アルファベット文字の登場頻度について

・よく使わる文字・組み合わせの頻度を測り，スコア化

・よく出る組み合わせが多い文＝スコアが高い文になるように暗号文を解読する
・暗号化する人は，文字の頻度が一定となり解読されないようにする必要がある

- 1文字あたりの頻度

- 5文字あたりの頻度

Monogram Frequencies

English single letter frequencies are as follows (in percent %):

A : 8.55	K : 0.81	U : 2.68
B : 1.60	L : 4.21	V : 1.06
C : 3.16	M : 2.53	W : 1.83
D : 3.87	N : 7.17	X : 0.19
E : 12.10	O : 7.47	Y : 1.72
F : 2.18	P : 2.07	Z : 0.11
G : 2.09	Q : 0.10	
H : 4.96	R : 6.33	
I : 7.33	S : 6.73	
J : 0.22	T : 8.94	

Quintgram Frequencies

These are 5-gram frequencies for English. We can't list all of them here, the top 30 are the following (in percent %):

OFTHE : 0.18	ANDTH : 0.07	CTION : 0.05
ATION : 0.17	NDTHE : 0.07	WHICH : 0.05
INTHE : 0.16	ONTHE : 0.07	THESE : 0.05
THERE : 0.09	EDTHE : 0.06	AFTER : 0.05
INGTH : 0.09	THEIR : 0.06	EOFTH : 0.05
TOTHE : 0.08	TIONA : 0.06	ABOUT : 0.04
NGTHE : 0.08	ORTHE : 0.06	ERTHE : 0.04
OTHER : 0.07	FORTH : 0.06	IONAL : 0.04
ATTHE : 0.07	INGTO : 0.06	FIRST : 0.04
TIONS : 0.07	THECO : 0.05	WOULD : 0.04

http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/

暗号化の仕組み（英語の例）①

置換表の作成

- 英語の母音等，頻度が高い文字を多めに暗号文字を割り当てる．（但し最低１英字1数字とする）

・英字以外の文字種（数字文字，改行文字，スペースを含む）は削除する．ただし大段落は維持

```python
def generate_random_frequencies(max_value):
    # 一般的な，英字の頻度に基づく重み付けを下記に定義する
    weights = {
        'A': 8.55, 'B': 1.60, 'C': 3.16, 'D': 3.87, 'E': 12.10, 'F': 2.18, 'G': 2.09, 'H': 4.96,
        'I': 7.33, 'J': 0.22, 'K': 0.81, 'L': 4.21, 'M': 2.53, 'N': 7.17, 'O': 7.47, 'P': 2.07,
        'Q': 0.10, 'R': 6.33, 'S': 6.73, 'T': 8.94, 'U': 2.68, 'V': 1.06, 'W': 1.83, 'X': 0.19,
        'Y': 1.72, 'Z': 0.11
    }

    # 処理1．まず全文字に最低「1個」ずつ割り当てる
    # max_valueとは，置換表で使用する数値の最大値を指す
    counts = {char: 1 for char in weights.keys()}

    # 処理2．全体の置換文字総数（0からmax_valueまでなので +1）
    total_slots = max_value + 1

    # 処理3．残りの残数を計算
    remaining_slots = total_slots - len(counts)

    # 処理4．重みに基づいて、残りの枠をランダムに分配
    # chars: 文字リスト，probs: 重みリスト,weightsは重みづけ配列
    chars = list(weights.keys())
    probs = list(weights.values())

    if remaining_slots > 0: # 残り枠がある場合処理
        # （重みづけ付き）ランダムに選んで割り当て
        extra_allocations = random.choices(chars, weights=probs, k=remaining_slots)
        for char in extra_allocations:
            counts[char] += 1

    return counts #countsの形は配列，{'A': 5, 'B': 2, ...}
```

暗号化の仕組み（英語の例）②

置換表の作成

- 例はアルファベット26文字を50文字と100文字に置換した例である．（平文は同一）
- 右の画像は以後の解読時に用いた暗号文である
- 今回は数字だが必要な種類があるキャラクタなら何でもよい（片仮名なら50種など）
- 大文字小文字は区別しないものとする．

```
A : 079,041,047,035,081,026,010,059,054,082,055,086
B : 069,089,063
C : 068,044
D : 028,058,050,029
E : 087,001,095,016,039,094,020,092,060,048
F : 075,067,065,031
G : 078,042,037,014
H : 088,085,093,062,084,100,049,032
I : 064,003,030,036,076,034,043
J : 072,052
K : 066,008
L : 019,025,056,011
M : 005,040,038
N : 022,097,000,007
O : 045,051,070,009
P : 015,096,083
Q : 024
R : 090,023,033,013,061
S : 012,098,006,080
T : 091,057,046,071
U : 018,021,099
V : 027
W : 004,002
X : 053,077
Y : 074,073
Z : 017[EOF]
```

```
A : 041,048,030
B : 008,000
C : 050,049
D : 014
E : 032,020,002,026
F : 036,037
G : 004
H : 016,043
I : 044,038,047,035
J : 015
K : 018
L : 046
M : 009
N : 028,031,011,013
O : 022,042,003,034
P : 007
Q : 005
R : 039,021
S : 045,024
T : 025,001,006,010
U : 017
V : 012,033
W : 023
X : 019
Y : 040,029
Z : 027[EOF]
```

暗号化の仕組み（英語の例）③

暗号文の作成

- 置換表を複数個作成し，それをさらにランダムに選択し，平文を1文字ずつ変換する．

```python
def generate_homophonic_table(max_value):

    # 処理1: 各文字に何個割り当てるかを決定
    frequencies = generate_random_frequencies(max_value)
    # 0からmax_valueまでの数値リストを作成
    available_numbers = list(range(max_value + 1))
    # ランダム性を担保するためにシャッフル
    random.shuffle(available_numbers)
    cipher_table = {}
    # 処理2: 決定した個数分だけ数値を割り当てる
    for char, count in frequencies.items():
        homophones = []
        for _ in range(count):
            if available_numbers:
                num = available_numbers.pop()
                # 3桁埋め（000 ~ 200）で統一
                homophones.append(f"{num:03d}")

        # 安全装置: 万が一空ならランダム値（通常ここは通りません）
        # 例えばmax_valueが50の場合，51個割り当てようとするとここに来る
        if not homophones:
            homophones.append(f"{random.randint(0, max_value):03d}")

        cipher_table[char] = homophones

    return cipher_table
```

暗号化の仕組み（英語の例）④

置換前の平文(ゲティスバーグ演説，264単語1452文字)

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate-we can not consecrate-we can not hallow-this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us-that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion-that we here highly resolve that these dead shall not have died in vain-that this nation, under God, shall have a new birth of freedom-and that government of the people, by the people, for the people, shall not perish from the earth.

置換後の暗号文

ゲティスバーグ平文utf8.txt

3FF9B27A.txt

出典：渡部純，ゲティスバーグ演説の訳し方，明治学院大学，2013年1月

置換表を用いた復号化プログラム①

- 置換表が入手できる場合、暗号文に現れるキャラクター（今回の場合は0~50の数字）を、表に当てはめて１語１語置換し解読してく。
- 暗号文には大文字小文字等の情報は含まれていない為、このプログラムではすべて大文字としている。
- 第一引数は暗号文，第二引数は置換表とする．

```python
def load_decryption_table(filename):
    table = {}
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            if ':' in line:
                char, nums_part = line.split(':', 1)
                for num in nums_part.strip().split(','):
                    if num.strip():
                        table[num.strip()] = char.strip()
    return table

def decrypt_text(ciphertext, table):
    result, i = [], 0
    while i < len(ciphertext):
        chunk = ciphertext[i:i+3]
        if len(chunk) == 3 and chunk.isdigit() and chunk in table:
            result.append(table[chunk])
            i += 4 if i+3 < len(ciphertext) and ciphertext[i+3] == ',' else 3
        else:
            result.append(ciphertext[i])
            i += 1
    return "".join(result)

def main():
    import sys
    table = load_decryption_table(sys.argv[2])
    with open(sys.argv[1], 'r', encoding='utf-8') as f:
        ciphertext = f.read()
    with open('decrypted.txt', 'w', encoding='utf-8') as f:
        f.write(decrypt_text(ciphertext, table))

if __name__ == "__main__":
    main()[EOF]
```

置換表を用いた復号化プログラム②

暗号前

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate-we can not consecrate-we can not hallow-this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us-that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion-that we here highly resolve that these dead shall not have died in vain-that this nation, under God, shall have a new birth of freedom-and that government of the people, by the people, for the people, shall not perish from the earth.

置換表

```
A : 041,048,030
B : 008,000
C : 050,049
D : 014
E : 032,020,002,026
F : 036,037
G : 004
H : 016,043
I : 044,038,047,035
J : 015
K : 018
L : 046
M : 009
N : 028,031,011,013
O : 022,042,003,034
P : 007
Q : 005
R : 039,021
S : 045,024
T : 025,001,006,010
U : 017
V : 012,033
W : 023
X : 019
Y : 040,029
Z : 027[EOF]
```

復号結果

FOUR SCORE AND SEVEN YEARS AGO OUR FATHERS BROUGHT FORTH ON THIS CONTINENT A NEW NATION CONCEIVED IN LIBERTY AND DEDICATED TO THE PROPOSITION THAT ALL MEN ARE CREATED EQUAL.

NOW WE ARE ENGAGED IN A GREAT CIVIL WAR TESTING WHETHER THAT NATION OR ANY NATION SO CONCEIVED AND SO DEDICATED CAN LONG ENDURE. WE ARE MET ON A GREAT BATTLE-FIELD OF THAT WAR. WE HAVE COME TO DEDICATE A PORTION OF THAT FIELD AS A FINAL RESTING PLACE FOR THOSE WHO HERE GAVE THEIR LIVES THAT THAT NATION MIGHT LIVE. IT IS ALTOGETHER FITTING AND PROPER THAT WE SHOULD DO THIS. BUT IN A LARGER SENSE WE CAN NOT DEDICATE-WE CAN NOT CONSECRATE-WE CAN NOT HALLOW-THIS GROUND. THE BRAVE MEN LIVING AND DEAD WHO STRUGGLED HERE HAVE CONSECRATED IT FAR ABOVE OUR POOR POWER TO ADD OR DETRACT. THE WORLD WILL LITTLE NOTE NOR LONG REMEMBER WHAT WE SAY HERE BUT IT CAN NEVER FORGET WHAT THEY DID HERE. IT IS FOR US THE LIVING RATHER TO BE DEDICATED HERE TO THE UNFINISHED WORK WHICH THEY WHO FOUGHT HERE HAVE THUS FAR SO NOBLY ADVANCED. IT IS RATHER FOR US TO BE HERE DEDICATED TO THE GREAT TASK REMAINING BEFORE US-THAT FROM THESE HONORED DEAD WE TAKE INCREASED DEVOTION TO THAT CAUSE FOR WHICH THEY GAVE THE LAST FULL MEASURE OF DEVOTION-THAT WE HERE HIGHLY RESOLVE THAT THESE DEAD SHALL NOT HAVE DIED IN VAIN-THAT THIS NATION UNDER GOD SHALL HAVE A NEW BIRTH OF FREEDOM-AND THAT GOVERNMENT OF THE PEOPLE BY THE PEOPLE FOR THE PEOPLE SHALL NOT PERISH FROM THE EARTH.

89B485CA.txt

アルファベット文字の登場頻度について（再掲）

・よく使わる文字・組み合わせの頻度を測り，スコア化

・よく出る組み合わせが多い文＝スコアが高い文になるように暗号文を解読する
・暗号化する人は，文字の頻度が一定となり解読されないようにする必要がある

- 1文字あたりの頻度
- 5文字あたりの頻度

Monogram Frequencies

English single letter frequencies are as follows (in percent %):

A : 8.55	K : 0.81	U : 2.68
B : 1.60	L : 4.21	V : 1.06
C : 3.16	M : 2.53	W : 1.83
D : 3.87	N : 7.17	X : 0.19
E : 12.10	O : 7.47	Y : 1.72
F : 2.18	P : 2.07	Z : 0.11
G : 2.09	Q : 0.10	
H : 4.96	R : 6.33	
I : 7.33	S : 6.73	
J : 0.22	T : 8.94	

Quintgram Frequencies

These are 5-gram frequencies for English. We can't list all of them here, the top 30 are the following (in percent %):

OFTHE : 0.18	ANDTH : 0.07	CTION : 0.05
ATION : 0.17	NDTHE : 0.07	WHICH : 0.05
INTHE : 0.16	ONTHE : 0.07	THESE : 0.05
THERE : 0.09	EDTHE : 0.06	AFTER : 0.05
INGTH : 0.09	THEIR : 0.06	EOFTH : 0.05
TOTHE : 0.08	TIONA : 0.06	ABOUT : 0.04
NGTHE : 0.08	ORTHE : 0.06	ERTHE : 0.04
OTHER : 0.07	FORTH : 0.06	IONAL : 0.04
ATTHE : 0.07	INGTO : 0.06	FIRST : 0.04
TIONS : 0.07	THECO : 0.05	WOULD : 0.04

http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/

解読プログラム①

頻度解析について

- 今回作成したプログラムは、暗号文が事前に同音字暗号であることを前提とし解読を行う。

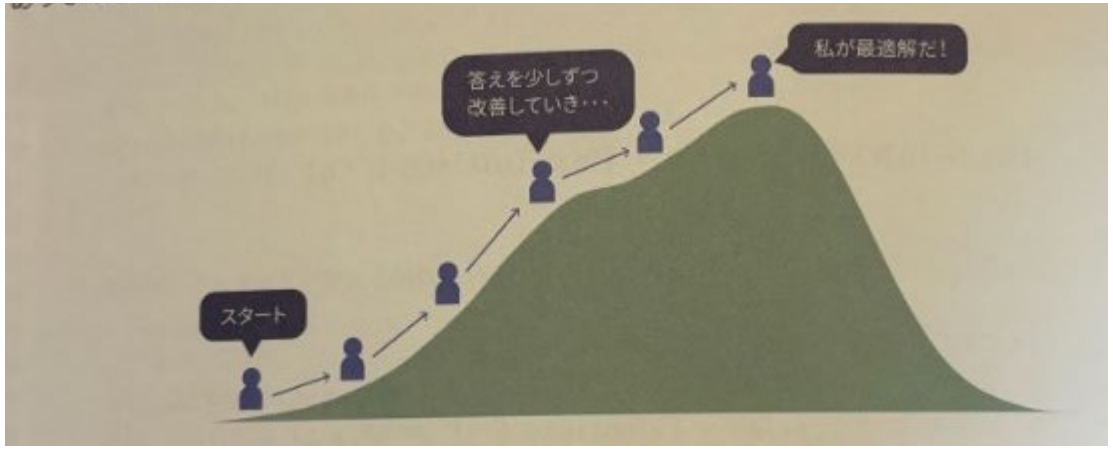
↓

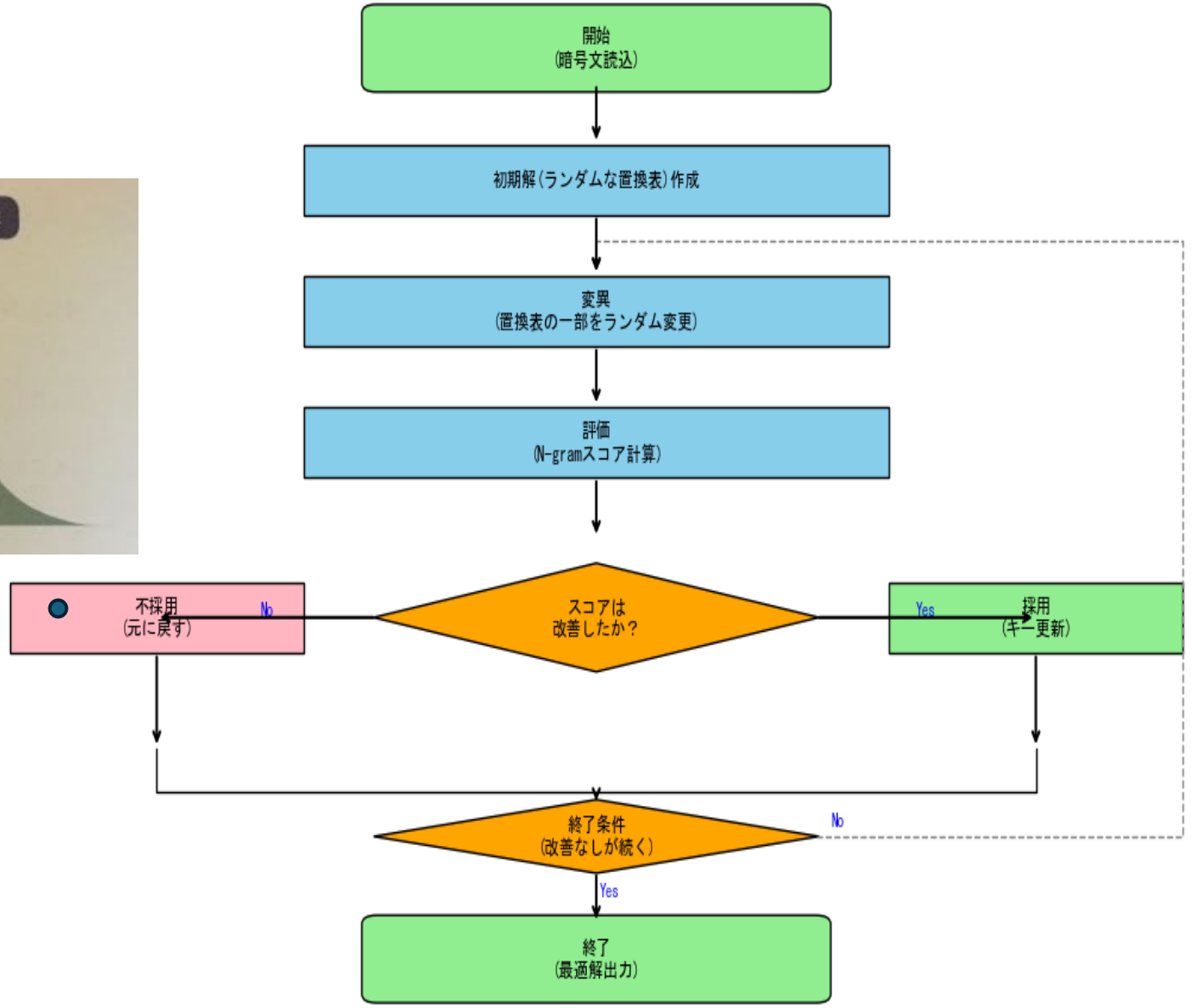
- そのため、予め暗号文が同音字暗号か否かは前述の方法を用いて判定を行う。

- 一定の暗号文字（今回は５文字）ごとに頻度解析を行う．これを1つずつ右へずらしながら1サイクルごとに，スコアを加算する．
- 暗号文字の区切りはコンマまたはスペースである．（今回作成したプログラムはスペースとカンマを区別しておらず，単語の境目は検知していない）

解読プログラム②

ヒルクライム法について



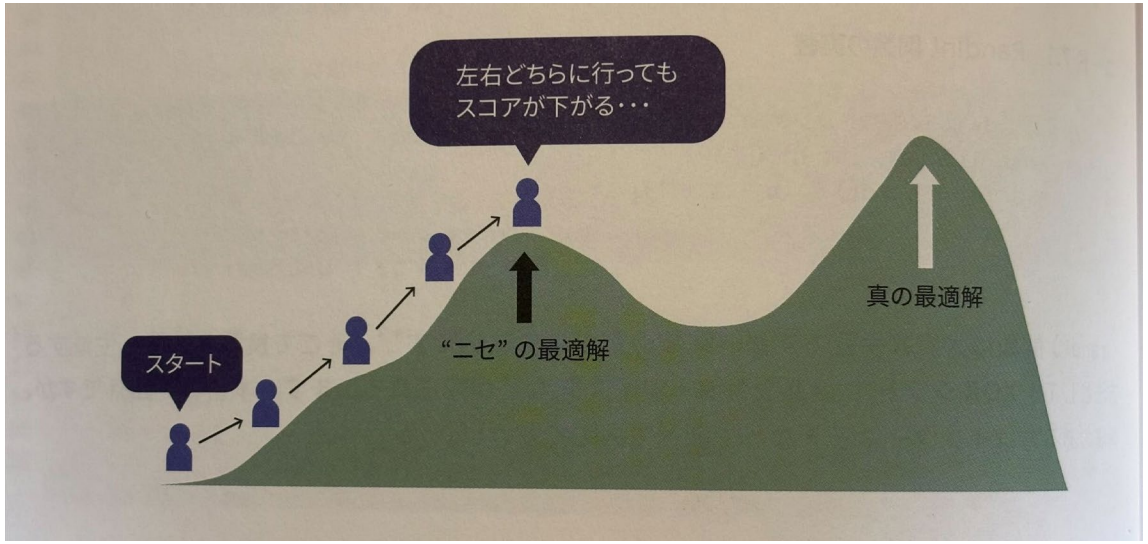


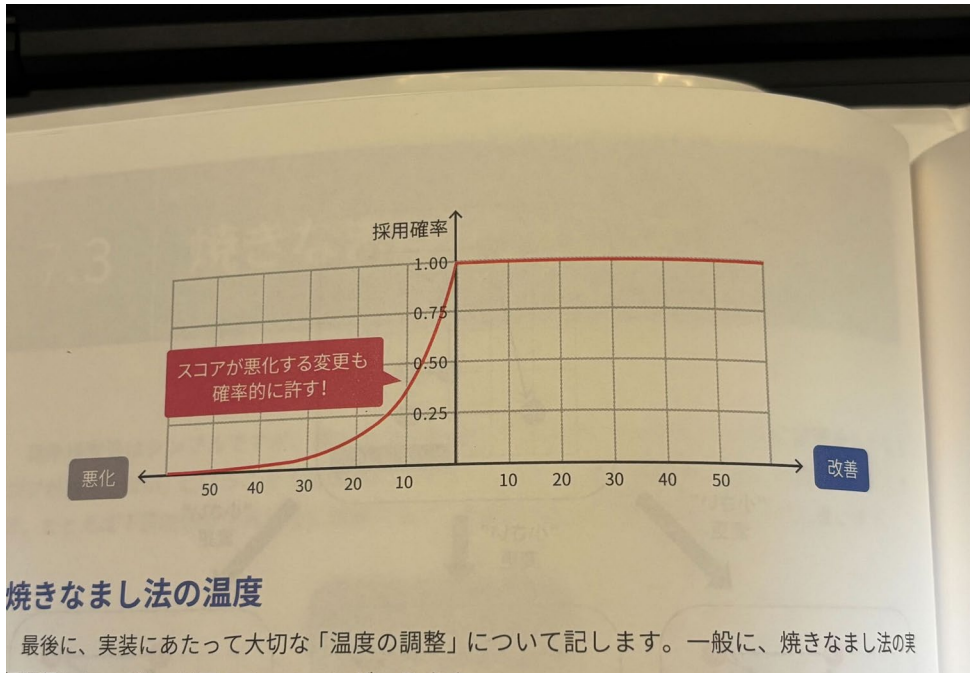
焼きなまし法では，スコアが改善しなくても，一定確率で採用し，局所解に陥ることを防ぐ

解読プログラム③

焼きなまし法における解読プログラムについて

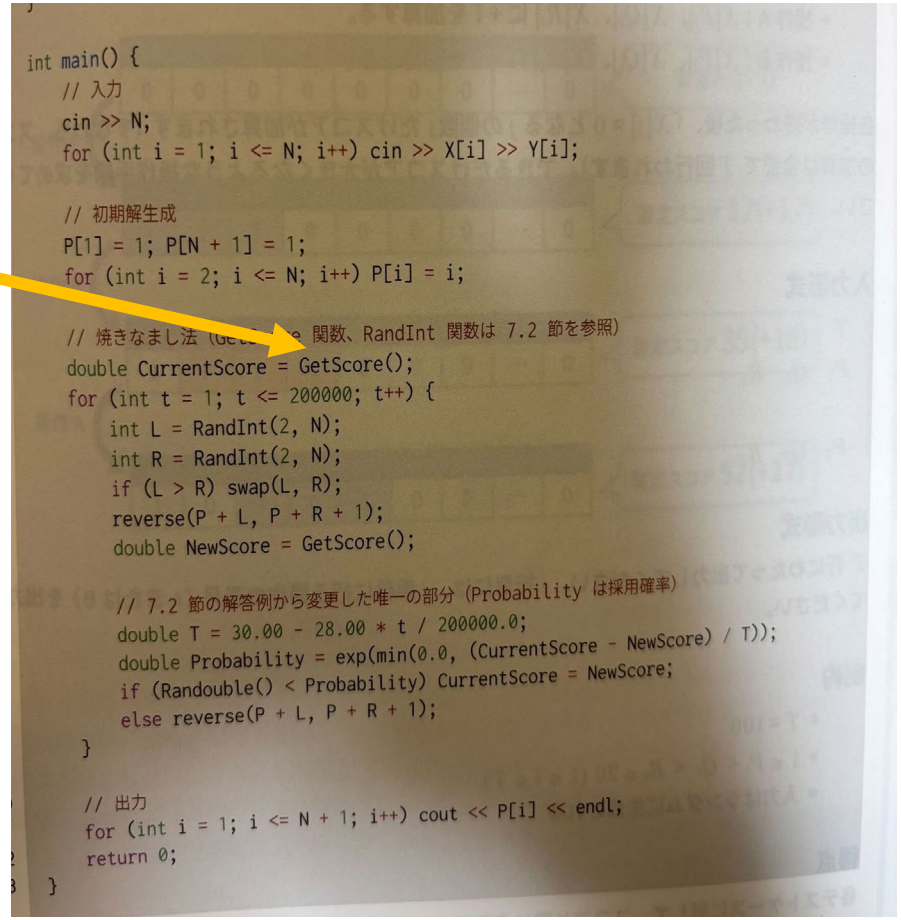
- ヒルクライム法を発展させ局所解に陥りにくい焼きなまし法を採用
- 文の途中での改行に対処（段落を削除）し，段落ごとに頻度解析
- マルチスレッドによる数十回の試行を行い最適解を求める





解読プログラム④

- 文章スコア
- $S = \sum_{i=0}^{L-N} \log_{10}(P(w_i))$
- S: 算出されるスコア
  ここに数式を入力します。(対数をとっているため負値)
- L: 復号されたテキストの全長
- N: 計算対象$(N-gram)$の長さ
  (今回のプログラムでは5)
- w_i: テキストのi番目から始まるN文字の単語列
  $(N-gram)$
- $P(w_i)$: 計算対象の 5 文字$(N-gram)$
  が英語統計データ内で出現する確率



```
int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> X[i] >> Y[i];

    // 初期解生成
    P[1] = 1; P[N + 1] = 1;
    for (int i = 2; i <= N; i++) P[i] = i;

    // 焼きなまし法 (Ge      関数、RandInt 関数は 7.2 節を参照)
    double CurrentScore = GetScore();
    for (int t = 1; t <= 200000; t++) {
        int L = RandInt(2, N);
        int R = RandInt(2, N);
        if (L > R) swap(L, R);
        reverse(P + L, P + R + 1);
        double NewScore = GetScore();

        // 7.2 節の解答例から変更した唯一の部分 (Probability は採用確率)
        double T = 30.00 - 28.00 * t / 200000.0;
        double Probability = exp(min(0.0, (CurrentScore - NewScore) / T));
        if (Randouble() < Probability) CurrentScore = NewScore;
        else reverse(P + L, P + R + 1);
    }

    // 出力
    for (int i = 1; i <= N + 1; i++) cout << P[i] << endl;
    return 0;
}
```

米田 優峻, 競技プログラミングの鉄則: アルゴリズム力と思考力を高める77の技術, マイナビ出版, 2022.

実行画面

ans.txt

同音字暗号.exe

同音字暗号プログラム.zip

解読結果（置換表）

JやZなどは
登場頻度が少なく
精度が悪い

焼き直し法の結果.zip

単語数	264
文字数	1452
Temp	50
Step per Temp	5000
C_Rate	0.98
試行回数	32
処理時間	27分

ヒルクライム法

```
A : 030, 041, 048
B : 000
C : 049, 050
D : 005, 014
E : 002, 020, 026, 032, 040
F : 036, 037
G : 004
H : 016, 043
I : 035, 038, 044, 047
L : 046
M : 009
N : 011, 013, 028, 031
O : 003, 022, 034, 042
P : 007
R : 021, 039
S : 024, 029, 045
T : 001, 006, 008, 010, 018, 023, 025
U : 017
V : 012, 033
```

正答

```
A : 041,048,030
B : 008,000
C : 050,049
D : 014
E : 032,020,002,026
F : 036,037
G : 004
H : 016,043
I : 044,038,047,035
J : 015
K : 018
L : 046
M : 009
N : 028,031,011,013
O : 022,042,003,034
P : 007
Q : 005
R : 039,021
S : 045,024
T : 025,001,006,010
U : 017
V : 012,033
W : 023
X : 019
Y : 040,029
Z : 027[EOF]
```

焼きなまし法

```
A : 030, 041, 048
B : 000, 008
C : 049, 050
D : 014
E : 002, 020, 026, 032
F : 036, 037
G : 004
H : 016, 043
I : 035, 038, 044, 047
L : 046
M : 009
N : 011, 013, 028, 031
O : 003, 022, 034, 042
P : 007
Q : 005
R : 021, 039
S : 024, 045
T : 001, 006, 010, 018, 025
U : 017
V : 012, 033
W : 023
Y : 029, 040
```