アルゴリズムとデータ構造 B　グループワーク　報告書

水本 幸希

2023 年 1 月 26 日

1　メンバー

- 水本 幸希
- 山口 慧
- 杣谷 星音
- 斧田 洋人
- 杉山 亮太

2　基本的な考え方

与えられたクエリと放送局のデータを切り取ったもの比較して、編集距離が最短となった放送局を答えとする。

3　編集距離の求め方

編集距離を求める方法には様々なものが存在するが、今回はビットパラレル法を採用した。ビットパラレル法は動的計画法を応用しており、ビット演算を利用することで並列処理を行うことで計算時間を短縮している。動的計画法の場合、計算量は $O(N^2)$ であるが、ビットパラレル法を用いると $O(N)$ まで計算量を減らすことができる。ただし、この計算量にするには配列の要素数が演算に使用するビット長と同じかそれ以下でなければならない。C 言語の場合、使用できる変数の最大ビット長は 64 ビットであるからクエリの長さが 65 文字以上だとそのまま適用することができない。そのため今回はクエリの 65 文字目以降を捨てることで解決させた。これはクエリの長さが長いほど正答率が高くなり、64 文字あればほぼ確実に正解を出せるためである。

3.1　各アルゴリズムでの計測時間の比較

以下に、様々な方法での実行時間を掲載する。なお、動的計画法での実行時間を 100 としている。各アルゴリズムの詳細については割愛する。

動的計画法	O(ND) アルゴリズム	O(NP) アルゴリズム	ビットパラレル法
100	65.39	25.33	1.965

表 1　各アルゴリズムでの実行時間

このように、ビットパラレル法が最も効率良く計算できることが分かる。なお、O(NP) アルゴリズムと 64 文字で打ち切ったビットパラレル法のスコアを比較すると次のようになった。なお、本番で使用したプログラムに O(NP) アルゴリズムを組み込むと制限時間内に終わらないので、精度を粗くして計測した。（スコアは 5 個の平均値）

O(NP) アルゴリズム	ビットパラレル法
9749	9702

表 2　文字の切り取りによる精度の違い

このように、64 文字あればほぼ問題なく答えを当てられるが、少し精度が落ちる。今回スコアを極限まで上げることができなかったのは 64 文字だけで判定したことことが原因だったと考えられる。

4　放送局の信号分割と打ち切りの条件

信号をクエリの長さだけ切り取る、ということを全ての箇所で行えば理論上かなり高い確率で当てることができる。ただし、切り取った開始位置の候補は、50 万個存在しこれを全て走査してしまうと 10 秒以内には終わるが時間がかかりすぎてポイントに影響してくる。そのため、ある程度走査は粗くする必要性が生じる。以下に走査の幅とスコアの関係について、表と図で示す。変化が分かりやすいよう、高いスコアの出しにくいエラー率高めのケースで確認を行った。

走査のステップ（クエリ長/N）	スコア	実行時間 [Sec]
1.0	3375	0.159
2.0	5345	0.323
3.0	7165	0.438
4.0	8470	0.605
5.0	8360	0.719
6.0	8395	0.879
7.0	8785	1.005
8.0	8815	1.108
9.0	8815	1.296
10.0	8985	1.453
15.0	9225	2.328
20.0	9280	3.248
走査幅=1	9175	6.708

表 3　走査幅とスコアの関係

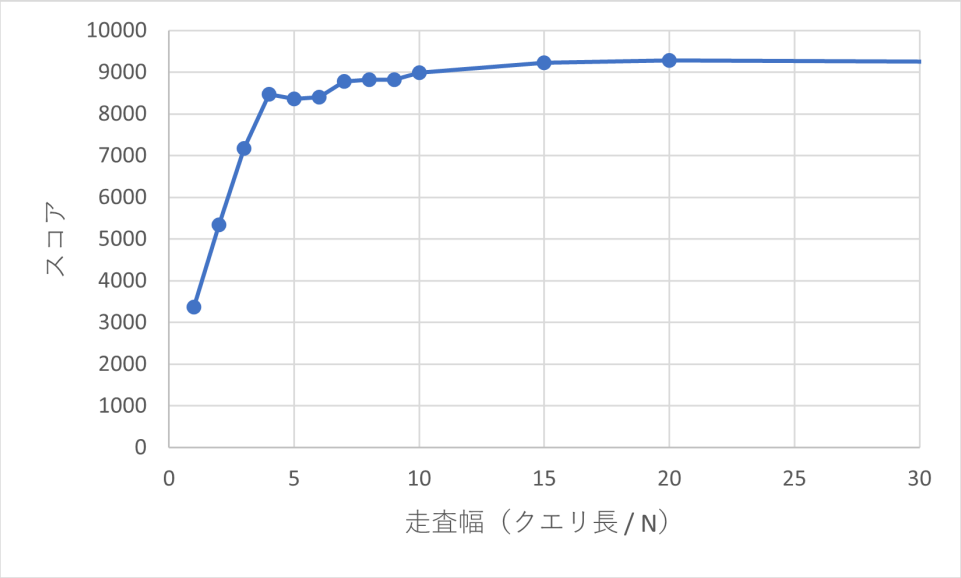

図 1　走査幅とスコアの関係

今回は、放送局の持つ信号からクエリの長さ分だけ信号を切り取って（最初は当然放送局の最初の位置から）編集距離を調べた後、放送局から切り取った信号の開始地点から (クエリの長さ)/10 だけ右に移動し、再度編集距離を求める、といったことを繰り返すということを繰り返した。この場合、切り取った部分を走査するときは最大で 1/20 ずれる可能性がある。しかし、これ以上細かく走査してもスコアはあまり上がらず、逆に粗くすると正答率が低くなったので、走査するときのステップはこのように設定した。

また、これだけでも十分時間内に答えを求めることができるが、編集距離が著しく低いものが発見されたらその時点で答えを確定させて打ち切ってしまえば、正答率を下げることなく必要な時間をさらに短縮することができるその閾値の設定も色々試すことで決定した。以下に実験してみた結果を示す。この実験結果もエラー率が高い状態を想定している。

打ち切りの閾値（クエリ長/N）	スコア	打ち切った回数	実行時間 [Sec]
1.0	0	100	0.009
2.0	1800	100	0.233
3.0	7750	77	1.055
3.5	8800	54	1.271
4.0	9100	33	1.448
5.0	8800	12	1.627
6.0	8955	4	1.642
7.0	8970	1	1.651
8.0	9100	0	1.700
9.0	8815	0	1.703
10.0	8860	0	1.714
打ち切りなし	8970	0	1.719

表 4　打ち切りの閾値とスコアの関係

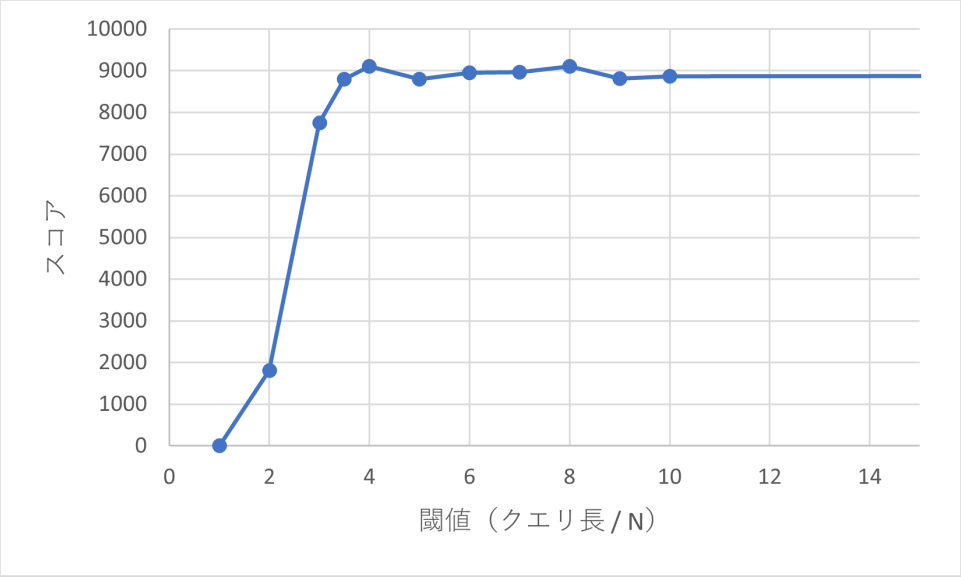

図 2　打ち切りの閾値とスコアの関係

今回は、編集距離がクエリの長さの 1/4 以下になった場合はその場で打ち切った。これ以上小さくしても結果は変わらず逆に大きくすると、間違える可能性が増加したためである。なお、発表を聞いて編集距離の打ち切りに関しては信頼区間内に収まっているかどうかで判断すればよいということを新しく知った。今回は手探りでパラメータを指定したが、今後は手探りだけではなく論理的に設定することが重要だと感じた。

5　クエリを聞き直す条件

これまでの工夫で、それなりに正解を出すことができるが信号が短い場合の正答率が低くなってしまう。これは各放送局での最短編集距離が最も小さい放送局が複数存在する場合、最初に発見した放送局を答えとしてしまうようなプログラムにしたことが原因であった。これを解決するため、最短編集距離が最も短いものが複数存在する場合はクエリを聞き直し、それらの放送局に対して最短編集距離を再度求めることで、放送局を確定させた。これによってさらに正答率を上昇させることができた。一方、最短編集距離と差が 1 しかない場合も聞き直してみたが、精度が上がらないどころが時間内に終わらせることができなかったので却下となった。

6　苦戦したところ

今回ビットパラレル法を用いた手法を提案した。この手法では挿入、削除コストは 1、置換コストは 2 となっている（置換は考慮されず、挿入と削除を組み合わせて計算しているため)。色々試したところ、今回の条件では置換コストを 2 とした方が確実に正答率が上昇したため置換コストは 1 とすることはしなかった。そのため、挿入、削除率が高い場合でも高いスコアを出すことができるが、置換率が高い状況では高いスコアを出すことが難しかった。特に、クエリの長さが短い場合は正答を出すことが困難であり今回はそれらのケースで確実に正解を出すことは断念した。

7　没となった案

1. メモリピークを抑えるために各データを 2 ビット（0, 1, 2, 3）に圧縮
   → 制限時間内にプログラムが終わらなくなったので没
2. クエリを聞き直したときにすでに聞いた信号と合わせてより正確な信号にする
   → やり方が分からなかったので没
3. 何個かのクエリを用意してそれぞれの場合での編集距離の平均から最短編集距離を出す
   → 全くスコアが上がらなかったので没

8　結果

以下に 100 個のテストケースで実行したときのスコアを記載する。（下の二つは最初に配布された 10 個のテストケースでの平均値）

- スコア 968035
- 実行時間 1.24 [Sec]
- メモリピーク 2.209 [MB]
- クエリを聞き直した回数 9.7 回
- 編集距離が短いものを発見して中断した回数 62.7 回

答えを外したのは、クエリが短かったり、置換率が高い場合がほとんどであった。また、エラー率が高い状況では他班と比較して高いスコアが得られたが、逆にエラー率が低い条件では高いスコアを出すことができ

ず、1 位を取ることが出来なかった。これは、65 文字目以降を捨てていることでわずかに精度が落ちていることが原因であると考えられる。

9　まとめ

- ビットパラレル法を用いて各放送局の最短編集距離を求めるが、クエリの 65 文字目以降は使わずに予測する。
- 走査は 1 バイトずつではなく、1 回につきクエリの長さの 1/10 だけ右に進めていく
- 編集距離が著しく短い（クエリの長さの 1/4 以下）場合はその時点で答えを確定させてそのクエリに対する走査は打ち切る
- 最短編集距離が複数で同一だった場合はクエリを聞き直して再度求めなおす
- 置換コストは 2、挿入、削除コストを 1 とすると切り取った部分以外との編集距離を増やすため正確に答えを出せる
- 今回の手法では、「クエリが短い」、「置換率が高い」場合に正答率が低くなる

以下にフローチャートを示す。

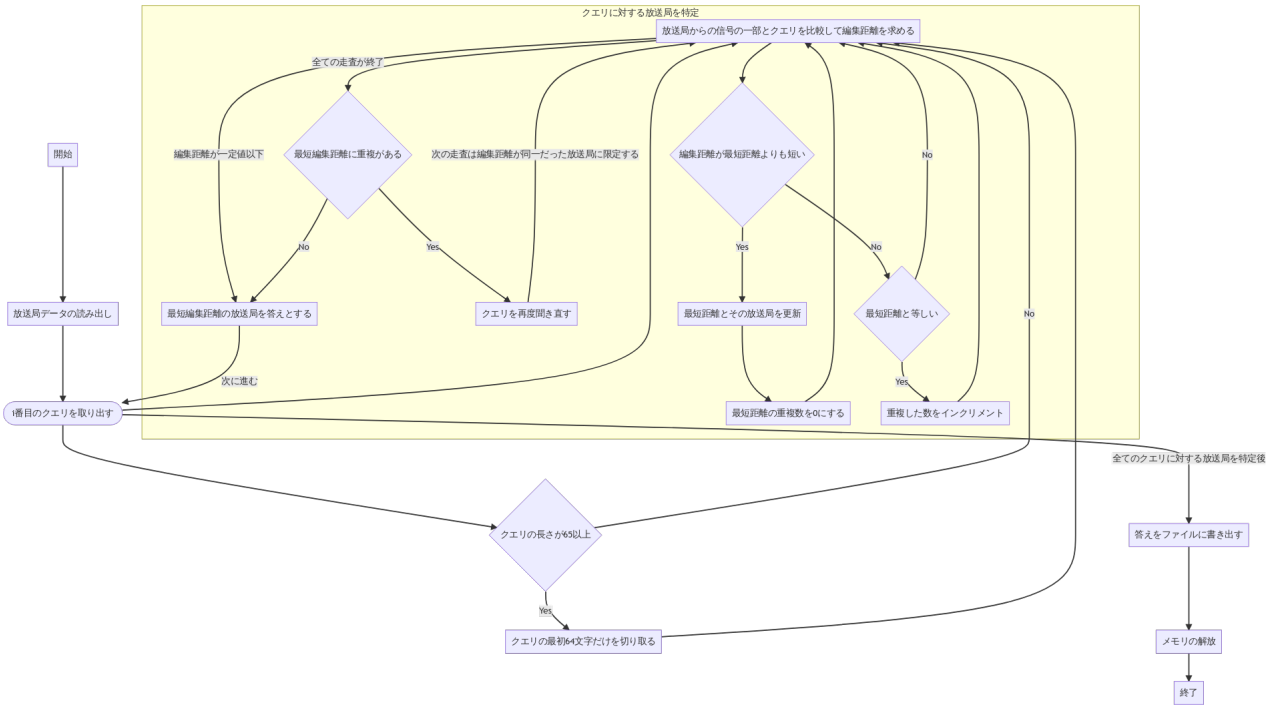

図 3　実行フローチャート

10 ソースコード

以下に今回使用したソースコードを示す。マクロを定義することで、得点や時間が表示されるように設計した。

```
1  // #define EVALUATE_MODE
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #ifdef EVALUATE_MODE
7  #include <time.h>
8  #endif
9
10 #include "ask.h"
11
12 #ifndef INT_MAX
13 #define INT_MAX 0x7fffffff
14 #endif
15
16 static int p_ins, p_sub, p_del;
17 static char **S;
18 static char *q;
19
20 #ifdef EVALUATE_MODE
21 static int abort_count = 0;
22 static int ask_count = 0;
23 static int compute_time;
24
25 void evaluate(char *argv[])
26 {
27 #pragma GCC diagnostic ignored "-Wunused-result"
28         FILE *output_file = fopen(argv[2], "r");
29         FILE *answer_file = fopen(argv[3], "r");
30
31         int p_ins, p_sub, p_del;
32         fscanf(answer_file, "%d %d %d", &p_ins, &p_sub, &p_del);
33
34         int correct = 0, i;
```

```
35          for (i = 0; i < N; i++)
36          {
37                  int input[N], answer[N];
38                  char data[N];
39                  if (fscanf(output_file, "%d", &input[i]) == EOF)
40                          break;
41                  fscanf(answer_file, "%d %s", &answer[i], data);
42                  if (input[i] == answer[i])
43                          correct++;
44          }
45
46          printf("%d\n", correct * 100 - ask_count * 5);
47          // printf("%d/%d Correct.\n", correct, i);
48          // printf("Score: %d\n", correct * 100 - ask_count * 5);
49          // printf("Time: %lf seconds\n", (double)compute_time / CLOCKS_PER_SEC);
50          // printf("Ask count: %d\n", ask_count);
51          // printf("Abort count: %d\n", abort_count);
52          // printf("\n");
53
54          fclose(output_file);
55          fclose(answer_file);
56  #pragma GCC diagnostic warning "-Wunused-result"
57  }
58  #endif
59
60  #pragma region BITPARALLEL
61  int weighted_levenshtein_bitpal(char *a, char len_a, char *b, int len_b)
62  {
63          if (len_a > 64)
64          {
65                  return weighted_levenshtein_bitpal(a, 64, b, len_b > 64 ? 64 : len_b);
66          }
67
68          unsigned long long posbits[256] = {0};
69
70          for (int i = 0; i < len_a; i++)
71          {
72                  posbits[(unsigned char)a[i]] |= 1ull << i;
73          }
74
```

```
75          unsigned long long DHneg1 = ~0x0ull;
76          unsigned long long DHzero = 0;
77          unsigned long long DHpos1 = 0;
78
79          // recursion
80          for (int i = 0; i < len_b; i++)
81          {
82                  unsigned long long Matches = posbits[(unsigned char)b[i]];
83                  // Complement Matches
84                  unsigned long long NotMatches = ~Matches;
85
86                  // Finding the vertical values.
87                  // Find 1s
88                  unsigned long long INITpos1s = DHneg1 & Matches;
89                  unsigned long long DVpos1shift = (((INITpos1s + DHneg1) ^ DHneg1) ^ INIT
90
91                  // set RemainingDHneg1
92                  unsigned long long RemainDHneg1 = DHneg1 ^ (DVpos1shift >> 1);
93                  // combine 1s and Matches
94                  unsigned long long DVpos1shiftorMatch = DVpos1shift | Matches;
95
96                  // Find 0s
97                  unsigned long long INITzeros = (DHzero & DVpos1shiftorMatch);
98                  unsigned long long DVzeroshift = ((INITzeros << 1) + RemainDHneg1) ^ Rem
99
100                 // Find -1s
101                 unsigned long long DVneg1shift = ~(DVpos1shift | DVzeroshift);
102                 DHzero &= NotMatches;
103                 // combine 1s and Matches
104                 unsigned long long DHpos1orMatch = DHpos1 | Matches;
105                 // Find 0s
106                 DHzero = (DVzeroshift & DHpos1orMatch) | (DVneg1shift & DHzero);
107                 // Find 1s
108                 DHpos1 = (DVneg1shift & DHpos1orMatch);
109                 // Find -1s
110                 DHneg1 = ~(DHzero | DHpos1);
111         }
112         // find scores in last row
113         unsigned long long add1 = DHzero;
114         unsigned long long add2 = DHpos1;
```

```
115
116         int dist = len_b;
117
118         for (int i = 0; i < len_a; i++)
119         {
120                 unsigned long long bitmask = 1ull << i;
121                 dist -= ((add1 & bitmask) >> i) * 1 + ((add2 & bitmask) >> i) * 2 - 1;
122         }
123
124         return dist;
125 }
126 #pragma endregion
127
128 int predict_answer(const int index, char *answer_file, const int length, int *ids, const
129 {
130         int ans_id = -1;
131         int min_distance = INT_MAX;
132         int multiple = 0;
133         int ans_ids[N] = {0};
134         const int step = length / 10.0;
135         for (int j = 0; j < k; j++)
136         {
137                 int id = ids[j];
138                 for (int i = 0; i < DATA_LENGTH; i += step)
139                 {
140                         static char temp[N + 1];
141                         strncpy(temp, S[id] + i, length);
142                         temp[length] = '\0';
143                         int distance = weighted_levenshtein_bitpal(temp, length, q, leng
144                         if (distance < min_distance)
145                         {
146                                 min_distance = distance;
147                                 ans_id = id;
148                                 multiple = 0;
149                                 ans_ids[0] = id;
150                         }
151                         if (distance == min_distance && ans_id != id)
152                         {
153                                 if (ans_ids[multiple] != id)
154                                 {
```

```
155                                             multiple++;
156                                             ans_ids[multiple] = id;
157                                     }
158                             }
159                             if (distance < length / 4.0)
160                             {
161 #ifdef EVALUATE_MODE
162                                     abort_count++;
163 #endif
164                                     return ans_id + 1;
165                             }
166                     }
167             }
168             if (multiple)
169             {
170                     free(q);
171 #ifdef EVALUATE_MODE
172                     ask_count++;
173 #endif
174                     q = ask(index + 1, answer_file);
175                     return predict_answer(index, answer_file, strlen(q) + 1, ans_ids, multi
176             }
177             return ans_id + 1;
178 }
179
180 int main(int argc, char *argv[])
181 {
182 #pragma GCC diagnostic ignored "-Wunused-result"
183 #pragma region INITIALIZE
184 #ifdef EVALUATE_MODE
185             compute_time = clock();
186 #endif
187             srand((unsigned int)time(NULL));
188             FILE *input_file = fopen(argv[1], "r");
189             FILE *output_file = fopen(argv[2], "w");
190             FILE *answer_file = fopen(argv[3], "r");
191             int ids[N];
192             for (int i = 0; i < N; i++)
193             {
194                     ids[i] = i;
```

```
195         }
196
197         if (!input_file || !output_file || !answer_file)
198         {
199                 fprintf(stderr, "error\n");
200                 exit(EXIT_FAILURE);
201         }
202
203         fscanf(input_file, "%d %d %d", &p_ins, &p_sub, &p_del);
204
205         S = (char **)malloc(sizeof(char *) * N);
206
207         for (int i = 0; i < N; i++)
208         {
209                 S[i] = (char *)malloc(sizeof(char) * (DATA_LENGTH + 1));
210                 fscanf(input_file, "%s", S[i]);
211         }
212 #pragma endregion
213
214         for (int i = 0; i < Q; i++)
215         {
216                 q = malloc(sizeof(char) * (N + 1));
217                 fscanf(input_file, "%s", q);
218                 int length = strlen(q) + 1;
219
220                 int answer = predict_answer(i, argv[3], length, ids, N);
221                 free(q);
222
223                 fprintf(output_file, "%d\n", answer);
224         }
225
226 #pragma region FINALIZE
227         fclose(input_file);
228         fprintf(output_file, "%lf\n", clock() / (double)CLOCKS_PER_SEC);
229         fclose(output_file);
230         fclose(answer_file);
231         for (int i = 0; i < N; i++)
232         {
233                 free(S[i]);
234         }
```

```
235         free(S);
236 #pragma endregion
237 #pragma GCC diagnostic warning "-Wunused-result"
238
239 #ifdef EVALUATE_MODE
240         compute_time = clock() - compute_time;
241         evaluate(argv);
242 #endif
243
244         return 0;
245 }
```

また、提出したソースコードにはないが、時間の確認に使った動的計画法、O(ND) アルゴリズム、O(NP) アルゴリズムのソースコードについても以下に記載する。

```
1  #define max(a,b) (((a) > (b)) ? (a) : (b))
2  #define min(a,b) (((a) < (b)) ? (a) : (b))
3
4  int edit_distance_dp(char *a, int len_a, char *b, int len_b)
5  {
6      static int d[101][101];
7
8      for (int i = 0; i < len_a + 1; i++) d[i][0] = i;
9      for (int i = 0; i < len_b + 1; i++) d[0][i] = i;
10     for (int i = 1; i < len_a + 1; i++)
11         for (int j = 1; j < len_b + 1; j++)
12             d[i][j] = min(min(d[i-1][j], d[i][j-1]) + 1, d[i-1][j-1] + (a[i-1] == b[j-1]
13
14     return d[len_a][len_b];
15 }
16
17 int edit_distance_ond(char *a, int len_a, char *b, int len_b)
18 {
19     static int V[201];
20     int x, y;
21     int offset = len_a;
22     V[offset + 1] = 0;
23
24     for (int D = 0; D <= len_a + len_b; D++) {
25         for (int k = -D; k <= D; k += 2) {
26             if (k == -D || k != D && V[k-1+offset] < V[k+1+offset]) x = V[k+1+offset];
```

```
27              else x = V[k-1+offset] + 1;
28              y = x - k;
29              while (x < len_a && y < len_a && a[x] == b[y]) {
30                  x++;
31                  y++;
32              }
33              V[k+offset] = x;
34              if (x >=  len_a && y >= len_b) return D;
35          }
36      }
37
38      return -1;
39  }
40
41  static int snake(int k, int y, char *a, int len_a, char *b, int len_b)
42  {
43      int x = y - k;
44
45      while (x < len_a && y < len_b && a[x] == b[y]) {
46          x++;
47          y++;
48      }
49
50      return y;
51  }
52
53  int edit_distance_onp(char *a, int len_a, char *b, int len_b)
54  {
55      // required: s1->size() <= s2->size()
56      char** s1 = len_a > len_b ? &b : &a;
57      char** s2 = len_a > len_b ? &a : &b;
58          const int s1_size = len_a > len_b ? len_b : len_a;
59          const int s2_size = len_a > len_b ? len_a : len_b;
60      static int fp[201];
61      int x, y, k, p;
62      int offset = s1_size + 1;
63      int delta = s2_size - s1_size;
64      for (int i = 0; i < 201; i++) fp[i] = -1;
65
66      for (p = 0; fp[delta + offset] != s2_size; p++) {
```

```
67          for(k = -p; k < delta; k++)
68              fp[k + offset] = snake(k, max(fp[k-1+offset] + 1, fp[k+1+offset]), *s1, s1_s
69          for(k = delta + p; k > delta; k--)
70              fp[k + offset] = snake(k, max(fp[k-1+offset] + 1, fp[k+1+offset]), *s1, s1_s
71          fp[delta + offset] = snake(delta, max(fp[delta-1+offset] + 1, fp[delta+1+offset]
72      }
73
74      return delta + (p - 1) * 2;
75  }
```

参考文献

[1] "bitparallel weighted Levenshtein distance". Stackoverflow. https://stackoverflow.com/questions/65363769/bitparallel-weighted-levenshtein-distance (参照 2022-1-25)

[2] "C++: 編集距離を求めるアルゴリズム". 良いもの。悪いもの。 http://handasse.blogspot.com/2009/04/c_29.html (参照 2022-1-25)