5

名前 (Name)

Author: 米田 寛峻 (square1001)

0　はじめに

この問題では、S と T を部分列に含み、どの同じ文字の間にも別の文字が K 文字以上あるような最短の文字列の文字数を求める必要があります。

1　小課題 1 ($S = T, K = 0$)

$S = T, K = 0$ では、文字列 S がそのまま条件を満たします。その文字数 N を出力すればよいです。

2　小課題 2 ($S = T, K = 1$)

$S = T, K = 1$ では、文字列 S で同じ文字が隣接する箇所があった場合、そのままでは条件を満たしません。同じ文字が隣接する箇所について、間に別の文字を 1 文字挿入してやれば、条件を満たすようになります（そしてそれは当然最短です）。文字数は $N + (S[i] = S[i+1]$ となる i の個数) となります。

3　小課題 3 ($S = T$)

小課題 2 と同じように考えます。同じ文字の間に $K - 1$ 文字未満しかない箇所があった場合、いくつか文字を挿入して条件を満たすようにしてやる必要があります。

具体例として、$K = 3$ で S, T が abcdbcda の場合を考えます。まず、2 つの b の間には 2 文字しかないので、その間に別の文字を 1 文字挿入する必要があります。bc の間、cd の間、db の間のどこに挿入するのが良いでしょうか？答えは、一番後ろの db の間に挿入するのが最適です。理由は、そうすれば 2 つの c の間と 2 つの d の間（両方とも間に 2 文字しかない）にも 1 文字挿入したことになり[*1]、より条件の達成に近づけるからです（図 1）。

[*1] 実際には 2 つの a の間にも挿入されていますが、2 つの a の間には既に 3 文字以上あるので考える必要はありません。

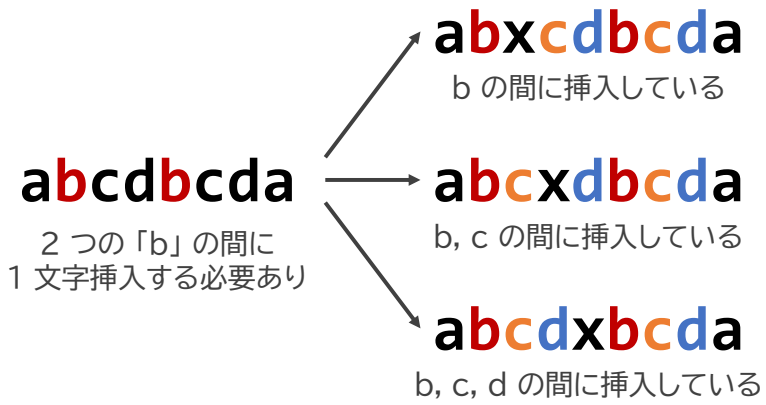

図 1　abcdbcda の 2 つの b の間の挿入位置による文字列の変化

しかし、この考え方ではどこに挿入するか定まらない場合もあります。例えば $K = 3$ で文字列が xayxzybzab のときに、2 つの y の間に 1 文字挿入することを考えます。このとき

- yx の間に何か別の文字を挿入すると、2 つの y の間、2 つの x の間に挿入したことになる
- zy の間に何か別の文字を挿入すると、2 つの y の間、2 つの z の間に挿入したことになる

となり、どちらを選べばよいかの判断がつきません[*2]。

しかし、$S[l] = S[r]$ $(1 \leqq r - l \leqq K - 1)$ となる箇所の中で r が最も左にあるものに着目すれば、必ず「挿入する場所として一番右を選べばよい」と決まります[*3]。前の例ならば、y ではなく x に着目すべきだったということです。「このような (l, r) を見つけ、$S[r-1]$ と $S[r]$ の間に（左右 K 文字とは異なる）1 文字を挿入する」を条件を満たす文字列ができるまで繰り返すことで、最適解が求まります。(l, r) を探すのに計算量 $O(NK)$ かかり、最大で NK 回程度の繰り返しが起こるので、全体の計算量は $O(N^2K^2)$ となります。

もっとシンプルな計算量 $O(NK)$ の解法もあります。

1. 答えの文字列を Z とする。最初、Z を空文字列とする。
2. $i = 1, 2, \ldots N$ の順に以下を繰り返す。
   - Z の末尾にダミーの文字を（$S[i]$ が直前の同じ文字と K 文字以上空けるように）必要最小限な数だけ追加した後、Z の末尾に $S[i]$ を追加する。

このように、現時点で最も良いと考えられる選択を繰り返して答えを出す手法を **貪欲法** といいます。

[*2] 最適解となるのは前者の場合のみとなります。条件を満たす最適な文字列の一例は xayPxzybQzab です。
[*3] 理由は少し難しいですが、ある（間に $K-1$ 文字以下しかない）2 つの文字の間に挿入される条件が「一定より右の場所に挿入すること」となるからです。

4　小課題 4 ($K = 0$)

$K = 0$ の場合は「S と T を部分列に含む最短の文字列」の文字数を求めることになります。この問題は、**動的計画法**[*4]という手法を用いて解くことができます。

まずは、解法をイメージしやすいように、図 2 のように S と T を同じ列に同じ文字が来るように縦に並べるときに最短何列必要か、という問題として考えます。図 2 は、S が JOIIOI で T が JOIGEGOI のときに、最短の文字列 JOIGEIGOI が作られる場合の例を示しています。

S:	J	O	I			I		O	I
T:	J	O	I	G	E		G	O	I

図 2　JOIIOI と JOIGEGOI を部分列に含む最短の文字列

この並べ方を左から順に作っていくことを考えます。状態 (i, j) を「現時点で S の i 文字目まで、T の j 文字目までが作られた状態」とします。例えば、図 2 で 6 列目まで作ったら「状態 $(4, 5)$」です（図 3）。

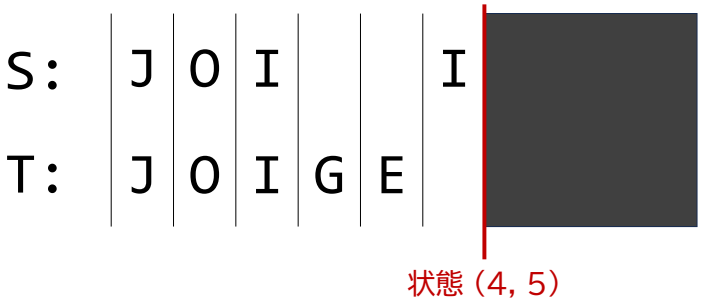

図 3　状態 $(4, 5)$: S の 4 文字目まで、T の 5 文字目までが作られた状態

$d[i][j]$ を「状態 (i, j) にするために最小何列必要か」とします（例 : $d[4][5] = 6$）とします[*5]。これを (i, j)

[*4] 動的計画法 (Dynamic Programming, DP) とは、小さいスケールの問題から順に、これまでの結果を利用して答えを積み上げていくことで、最終的な問題の答えを求める方法のことです。動的計画法を使って解く問題は、情報オリンピックでもよく出題されます。分からない方はインターネットなどで調べてみましょう。書籍による説明では，「競技プログラミングの鉄則」（米田優峻著） p.105-152 などがあります。

[*5] $d[i][j]$ を「『S の i 文字目まで』と『T の j 文字目まで』を部分列に含む最短の文字列の文字数」としても結果は同じですが、図 2 のような形で考えたほうが DP の式が立てやすいです。

の小さい順に計算していくことを考えます。まず、状態 (i, j) にたどり着くためには、直前の状態から以下のいずれかのようにする必要があります。

- 状態 $(i-1, j)$ から、次の列で 1 行目に $S[i]$ を書く。
- 状態 $(i, j-1)$ から、次の列で 2 行目に $T[j]$ を書く。
- 状態 $(i-1, j-1)$ から、次の列で 1 行目に $S[i]$、2 行目に $T[j]$ を書く。これは $S[i] = T[j]$ の場合にのみ行える。

よって、$d[i][j]$ は以下の式で求めることができます。

$$d[i][j] = \begin{cases} \min\{d[i-1][j]+1, d[i][j-1]+1\} & (S[i] \neq T[j]) \\ \min\{d[i-1][j]+1, d[i][j-1]+1, d[i-1][j-1]+1\} & (S[i] = T[j]) \end{cases} \quad (1)$$

この問題の答えは $d[N][M]$ ですので、計算量 $O(NM)$ で答えを求めることができます。

小課題 4 には別の解法もあります。最長共通部分列問題 (LCS) というよく知られた問題があります。これは、文字列 S と文字列 T の両方の部分列として現れる最長の文字列を求めてください、というものです（例：`JOIIOI` と `JOIGEGOI` の最長共通部分列は `JOIOI`）。このとき、元の問題の答えは、$N + M -$ (S と T の最長共通部分列の長さ) となります。最長共通部分列は動的計画法を使って計算量 $O(NM)$ で求められるので、元の問題も計算量 $O(NM)$ で解けました。

5　小課題 5 ($K = 1, N \leqq 25, M \leqq 25$)

基本的には小課題 4 の解法と同じように考えたいです。でも、$K = 1$ の場合は、隣接する文字が同じではいけないという制約があります。なので、図 2 と同様に文字を並べると、どうしても $S[i]$ も $T[j]$ も書かない列が出てくる場合があることに注意してください（図 4）。

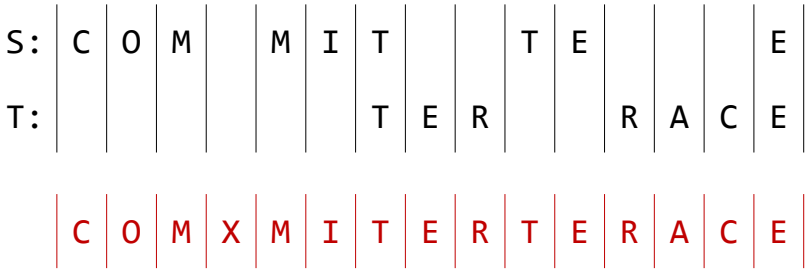

図 4　$K = 1$、S が `COMMITTEE`、T が `TERRACE` のときの最短の文字列

$d[i][j][k]$ を「状態 (i, j) で、最後の列の文字が k（アルファベット 52 種類のいずれか）にするために、最小何列必要か」とします。(i, j, k) の時点から、次の列を作ったとき、以下のいずれかになります。

1. 次の列に何も書かないことで、(i, j, k') に遷移する（ただし $k \neq k'$）。
2. 次の列で 1 行目に $S[i+1]$ を書くことで、$(i+1, j, S[i+1])$ に遷移する。
3. 次の列で 2 行目に $T[j+1]$ を書くことで、$(i, j+1, T[j+1])$ に遷移する。
4. 次の列に 1 行目に $S[i+1]$、2 行目に $T[j+1]$ を書くことで、$(i+1, j+1, S[i+1])$ に遷移する。これは $S[i+1] = T[j+1]$ の場合にのみ行える。

よって、各 (i, j, k) を頂点としたグラフで、上に述べた状態遷移に従って（辺の重みが 1 の）有向辺を張ったものを考えたときに、頂点 $(0, 0, ?)$ から頂点 $(N, M, ?)$ までの最短距離が答えになります。これは、文字の種類数を $\sigma = 52$ として、$NM\sigma$ 頂点 $NM\sigma^2$ 辺程度のグラフになります。よって、幅優先探索 (BFS) をすれば[*6]、答えが計算量 $O(NM\sigma^2)$ で求まります[*7]。

6　小課題 6 ($K = 1$)

小課題 5 の解法は、直前の文字 k を限定することによって、高速化することができます。

現時点の状態が (i, j) であるとき、最後の列の文字 k として「$S[i]$」「$T[j]$」「ダミーの文字（列に何も書かない場合）」の 3 通りしか考えなくてよいです。すると、グラフの頂点数を $3NM$ 程度に、辺数を $12NM$ 程度に減らせます[*8]。よって、幅優先探索をすれば、答えが計算量 $O(NM)$ で求まります。

小課題 6 には、動的計画法 (DP) を使った別の解法もあります。今度は、図 5 のように、何も書かない列を許さないことを考えます。すると、同じ文字が隣接する場合があり、その時は「何かダミーの文字を入れる」ことになります。

図 5　$K = 1$、S が COMMITTEE、T が TERRACE のときの最短の文字列（空列を許さない場合）

$d[i][j][0]$ を「状態 (i, j) で最後の列の文字が $S[i]$」、$d[i][j][1]$ を「状態 (i, j) で最後の列の文字が $T[j]$」に

[*6] この方法では、動的計画法を直接使えないことに注意してください。理由は、作られるグラフにループが生じるからです。
[*7] 頂点 $(0, 0, k)$ からの最短経路を各 k に対して求めれば計算量 $O(NM\sigma^3)$ かかりますが、幅優先探索の最初の時点でキューにすべての $(0, 0, k)$ を追加してやれば、1 回しか幅優先探索を行う必要がありません。
[*8] 各 (i, j) に対して、小課題 5 の解法の説明にある 1, 2, 3, 4 の遷移は最大でも 9, 1, 1, 1 通りとなります。

するために最小何列必要か、とします。小課題 4 と同じように考えて動的計画法の式を立てます。まずは $d[i][j][0]$ に関して以下の式が作れます[*9]。

$$d[i][j][0] = \begin{cases} \min\{d[i-1][j][0] + c(S[i-1], S[i]), d[i-1][j][0] + c(T[j], S[i])\} & (S[i] \neq T[j]) \\ \min\{d[i-1][j][0] + c(S[i-1], S[i]), d[i][j-1][0] + c(T[j], S[i]), & \\ \quad d[i-1][j-1][0] + c(S[i-1], S[i]), d[i-1][j-1][1] + c(T[j-1], S[i])\} & (S[i] = T[j]) \end{cases} \tag{2}$$

ただし、$c(x, y)$ は「$x = y$ のとき 2、$x \neq y$ のとき 1」であり、x と y が隣接したときに追加で必要になる文字数を表します。$d[i][j][1]$ に関しても同様の式が作れます。よって、答えが計算量 $O(NM)$ で求まります。

7　小課題 7, 8

小課題 7, 8 は、前の小課題よりも難易度が高いので、解説も少し難しい内容になります。

小課題 5 の解法をそのまま $K \geqq 2$ の場合に一般化しようとすると、最後の K 列の文字を状態として持っておく必要があるので、文字の種類数を $\sigma = 52$ として、グラフの頂点数が $O(NM\sigma^K)$ となり、計算量が $O(NM\sigma^{K+1})$ となってしまいます。これでは実行時間制限には到底間に合わないので、なんとかして「直前 K 列の情報」の量を削減できないか考えます。

現在状態 (i, j) まで作ったとします。このとき、最後の列は「1 行目に $S[i]$ を書く」「2 行目に $T[j]$ を書く」「$S[i] = T[j]$ の場合に、1 行目に $S[i]$ を書き、2 行目に $T[j]$ を書く」「何も書かない」の 4 通りがあり得ます（このとき最後の列の文字はそれぞれ「$S[i]$」「$T[j]$」「$S[i] = T[j]$」「ダミーの文字」となります）。

では、その前の列まで考えると何通りあるのでしょうか？各「最後の列のパターン」ごとに 4 が通り考えられるので、$4 \times 4 = 16$ 通り考えられます（図 6）。

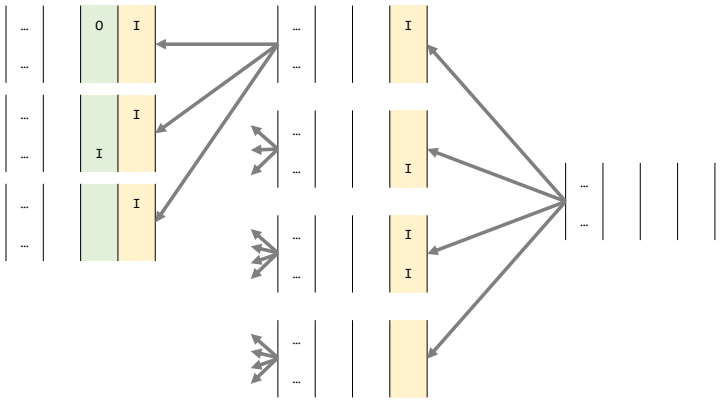

図 6　S が JOIIOI、T が JOIGEGOI のときの最後の列、最後から 2 番目の列の可能性

[*9] $d[i][j][0]$ では最後の列の文字が $S[i]$ なので、直前の状態は $(i-1, j)$ か $(i-1, j-1)$ である必要があります。

このように考えていくと、最後の K 列の状態は多くとも 4^K 通りになります。これは元々 σ^K 通りだったのと比べると大きな改善です。そうすると、小課題 5, 6 と同じようなやり方で幅優先探索で解くと、計算量 $O(NM4^K)$ で答えが求まります。

なお、$S[i] = T[j]$ の場合に「1 行目に $S[i]$ を書く」「2 行目に $T[j]$ を書く」パターンを考えなくても最適解は同じなので、状態数を 3^K 通りに減らして、計算量 $O(NM3^K)$ で解くこともできます。

動的計画法で解くこともできます。グラフからループをなくすために、例えば「『何も書かない』が K 連続するようにはしない」とする方法が考えられます。動的計画法を使うと、先ほどの幅優先探索の解法と同じ計算量オーダーですが、(定数倍の面で) 比較的高速なプログラムを書くことができます[*10]。

この問題では、制約が $N \leqq 500, M \leqq 500, K \leqq 3$ で実行時間制限 1 秒と厳しめなので、実装のやり方やプログラミング言語によっては小課題 8 ($K \leqq 3$) や、場合によっては小課題 7 ($K \leqq 2$) でも実行時間制限超過 (TLE) になる可能性があります。しかし、実装の工夫をすれば、Python (PyPy3) でも 0.3 秒以内の実行で満点を取ることができます。

8 補足

日本情報オリンピックのウェブサイト上に、小課題 3, 8 の解法を実装した C++, Python プログラムを載せています。ご参考になれば幸いです。

[*10] この理由として、キューを使わずに済むこと、計算式がより単純化されることなどが挙げられます。