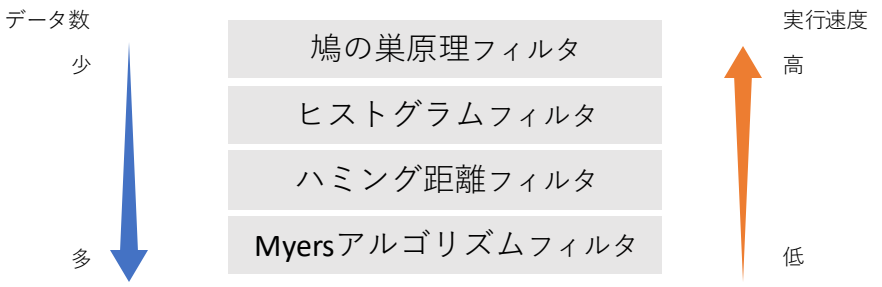
アルゴリズムとデータ構造B Group 1

1W242023 伊藤悟 / 1W242030 井上大阿 / 1W242038 植木敬太郎 / 1W242065 岡村圭吾 / 1W242339 保科智哉

Group1が採用した手法と流れ

1班では、**計算コストの異なる4つの手法を階層的に組み合わせることで100%の精度と高速化の両立**を達成した。

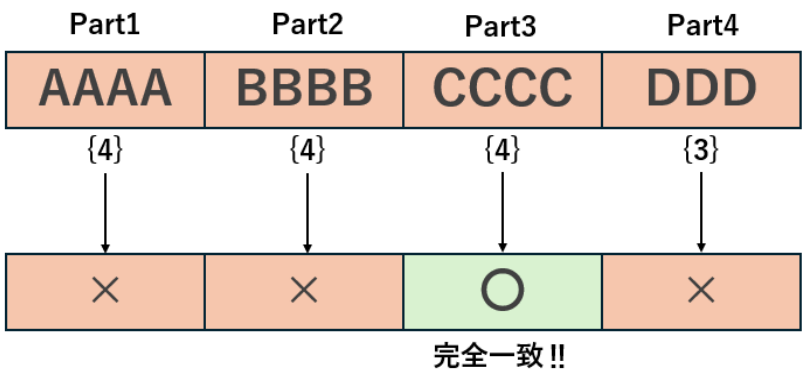
また、prep.cにおける索引データ作成では上記の4つの手法に対応するようにデータを処理した。さらに、索引データの読み込み時の計算をなくすことによって高速化を図った。



鳩の巣原理による探索空間の削減

編集距離K以内の類似データを抽出する際、対象の文字列をK+1個のパーツに分割することで、「**少なくとも1つのパーツはエラーを全く含まない**」**という鳩の巣原理**を適用した。

4分割とスライドによる絞り込み 15文字を4区間に分割し、いずれかの区間で完全一致があるもののみを正解候補とする。置換に加え、挿入・削除による位置のズレを考慮し、検索時にクエリの分割位置を前後1文字スライドさせて参照する手法を導入した。



索引のデータ構造

prep.cにおける索引データ作成では以下の点を用いた。

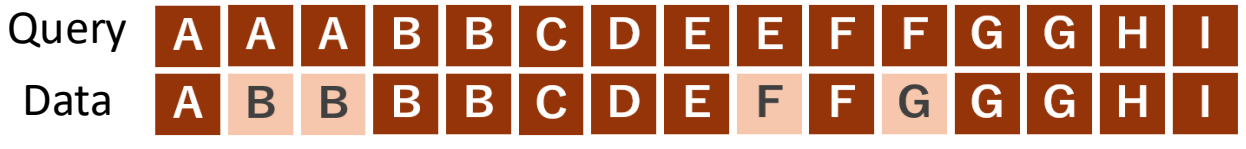
- **鳩の巣原理**に基づく4つの分割領域それぞれに対し、**計数ソート**を用いた10,000個のバケット分類
  - 文字列パターンは 10^4 通りしかない点に着目
- **計算なしで直接参照**できる最適化済みデータ構造
  - `fwrite` を使って、検索時に `mmap` でそのままメモリに展開できるバイナリ形式で出力
  - 通常、ファイルを読み込むときは`read()`を用いてファイルからプログラムのメモリへデータをコピーする。しかし、100万件ものデータを扱うことから、ファイルをメモリ上の仮想アドレスに直接マッピングする`mmap()`を使用した。

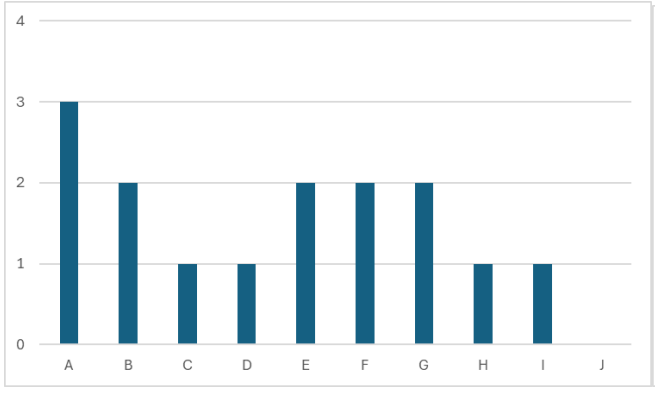
ヒストグラムフィルタ

■ 原理

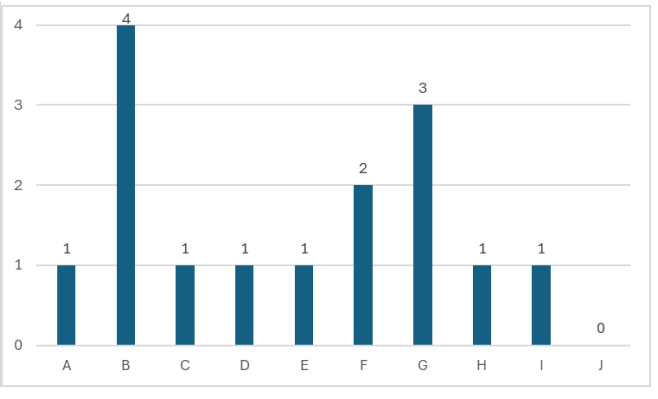
編集距離k以内で変換可能な文字列同士は、**各文字の出現回数の差の合計が2kを超えない。**
→ 差の合計が6を超えたら候補から除外可能

Ex)
- **置換(A→B)** : Aが1減りBが1増える→差の合計は**2**変動
- **挿入/削除** : 1文字増減→差の合計は**1**変動





Query Histogram



Data Histogram

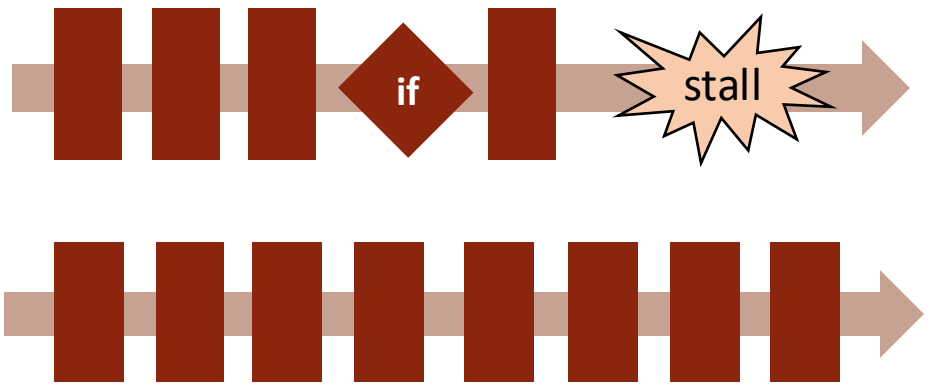
文字	クエリ	データ	差分
A	3	1	2
B	2	4	2
C	1	1	0
D	1	1	0
E	2	1	1
F	2	2	0
G	2	3	1
H	1	1	0
I	1	1	0
J	0	0	0
差分の合計			**6**

ヒストグラムの出現回数の差分の合計が6

→6を超えていないため，この場合は除外されない

if文を排除することで分岐予測ミスを無くす

If文を一切含まないビット演算のみで構成する。
これにより、CPUパイプラインの実行停止を引き起こす**分岐予測ミスが0**になる。データの内容に依らず、常に一定かつきわめて短いクロックサイクルで実行が完了する。



ハミング距離フィルタ

■ 原理
ハミング距離(同じ位置で文字が異なる箇所の数)がk以下であれば、挿入・削除の編集距離もk以下である。(十分条件)

■ 文字列の64bitキャスト
メモリ上の文字列を64bit整数としてキャストし、**XOR**を適用した。
これにより**1命令で最大8文字分の一致・不一致を判定**することが可能。

■ POPCNT命令による不一致ビットの集計
CPUのハードウェア命令である**POPCNT**を呼び出す__builtin_popcontllを活用した。**ビットが立っている箇所の合計を定数時間でカウントできる。**

ビット並列アルゴリズムによる検証 (Myers Algorithm)

通常の動的計画法(DP)は表計算が必要である。

動的計画法（DP）による編集距離計算を論理演算に置き換え、**64bit整数**として並列に処理することで、検証プロセスの圧倒的な高速化を実現した。15文字の文字列であれば、これらを15ビットの整数（uint16_tやuint64_t）として保持する。各ビットを編集距離の増分として扱い、加算と論理和（AND/OR）の組み合わせにより、全セルの状態を同時に更新する。また、この方式により、計算過程をCPUレジスタ上の論理演算のみで完結させ、メモリアクセスのボトルネックを排除した。

- **VP (Vertical Positive)**: 上のセルより値が +1 される場合に 1 が立つビット列
- **VN (Vertical Negative)**: 上のセルより値が -1 される場合に 1 が立つビット列

通常のDP表

初期の文字列

編集後の文字列		A	B	C	A	B	B	A
	0	1	2	3	4	5	6	7
C	1	1	2	2	3	4	5	6
B	2	2	1	2	3	3	4	5
A	3	2	2	2	2	3	4	4
B	4	3	2	3	3	2	3	4
A	5	4	3	3	3	3	3	3
C	6	5	4	3	4	4	4	4
A	7	6	5	4	3	4	5	4

O(mn)

m × n のグリッドを作成し、左上から右下へセルを埋めていく。
全てのセルを計算するため、処理時間はO(mn)。
長いテキストやパターンでは致命的に遅くなる。

Myers AlgorithmにおけるVP, VN, 編集距離の遷移表

初期の文字列　：ABCABBA
編集後の文字列：CBABACA

段階	VP	VN	編集距離
初期	1111111	0000000	7
1	1111011	0000000	6
2	1101101	0000010	5
3	1101100	0000001	4
4	1001101	0010011	4
5	1111000	0000011	3
6	1110001	0001111	4
7	1110101	1001111	4

O(n)

コンピュータのワード長(通常64bit)単位で列全体の更新をO(1)で実行。
そのため、クエリ長mがワード長w以下であれば、計算量はテキスト長nのみに比例するO(n)に低減される。

ループ展開

ループ処理において、CPU内部では「カウントを増やす」「15未満か判定する」「先頭に戻る」という**計算そのものとは無関係な命令が毎回実行されている。**
そのため、文字列長が15で固定であることを利用し、**コンパイラ任せにせず、明示的にマクロを用いて処理を15回書き並べた。**
例) STEP(0); STEP(1); ... STEP(14);

出力バッファリング

検索結果の出力において、1文字ごとに `printf` や `putc` を呼ぶとシステムコール呼び出しのオーバーヘッドが無視できなくなる。本実装では `out_buffer` (64KB) を用意し、**結果を一度メモリ上に溜め込んでから `fwrite` でまとめて出力する自作の `fast_put` 関数を実装**した。I/Oボトルネックの解消に寄与している。