第二回 アルゴリズム実技検定 解説

chokudai, evima, kobae964, satashun, ynymxiaolongbao

2020 年 4 月 18 日 - 5 月 2 日

A: エレベーター

まず、上の階が下の階の整数 + 1 になるように、各階に整数を割り当てます。一例として、B9, B8, ..., B1 に -8, -7, ..., 0 を割り当て、1F, 2F, ..., 9F に 1, 2, ..., 9 を割り当てる方法があります。割り当てた後は、入力 S, T を読み取り、S と T に割り当てられた整数の差 (の絶対値) を出力すればよいです。

Listing 1　解答例 (Python3)

```
#!/usr/bin/env python3

# B9 -> -8, ..., B1 -> 0, 1F -> 1, ..., 9F -> 9
def to_int(f):
    if f[0] == 'B':
        return - (int(f[1]) - 1)
    return int(f[0])


f1, f2 = input().split()
print(abs(to_int(f1) - to_int(f2)))
```

B: 多数決

まず、a,b,c の各候補者について、何票獲得したかを調べます。これは、大きさ 3 の配列や連想配列などの各要素を 0 で初期化して、文字列 S の先頭から調べていき、見た文字に応じて該当する場所に 1 を足す、という処理を行えばできます。続いて、最多得票を得た人物を求めます。これは最大値を管理するための変数を用意し、 a,b,c の各候補者について、今持っている最大値よりも多かったら更新する、ということをやればよいです。

Listing 2　解答例 (Python3)

```
#!/usr/bin/env python3

s = input()
count = {'a': 0, 'b': 0, 'c': 0}
for c in s:
    count[c] += 1

ma = (0, '+')
for k, v in count.items():
    ma = max(ma, (v, k))

print(ma[1])
```

C: 山崩し

問題文の通りに $i = N - 1$ から順にシミュレートしていくことでマス目の状態を得ることができます。初期の色が黒であるマスにしか X が書かれないことに注意してください。

D: パターンマッチ

長さ 3 以下の英小文字と '.' から成るパターンの個数は、$27^3 + 27^2 + 27 = 20439$ 通りあります。これらを全て列挙し、S の各場所にマッチする場所があるかどうかチェックすると、時間計算量は $O(c^3 \times |S|)$ となります (c は文字の種類数)

E: 順列

1 以上 N 以下の整数 i それぞれに対して、再び i に戻るまで i を $f(i)$ で更新することを繰り返すと、i ごとに $O(N)$ 回の操作で最小の回数を求めることができ、全体の時間計算量は $O(N^2)$ となります。(順列であることからいずれ i に戻ることが言えます)

上述の解法でも十分高速ですが、i から初めたサイクルの途中に現れた値全てについても回数は同じになることが証明できるので、一度見た値を再び計算しないことで全体で $O(N)$ となります。

F: タスクの消化

あらかじめバケットソートを行い、$i = 1, 2, \ldots, N$ に対して i 日目から実行可能になるタスクを列挙しておきます。

任意の要素の追加および最大の要素の参照・削除ができる何らかの多重集合データ構造を用意し、$i = 1, 2, \ldots, N$ に対してこの順に、以下を行います。

1. 多重集合に、i 日目から実行可能になるタスクをすべて追加する。
2. 多重集合から、消化することで得られるポイントが最大のタスクを一つ選び、リストから削除し、現在の獲得ポイントにそのポイントの分を追加する。

多重集合データ構造としては、大きさ $\max_i B_i$ の配列や優先度付きキューを使うことができます。

- 前者の場合、データとして持つべきものは、得られるポイントごとにまとめた時のタスクの個数です。1 個のタスクの追加に $O(1)$ 時間、最大の要素の計算や削除に $O(\max_i B_i)$ 時間かかるので、全体の時間計算量は $O(N \max_i B_i)$ です。
- 後者の場合、1 個のタスクの追加に $O(\log N)$ 時間、最大の要素の計算や削除に $O(\log N)$ 時間かかるので、全体の時間計算量は $O(N \log N)$ です。(B_i には依存しません。)

G: ストリング・クエリ

まず、$T = 1$ のクエリに対して実際に文字を X_i 個追加する操作を行うと計算量が大きくなってしまうので、(文字種, 連続する文字数) でデータを管理することにします。このような状態で列を管理しておくと、$T = 1$ のクエリでは、列の末尾に (C_i, X_i) を追加すればよいです。

問題は $T = 2$ の削除クエリですが、これは列の前からの文字数の和が D_i 以下の間はデータを丸ごと削除し、最後の半端な部分が存在する場合、その場所だけは文字数を変化させるという操作を考えることでシミュレートできます。

計算量についてですが、この操作は deque というデータ構造で処理できることに注意すると (1 次元的な配列でも現在有効なデータを持っている範囲の両端を保持しておけば良いです) $T = 1$ では毎回 $O(1)$ で処理でき、$T = 2$ のクエリで見ることになる範囲は、毎回の操作では大きくなる可能性がありますが合計では $O(Q)$ であるので、全体でも $O(Q)$ で処理することができます。

H - 1-9 Grid

あるマス (r_1, c_1) からあるマス (r_2, c_2) までの移動回数の最小値は、$|r_1 - r_2| + |c_1 - c_2|$ です。

通るマスの組合せを全て列挙することはできませんが、条件を満たすある経路に含まれるマスのうち、S,1,2,...,9,G が書かれたマスをこの順に 1 つずつ取り出すような取り出す方が 1 つ以上存在します。

ここでそのようなマスを順に $(r_0, c_0), (r_1, c_1), \cdots, (r_9, c_9), (r_{10}, c_{10})$ と置きます (便宜上、0 は S, 10 は G のマスを表しています)

これらのマスをこの順で通るような経路の移動回数の最小値は、最初の考察により $\sum_{i=0}^{9}(|r_{i+1} - r_i| + |c_{i+1} - c_i|)$ ですが、求める答えは全ての $(r_1, c_1), \cdots, (r_9, c_9)$ の組合せについての最小値と一致することがわかります。

このままではまだ考える必要のあるマスの組合せが $O((NM)^9)$ 通りありますが、1 から 9 までが書かれたマスを分類し、$\mathrm{dp}[r][c] :=$ 直前にマス (r, c) を上述の取り出し方で使用した場合の移動回数の最小値と定義することで動的計画法で求めていくことができます。

この解法の時間計算量は $O((NM)^2)$ です。

I: トーナメント

まず、2^N 人分の結果を格納するための配列を用意します。

N 回のラウンドのシミュレーションを行います。i 回目のラウンドでは、残っている人全てに対して結果を格納する配列を i で更新し、その後隣り合う 2 人の強さの最大値からなる新しい配列を作り、次のラウンドで使います。i 回目のラウンドでは 2^{N+1-i} 人が戦うので、全体での時間計算量は $O(2^N + 2^{N-1} + \cdots + 2^1) = O(2^{N+1} - 2) = O(2^N)$ です。

J: 文字列解析

制約で、結果となる文字列はあまり長くならないことが保証されています。実装方法としては様々な手法が考えられますが、例えば次のように再帰的に結果の文字列を求める関数を設計することができます。現在ポインターが文字列のある場所を見ているとして、その文字が普通のアルファベットであればその文字を追加してポインターを 1 つ進め、'(' であればそこから対応する括弧までの区間を再帰的に変換した後続きを見ていきます。

K: 括弧

まず、括弧の対応が取れている文字列の判定は、次のようにスタックを用いて行うことができます。

文字列を前から見て、'(' が来たらスタックに '(' を積む。')' がきたら、スタックから '(' を 1 つ取り除く。途中でスタックから '(' を取り除くべき場合に存在しない場合がなく、また文字列を最後まで見たときにスタックが空であることが必要十分条件である。

この操作は、技術的には今括弧が何個開いているかという値を整数で持っておけばシミュレートすることができます。

この考察を活かして、$\mathrm{dp}[i][j] := i$ 文字目まで考慮し、現在 j 個の括弧が開いた状態である場合のコストの最小値 と定義して動的計画法を行うことができます。

状態 (i, j) からの遷移は、例えば i 文字をそのまま採用する場合は '(' であれば $(i+1, j+1)$、')' であれば $(i+1, j-1)$ に遷移します。変更する場合はその逆、削除する場合は $(i+1, j)$ に遷移します。

j が負になるような遷移は行うことができず、最終的な答えは $\mathrm{dp}[N][0]$ です。時間計算量は $O(N^2)$ です。

L: 辞書順最小

辞書順を最小化するためには、先頭が実現可能なものの中で最小、そしてそのうち 2 要素目が最小、$\cdots$ という風に扱うことができます。

まず、先頭として全体の最小値を採用できるかということを考えると、必ずしも可能ではありません。あるところから後ろ側をギリギリの間隔で作る場合にも $i, i+d, i+2d \cdots, i+(K-1)d$ となるので、先頭として使うことのできる要素は、数列のあるところから前側、という条件になっています。

この範囲で最小値を先として取り出せばいいですが、同じ値が複数個ある場合は前側を採用するべきです。(直感的には、それ以降の自由度が大きくなります) この操作を前から順番に K 回行えばよいですが、毎回使用可能になる範囲は後ろ側には d 個進み、前側は前回使用した要素 $+d$ 番目の要素からとなるので、優先度付きキューなどを用いることで時間計算量 $O(N \log N)$ で解くことができます。

M: 食堂

題意は複雑ですが、長さ D の円周上にメニューがあり、ある場所から回っていくという様に捉えることができます。あるメニューが好きな社員は、そのメニューがある日には必ず食堂を利用するというのが重要な考察です。すると、各メニューについてその次の同じメニューが出る日に向けて遷移を考えたとき、間に何回食堂を利用するかは L が一定であるため求まります。各メニューからその 2^k 回先まで同じメニューの日に遷移した場合に食堂を何回利用するかという情報をクエリに答える前に計算しておきます。(これは k が小さい方から全体に対して求めていくことで知ることができ、このような操作は doubling などと呼ばれています)

クエリに答える際には、まず最初の好みのメニューまでの回数を処理した後、上述の情報を用いて好みの料理を食べる回数について 2 分探索することができます。時間計算量は $O((N+D)\log\max(F))$ です。

N: ビルの建設

各ビルの座標について、単純に全ての敷地についてその座標を含むか調べると $O(NQ)$ です。

視点を変えると、各敷地について含まれるビルに $+C_i$ するという操作を行えると良いので、こちらで考えてみましょう。

2 次元ではなく 1 次元 (数直線上) であれば、ある区間 $[l,r]$ に $+C_i$ する・ある座標に加算されている値を求めるという操作は segment tree, Fenwick tree といったデータ構造を用いて高速に行うことが可能であることが知られています。

突然ですが、敷地を $X = xmin_i$ で y 座標についての区間 $[ymin_i, ymin_i + D_i]$ に $+C_i$ し、$X = xmin_i + D_i$ で y 座標についての区間 $[ymin_i, ymin_i + D_i]$ に $-C_i$ するクエリと捉えることにします。すると、ビルの計画についても $X = A_j$ で y 座標 $y = B_j$ に現在加算されている値の和を求めるクエリとみなすことで、全てのクエリを X 座標についてソートしてしまうことが可能であり、前述の 1 次元的な処理に帰着できます。このようにある座標を時間軸に取り替えることで次元を落とす手法は平面走査法と呼ばれています。

ある区間に加算する・ある場所の値を取得するという操作は両方 O(log 要素数) で行うことができます。クエリを扱う上では座標の大小関係のみが重要であるため、先に必要な y 座標をソートして区間の端として現れる値のみを取り出しておき、重要な座標に 0 から番号を振り直しておくと扱いやすいです。

全体の時間計算量は $O((Q+N)\log(Q+N))$ です。

O: 可変全域木

元のグラフの辺 i を e_i と表記します。以下の命題を証明します。

命題. 無向グラフ (V, E) に対し、Kruskal 法で構成した最小全域木を (V, T) とする。辺 $e' \in E$ を含む重み最小の全域木 (の 1 つ) は、ある辺 $e \in T$ に対して $(V, T \setminus \{e\} \cup \{e'\})$ と書ける。$e' = \{u, v\}$ としたとき、e は木 (V, T) における u から v への最短パスの上の重み最大の辺である。

Proof. e' をあらかじめ結合させた Union-Find 木を使って、Kruskal 法を実行することにより、e' を含む重み最小の全域木が得られる。Kruskal 法の実行時に T の辺を優先することにすれば、$T' \setminus \{e'\} \subseteq T$ が成立する。

e を $T \setminus T'$ に含まれる唯一の辺とする。$T' = T \setminus \{e\} \cup \{e'\}$ が成立する。$e' = \{u, v\}$ としたとき、$(V, T \setminus \{e\} \cup \{e'\})$ が全域木であるためには、e が u と v の間の最短パスの上になければならない。このとき e の重みが最大でないと仮定するとより重みの大きい辺 f がパス上に存在するはずだが、$(V, T \setminus \{f\} \cup \{e'\})$ が全域木であって $(V, T \setminus \{e\} \cup \{e'\})$ よりも重みが小さいので矛盾。 □

以上を踏まえ、以下のように解くことができます。最小全域木 (V, T) を一つ求めます。また、木 (V, T) における u から v への最短パスのことを u-v パスと呼ぶことにします。

並列二分探索解

$1 \leq i \leq M$ を満たす各 i について、$e_i = \{u, v\}$ として u-v パスの上の辺の重みの最大値を求めたいです。ここで、i が一つだけであれば、 Union-Find 木を用いて最小全域木の重みの小さい辺から繋げていくことで、u と v がどの辺の追加のタイミングで連結になるかを求めることができます。この方法だと各 i に対して $O(N\alpha(N))$ 時間かかり、全体で $O(NM\alpha(N))$ 時間かかるため、到底間に合いません。 ($\alpha(N)$ はアッカーマン関数の逆関数)

そこで、M 個の問題すべてについて、並列に二分探索を行うことを考えます。「最小全域木の $N-1$ 本の辺を重みの小さい順に追加していく」という流れを $\lceil \log_2 \max A_i \rceil$ 回繰り返し、追加する流れの途中の適切なタイミングで連結性を判定します。この「適切なタイミング」は、過去の二分探索の流れで求まった連結性の真偽により決まります。これの時間計算量は $O((N\alpha(N) + M) \log \max A_i)$ で、空間計算量は $O(M)$ です。最小全域木の構成が $O(M \log N)$ 時間、$O(M)$ 空間でできる[*1]ため、全部合わせて時間計算量 $O((N\alpha(N) + M) \log \max A_i + M \log N)$、空間計算量 $O(M)$ で、これは十分に高速です。

[*1] 辺の重みのソートがボトルネックです。グラフは連結なため $N - 1 \leq M \leq N(N-1)/2$ が成立し、したがって $\log M = O(\log N)$ であることに注意してください。

最小共通祖先 (LCA) 解

$1 \leq i \leq M$ を満たす各 i について、$e_i = \{u, v\}$ として u-v パスの上の辺の重みの最大値を求めたいです。ここで、適当に根を定めて木 (V, T) を根付き木にします。ダブリングで、$0 \leq i \leq \log_2 N$ を満たす整数 i および各頂点について以下の値を求めておきます:

- 2^i 個上の先祖の番号
- 2^i 個上の先祖に至るまでの 2^i 個の辺の重みの最大値

u-v パスの上の辺の重みの最大値は、u と v の LCA が求まれば、あらかじめ計算しておいたデータから $O(\log N)$ 時間で求めることができます。LCA を求めるのに必要な時間も $O(\log N)$ 時間であるため、全ての辺に対して合計 $O(M \log N)$ 時間で必要な値が求まります。最小全域木の構成が $O(M \log N)$ 時間 $O(M)$ 空間で、ダブリングの計算が $O(N \log N)$ 時間 $O(N \log N)$ 空間でできるため、合計 $O(M \log N)$ 時間 $O(N \log N + M)$ 空間で解けました。