ABC 122 解説

出題・解説: @evima0

2019 年 3 月 24 日

A: Double Helix

正解までの道のりを細かく分割すると次の通りです。

1. 標準入力から文字 b を受け取る。
2. 何らかの方法で b と対になる塩基を表す文字 b' を得る。
3. c を標準出力に出力する。

手順 1, 3 については practice contest 問題 A の言語別サンプルコードが参考になります。手順 2 については、結局のところ $b =$ `A` なら $b' =$ `G`、$b =$ `G` なら $b' =$ `A`、... と 4 つの文字それぞれに相方を指定することになるでしょう。方法は大きく分けて二つ考えられ、if, else if, else や switch にあたる構文で b に応じてプログラムを分岐させるか、「答えの一覧」にあたるデータ構造を連想配列などで作って b に対応する値を参照するかです。C++, Python3 による後者のアプローチの実装例を示します。

```cpp
#include <iostream>
using namespace std;
int main(){
    char b, c[128];
    c['A'] = 'G';
    c['G'] = 'A';
    c['C'] = 'T';
    c['T'] = 'C';
    cin >> b;
    cout << c[(int)b] << endl;
}
```

```python
b = input()
c = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
print(c[b])
```

B: ATCoder

文字列 S の長さを N とします。S の部分文字列は、空文字列以外に $N + (N-1) + (N-2) + ... + 1 = N(N+1)/2$ 個存在します (S の 1 文字目から始まるものが N 個、2 文字目から始まるものが $N-1$ 個、...、N 文字目から始まるものが 1 個。なお、中身が同じでも S の異なる位置から取り出された部分文字列は別として数えています)。

この問題では $N \leq 10$ であり、この個数は最大で $10(10+1)/2 = 55$ です。これらをすべて調べて、そのうち最も長い ACGT 文字列の長さを報告してほしいという問題です (より効率的に解くこともできますが不必要です)。Python3 による実装例をもってこの解説の残りとします。

```
S = input()
N = len(S)
ans = 0
for i in range(N):
    for j in range(i, N):
        if all('ACGT'.count(c) == 1 for c in S[i : j + 1]):
            ans = max(ans, j - i + 1)
print(ans)
```

C: GeT AC

文字列 S の長さ N と問いの数 Q さえ小さければ、この問題は前問と同等の「書かれた通りに実装する」問題になります。しかし、実際には N も Q も最大で 10 万と大きく、「書かれた通り」の実装を行うと最大で延べ 100 億文字を走査することになります。これは、C++ のような高速な言語でも 2 秒で処理できる量ではありません。処理を高速化する何らかの工夫が必要です。

範囲内の部分文字列 `AC` を数えよと問われていますが、その代わりに「右隣が `C` であるような `A`」を数えた方が単純です。文字列中のこのような `A` を `a` に置き換えると、問 i は「l_i 文字目から $r_i - 1$ 文字目までの `a` を数えよ」となります (右端が 1 左に動いたことに注意してください。r_i 文字目が `a` であってもその右隣の `C` が範囲からはみ出てしまうためです)。

ここで、$t_i =$「S の 1 文字目から i 文字目までに出現する `a` の数」として数列 $t = \{t_0, t_1, \ldots, t_N\}$ を考えると、問 i の答えは $t_{r_i-1} - t_{l_i-1}$ として求められます。この数列 t を問いの処理を始める前に構築しておいて使い回せば、入力サイズの線形時間ですべての問いを処理できます。

```python
N, Q = map(int, input().split())
S = input()
t = [0] * (N + 1)
for i in range(N):
    t[i + 1] = t[i] + (1 if S[i : i + 2] == 'AC' else 0)
for i in range(Q):
    l, r = map(int, input().split())
    print(t[r-1] - t[l-1])
```

D: We Like AGC

N が小さければ、4^N 通りの ACGT 文字列すべてを候補として列挙した中から数えられますが、N が最大 100 では絶望的です。しかし、結論から述べると、これと同等のことを短時間で行うことができます。

第三の条件「隣接する 2 文字の入れ替えを 1 回行うことで第二の条件 (部分文字列として AGC を含まない) に違反させることはできない」を言い換えると、「部分文字列として GAC, ACG, A?CG, AG?C のいずれも含まない (? は何らかの 1 文字)」となります。

よって、条件を満たす文字列を 1 文字目から順に作るとき、「i 文字目を何にするか」の選択は、「$i-4$ 文字目以前が何であったか」を気にすることなく、$i-3, i-2, i-1$ 文字目のみを考慮して行うことができます。これを元に、メモ化再帰または動的計画法により N の線形時間で解を求められます (“定数倍” は大きいです)。

```python
N, MOD = int(input()), 10 ** 9 + 7
memo = [{} for i in range(N+1)]

def ok(last4):
    for i in range(4):
        t = list(last4)
        if i >= 1:
            t[i-1], t[i] = t[i], t[i-1]
        if ''.join(t).count('AGC') >= 1:
            return False
    return True

def dfs(cur, last3):
    if last3 in memo[cur]:
        return memo[cur][last3]
    if cur == N:
        return 1
    ret = 0
    for c in 'ACGT':
        if ok(last3 + c):
            ret = (ret + dfs(cur + 1, last3[1:] + c)) % MOD
    memo[cur][last3] = ret
    return ret

print(dfs(0, 'TTT'))
```