交換クエリは列の交換クエリに、列の交換クエリは行の交換クエリに読み替え、参照クエリでは行番号と列番号をスワップすればよいです。

次に、行の交換クエリをどう処理するか考えましょう。

これは、実は $O(1)$ で処理することができます。

具体的には現在の各行が初期状態の行列のどの行に対応しているか行番号をもっておきます。そうすると、その行番号だけを入れ替えればいいので、すべての要素を愚直に入れ替えなくても交換クエリを処理することができます。

列の交換クエリも同様です。

これらの情報をもとに要素の参照クエリも $O(1)$ で処理できます。

J: 回転寿司

それぞれの人が最後に食べた寿司の美味しさを並べた数列 b を作ることにします(ここで、まだ寿司を食べていない場合も美味しさ 0 の寿司を食べたことにしておきます)。
すると、b は常に広義単調減少な数列になっています(そうでないとすると、ある子供が自分が今までに食べた寿司よりも美味しさが大きい寿司を食べなかったことになり、問題文の規則と矛盾する)。$b_j < a_i$ を満たす最小の j は二分探索を用いて $O(\log N)$ で見つけることが可能です。全体として $O(M \log N)$ でこの問題を解くことができ、これは十分高速です。

K: コンテナの移動

コンテナ i の真下にあるコンテナを a_i とします(コンテナ i の真下にコンテナが存在しない場合は -1)。同様に、机 i の頂上にあるコンテナを b_i とします(机 i の頂上にコンテナが存在しない場合は -1)。a, b の構築は $O(N)$ で可能です。

すると、i 番目のクエリは f, t, x を用いて以下の事象が同時に起こるとみなせます。

- b_f を a_x で置き換える
- b_t を b_f で置き換える
- a_x を b_t で置き換える

これらの操作は $O(1)$ で実行可能です。最後に a, b を用いて、どのコンテナがどの机にあるかを求めればよいです。全体として $O(N + Q)$ で実行可能で十分高速です。

L: スーパーマーケット

S_i を、現在の時点で各列の先頭から i 番目にある商品の消費期限の値の集合、とします。特に、S_1 と S_2 だけが重要です。

S_1, S_2 の現在の時点での最大値は C++ ならば set を用いると $O(\log N)$ で取得可能です。最大値を求めた後、最大値が属する列の情報を更新すればよいです。先頭、あるいは先頭から 2 番目のみが取り除かれることに着目すると、$O(1)$ で更新が可能です。
その際、S_1, S_2 についても更新を行う必要がありますが、こちらも $O(\log N)$ で可能です。
全体として $O((N+M)\log N)$ で実行可能で十分高速です。

M: 行商計画問題

まずタカーハ氏が訪れる K 個の街間の最短距離を全て求めることを考えます。これはダイクストラ法を K 回実行することで $O(KM \log N)$ で計算可能です。よって問題を，K 頂点の重み付き完全グラフにおける巡回セールスマン問題（Traveling Salesman Problem，TSP）に還元できました。

頂点数 x のグラフにおける TSP は，動的計画法を用いて $O(x^2 2^x)$ で解くことができます。具体的には，$dp[u][S]$ で頂点集合 S を既に訪れて現在頂点 u にいるときのコストの最小値，と定めたとき，$dp[u][S]$ の値を $dp[v][S-u]$ $v \in S-u$ の各値から計算できることを利用します。この問題では $x = K$ なので，前半のダイクストラ法と合わせて全体で $O(KM \log N + K^2 2^K)$ の計算量で問題を解くことができます。

N: 入れ替えと並び替え

数列 a の転倒数（$a_i > a_j (i < j)$ である (i, j) の個数）は最初 0 であり，種類 1 のクエリで高々 1 増加することに注目します。種類 2 のクエリを処理するために，$a_i > a_{i+1} (x \leq i < y)$ である隣接要素のスワップを繰り返すことを考えます。このスワップにより数列の転倒数は 1 減少します。問題を通して種類 1 のクエリで増加する転倒数が高々 Q であることを考えると，種類 2 のクエリ処理で発生するこのタイプのスワップ回数が全体で Q 回で抑えられることが分かります。

次に $a_i > a_{i+1} (x \leq i < y)$ である i を高速に見つける方法を考えます。これには数列の隣接要素をスワップした際に新しく発生する（もしくは消える）条件を満たす i が，スワップした要素の両隣に限られることを利用します。種類 1，2 のクエリ双方の処理中に発生する（もしくは消える）条件を満たす i を，C++ では set などのデータ構造を用いて管理すればいいです。

この問題のように，データのある種の複雑さを表す値を取って，その値が「増えた分しか減らせない」ことを利用してデータに対する操作の計算量を償却する方法は，データ構造のポテンシャルを用いた解析と呼ばれています。

O: Quoits

一部の人には動的計画法で解きたくなるような制約ですが、実際に解こうとすると極めて大変な労力を費やすことになってしまいます。

この問題を解くには、これが最小費用流の問題であることをいかに早く見抜けるかがポイントです。

まず、問題文を以下のように言い換えます。

- 各棒 j はペナルティセットというものを持っていて、そのセットには $A_j \times (B_j), A_j \times (B_j)^2 - A_j \times (B_j), A_j \times (B_j)^3 - A_j \times (B_j)^2$ が入っています。
- 各ラウンド i で輪を棒 j に命中させると、あなたは $\left(A_j \times (B_j)^i \mod R_i\right)$ 点を得ると同時にその棒に対応するペナルティセットから好きな数字をちょうど 1 つ取り出してその数字分減点します。
- このときの得られる得点の最大値を求めてください。

この正当性は、$A_j \times (B_j) \leq A_j \times (B_j)^2 - A_j \times (B_j) \leq A_j \times (B_j)^3 - A_j \times (B_j)^2$ であることから明らかです。

実際にグラフを構築していきます。

- 各ラウンドを表す 3 個の頂点と各棒を表す N 個の頂点間において、容量 1、コストは $-1 \times \left(\text{対応するラウンドで対応する棒に命中させたときに得られる得点}\right)$ の辺をはります。
- 始点と各ラウンドを表す 3 個の頂点間において、容量 M、コストは 0 の辺をはります。
- 各棒を表す N 個の頂点と終点間において、容量 1、コストはペナルティセットに含まれるそれぞれの数字からなる辺をはります。（辺の数は合計 $3 \times N$ 本です。）

このグラフ上で、始点から終点に流量 $3 \times M$ を流して得られた最小費用に -1 をかけたものがこの問題の答えになります。