文書に最適化されたキー配列の生成

▼ 研究の成果

アルファベットからなるドキュメントに最適化されたキー配列の生成を行うプログラムを作成した。

生成されたキー配列は、最適化に用いた文書と同類の文書に対して高い性能を発揮した。

▼ 研究用に開発したプログラム

https://github.com/yuichiro-kurose/keyboard

情報分野　２年８組１１番　黒瀬　諭一朗

▼目次

１．研究の背景：キーボードが抱える課題

各キーの使用頻度（日本語 -QWERTY）

Q W E R T Y U I O P
A S D F G H J K L _
Z X C V B N M _ _ _

各キーの使用頻度（英語 -QWERTY）

Q W E R T Y U I O P

A S D F G H J K L _

Z X C V B N M _ _ _

２．先行研究：大西配列

各キーの使用頻度（日本語 - 大西配列）

Q	L	U	_	_	F	W	R	Y	P
E	I	A	O	_	K	T	N	S	H
_	Z	X	C	V	G	D	M	J	B

大西配列の制作過程

1．片手の**連続使用**を減らす ▶ 左右交互打鍵で高速化

2．指の**移動**を減らす ▶ 速く楽にタイピング

3．指の**連続使用**を減らす ▶ 流れるようにタイピング

この過程に沿って最適なキー配列を作成するプログラムを作ることを目指す

３．研究方法：プログラムの開発

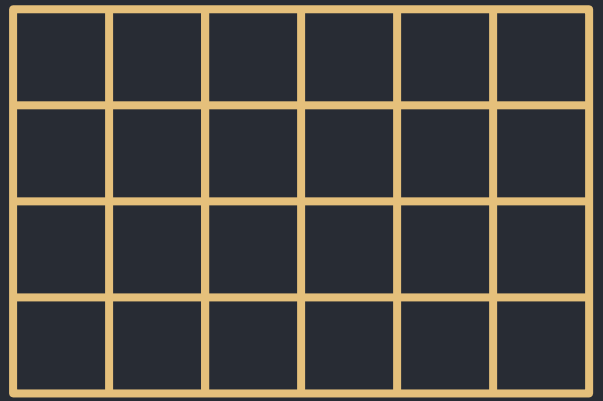
プログラムの完成像

ドキュメントを入力 ▶▶▶ キー配列が出てくる

```
Hey, my name is
Yuichiro Kurose.
```

```
#include <iostream>

int main() {
  std::cout << “Hello World” << std::endl;
  Return 0;
}
```



プログラムの概要

１．右手と左手の担当キーを決定（工夫が必要）

２．押しやすい場所に使用頻度の高いキーを配置

３．同じ指の連続使用を回避

４．完成したキー配列を出力

１．右手と左手の担当キーを決定

キーを右手と左手に分ける方法は、

$$_{26}C_{13} = 10400600 \fallingdotseq 10^7 \text{（通り）}$$

片手を連続使用した回数が最小となる分け方を見つけるためには、それぞれの分け方における片手の連続回数を数える必要がある。

これを愚直にやると、 10^7N 回の処理が必要。

これだと $N = 10^6$ のとき、普通のパソコンでは**約 1 日**の時間を要する。

※ ホームリーダー 10 冊で、約 10^6 文字

片手の連続回数を事前に処理することで効率化を図る

頂点集合を {a,b,...,z} とする重み付き完全無向グラフ K_{26} を考える。

ただし、

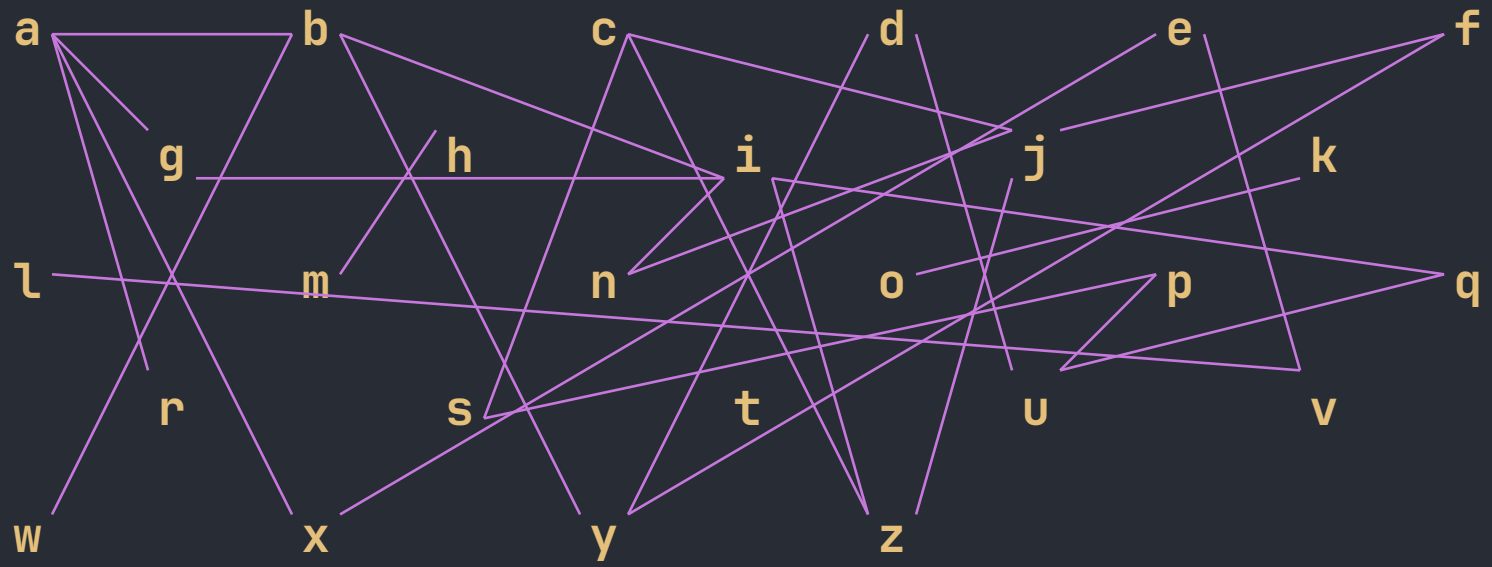
w(u,v) = ドキュメント内で「uv」または「vu」が現れる回数

とする。

（例）ドキュメントが「Yuichiro Kurose」のとき、

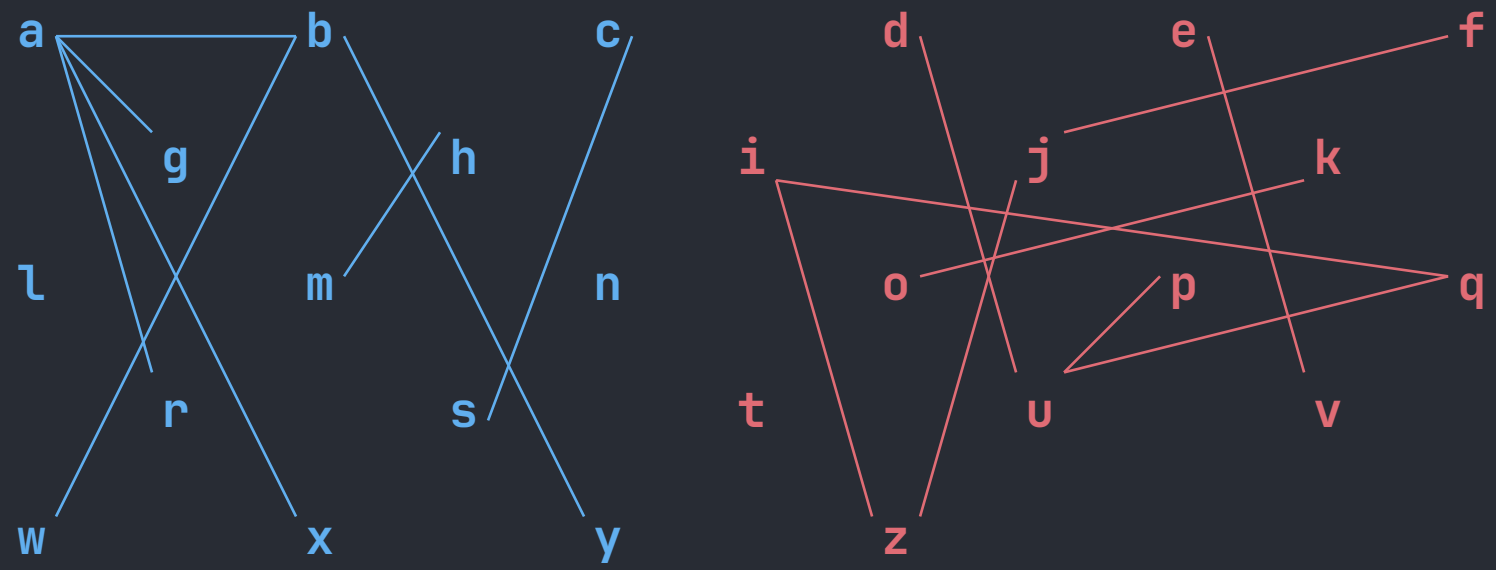
w(a,b) = 0, w(k,o) = 1, w(o,r) = 2

K_{26} のイメージ図



（実際は全頂点間が結ばれている）

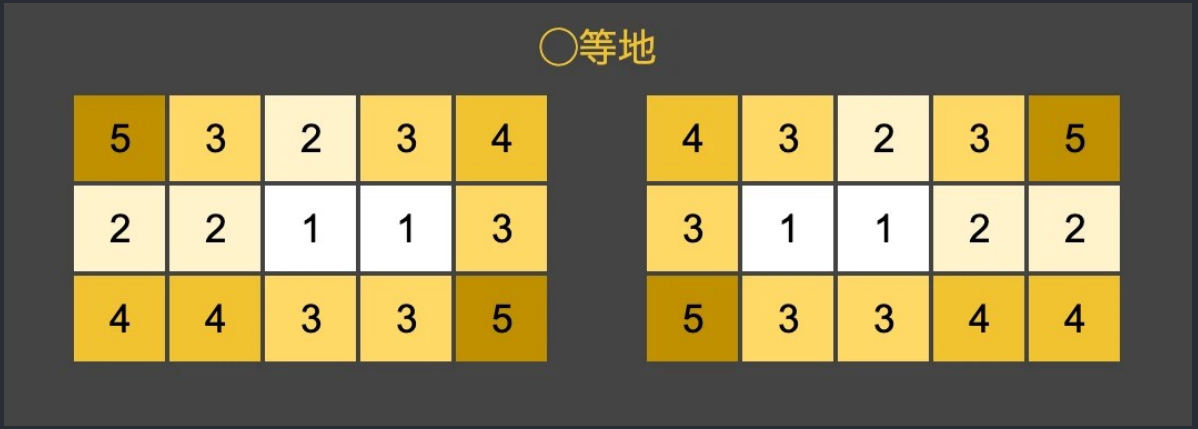
片手の連続使用回数が最小となる分け方は、各グループ内の辺の重みの合計が最小となるように K_{26} を分割する方法に対応する。



（実際は各グループ内の全頂点間が結ばれている）

２．押しやすい場所に使用頻度の高いキーを配置

押しやすさの評価には、大西氏が開発した「◯等地」を使用。◯等地によって定義されたコストを最小化することで、押しやすい場所に使用頻度の高いキーを配置することを実現。



４．結果：実行結果の評価

評価の手順

1. A Christmas Carol in Prose の序盤（約 30000 文字）を用いて、キー配列を生成
2. The Strange Case of Dr. Jekyll and Mr. Hyde の序盤（約 10000 文字）を用いて、 QWERTY と比較

出力結果

```
=== Splitting Keys ===
Hand 0 (Left) letters: a c e i j k o q t u x y z
Hand 1 (Right) letters: b d f g h l m n p r s v w

=== Placing Keys ===
--- Hand 0 (Optimal Layout) ---
_ u o c z
a i e t y
q j x k _
-------------------------------------------------
--- Hand 1 (Optimal Layout) ---
v g h w _
m s n r d
_ f l b p
-------------------------------------------------
```

QWERTY のスコア

Target Document Length: 9814 characters
1. Hand movement cost (Algorithm Definition): 4691
2. One-hand consecutive usage count (Actual): 4691
3. Finger movement cost (Algorithm Definition): 1033
4. Same finger consecutive usage count (Actual): 1033
5. Difficulty in pressing: 16189

太郎（仮）のスコア

Target Document Length: 9814 characters
1. Hand movement cost (Algorithm Definition): 3361
2. One-hand consecutive usage count (Actual): 3361
3. Finger movement cost (Algorithm Definition): 630
4. Same finger consecutive usage count (Actual): 630
5. Difficulty in pressing: 9791

各キーの使用頻度（英語 - 太郎（仮））

_	O	U	C	Z	V	G	H	W	_
A	I	E	T	Y	M	S	N	R	D
Q	J	X	K	_	_	F	L	B	P

5．考察

QWERTY と太郎（仮）の比較

同じ手の連続使用率	47.8%	▶	34.2%
同じ指の連続使用率	10.5%	▶	6.4%

```
Q W E R T Y U I O P          _ O U C Z V G H W _
A S D F G H J K L _    ▶     A I E T Y M S N R D
Z X C V B N M _ _ _          Q J X K _ _ F L B P
```

最適化の対象でない文書に対しても高性能

6. 結論・展望

結論:

- アルファベットからなるドキュメントに最適化されたキー配列の生成を行うプログラムを作成した。
- 生成されたキー配列は、最適化に用いた文書と同類の文書に対して高い性能を発揮した。

課題:

- 実際に人間にとって使いやすいかは不明。
- QWERTY からの乗り換えが困難である可能性が高い。

展望:

- 文書をアルファベットに変換するプログラムを用いれば、アルファベットでない文書にも対応できる可能性が高い。
- 記号に対応できれば、プログラミングの効率化が期待できる。

7．参考文献

[1] 大西拓磨「ローマ字入力に最適なキー配列を考える（制作編）」 https://note.com/illlillliliilill/n/n3b51f4aaf086

[2] Charles Dickens「A Christmas Carol in Prose」 https://www.gutenberg.org/cache/epub/46/pg46.txt

[3] Robert Louis Stevenson「The Strange Case of Dr. Jekyll and Mr. Hyde」 https://www.gutenberg.org/cache/epub/43/pg43.txt

[4] 秋葉拓哉・岩田陽一・北川宜稔『プログラミングコンテストチャレンジブック』株式会社 マイナビ出版

```cpp
std::vector<std::vector<int>> AssignLettersToHands(
    const std::vector<std::vector<int>>& bigram_weights) {
  int min_conflict_cost = INT_MAX;
  int optimal_hand_bitmask = (1 << kNumHandKeys) - 1;
  int current_hand_bitmask = (1 << kNumHandKeys) - 1;

  while (current_hand_bitmask < (1 << kNumAlphabet)) {
    int current_cost = CalcHandConflictCost(current_hand_bitmask, bigram_weights);
    if (current_cost < min_conflict_cost) {
      min_conflict_cost = current_cost;
      optimal_hand_bitmask = current_hand_bitmask;
    }

    // Gosper's Hack
    int x = current_hand_bitmask & -current_hand_bitmask;
    int y = current_hand_bitmask + x;
    current_hand_bitmask = ((current_hand_bitmask & ~y) / x >> 1) | y;
  }

  std::vector<std::vector<int>> hand_assignments(2);
  for (int i = 0; i < kNumAlphabet; i++) {
    int hand_index = (optimal_hand_bitmask >> i) & 1;
    hand_assignments[hand_index].push_back(i);
  }
  return hand_assignments;
}

// @brief 指ごとの連続使用コスト（同じ指の文字ペアの重み総和）を計算する
int CalcFingerContinuityCost(
    const std::vector<std::vector<int>>& finger_letter_assignments,
    const std::vector<std::vector<int>>& bigram_weights) {
  int total_cost = 0;
  for (const auto& letters_for_this_finger : finger_letter_assignments) {
    for (size_t i = 0; i < letters_for_this_finger.size(); ++i) {
```