**„Wiadomość biała”**

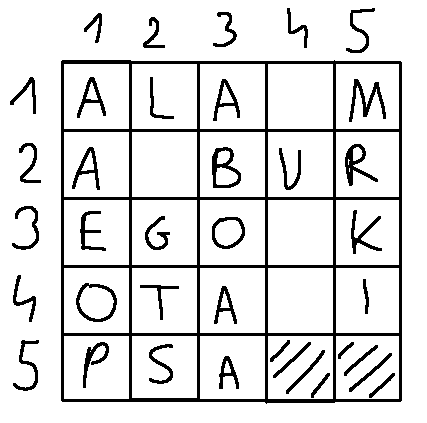
# opis zadania

Cywilizacja Binarjanów do zapisu tekstu używa tylko dwóch znaków: spacji oraz tabulatorów poziomych (**\t**) - są to znaki białe, bo standardowo nie mają reprezentacji graficznej.

Przy tłumaczeniu języka binarjańskiego na języki ziemskie trzeba wykorzystać poniższe informacje:

1. odpowiednia ilość spacji odpowiada konkretnej literze angielskiego alfabetu, zatem najkrótszy ciąg spacji składa się z 1 spacji, a najdłuższy z 26 spacji (nie ma innych znaków)
2. n spacji jest odpowiednikiem n-tej litery alfabetu, czyli:  
   1 spacja = ‘a’, 2 spacje = ‘b’, 3 spacje = ‘c’, …, 26 spacji = ‘z’
3. po każdym ciągu spacji znajduje się tabulator, więc linia:  
   (1 spacja)(tabulator)(2 spacje)(tabulator)(3 spacje)(tabulator) oznacza „abc”
4. w ten sposób w jednej linii można zapisać tylko jeden wyraz, zatem aby zapisać wielowyrazową informację, wiadomość musi mieć wiele linii - kolejne linie to kolejne wyrazy
5. pusta linia (nie zawiera ani spacji, ani tabulatorów) w języku ziemskim oznacza pojedynczą spację; Uwaga! takich pustych linii może być wiele z rzędu
6. wiadomość znajduje się w pliku „***wiadomosc.txt***”

Dodatkowo, Binarjanie zawsze szyfrują swoją wiadomość, ale szyfr został złamany. Jest to szyfr podstawieniowy na podstawie kwadratu o boku równym wartości klucza k. Kwadrat tworzy siatkę pól. Algorytm szyfrujący polega na wpisaniu kolejnych liter niezaszyfrowanej wiadomości od lewej do prawej, najpierw do 1. wiersza, potem do 2. itd. do k-tego wiersza, a szyfrogram powstaje poprzez odczytywanie liter od góry do dołu najpierw w 1. kolumnie, potem 2. kolumnie itd. do k-tej kolumny. Przykład obrazujący zaszyfrowanie wiadomości „ALA MA BUREGO KOTA I PSA” dla , czyli kwadratu o bokach 5x5:



* Uzyskany szyfrogram to: „AAEOPL\_GTSABOAA\_U\_\_MRKI” (znak ‘\_’ reprezentuje spację)
* Jak widać, wiele spacji może znaleźć się obok siebie
* Uwaga! zwróć uwagę, że do szyfrogramu nie trafiają „niewykorzystane” pola kwadratu (zakreskowane na rysunku)

Wartość klucza trzeba najpierw obliczyć na podstawie danych w pliku „***klucz.txt***”.

Wzór na klucz to: , gdzie:

* **a** - ilość liczb będących **dokładnymi palindromami**  
  dokładny palindrom, to wyrażenie brzmiące dokładnie tak samo przy czytaniu od lewej do prawej i na odwrót (od prawej do lewej); przykłady:
  + 10001 jest palindromem dokładnym (od tyłu to 10001, czyli to samo)
  + 1 jest palindromem dokładnym
  + 1**01**1 nie jest palindromem dokładnym, bo od tyłu to:  
    1**10**1 - nie pasują tu dwie cyfry
* **b** - ilość liczb będących **przybliżonymi palindromami**, gdzie dokładnie 1 cyfra powoduje odstępstwo od dokładnego palindromu, np.:
  + 1**01**1 to od tyłu 1**10**1 - zalicza się do tej grupy, bo ma dokładnie 1 cyfrę niepasującą do dokładnego palindromu (palindrom dokładny wyglądał by tak: 1001 lub 1111)
  + **111**0**000** to od tyłu **000**0**111** nie zalicza się do tej grupy, bo ma aż 3 błędne znaki; Uwaga! zauważ, że nie usunięto wiodących zer
  + 1001 nie zalicza się do tej grupy, ponieważ jest palindromem dokładnym i ma 0 niepasujących cyfr, a warunkiem jest dokładnie 1 niepasująca cyfra

# wejście - specyfikacja plików wejściowych

***wiadomosc.txt***

* w pierwszym wierszu znajduje się liczba w systemie dziesiętnym **n** () określająca ilość pozostałych linii do wczytania z pliku (każda linia może reprezentować wyraz poprzedzony spacją lub tylko spację)
* w kolejnych **n** wierszach tylko spacje i tabulatory poziome (oraz znak nowej linii na końcu każdego wiersza)
* po każdym ciągu spacji jest dokładnie 1 tabulator

***klucz.txt***

* w pierwszym wierszu znajduje się liczba w systemie dziesiętnym **n** () określająca ilość liczb w poniższych wierszach do wczytania
* w kolejnych wierszach **n** liczb zapisanych w systemie binarnym (jak na Binarjan przystało)
* Uwaga! przy sprawdzaniu, czy liczby są palindromami nie należy konwertować ich do innych systemów liczbowych; podane liczby zawsze powinny być reprezentowane w systemie binarnym

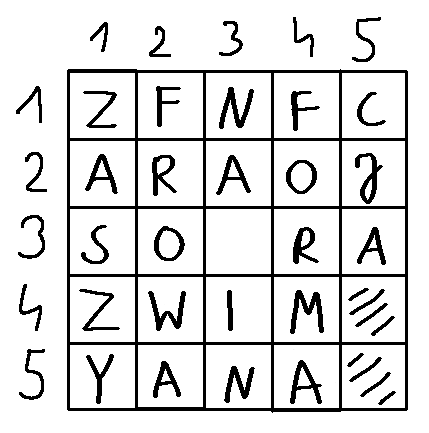
# wyjście

W pliku „**wynik.txt**”: 1 wiersz zawierający odszyfrowaną wiadomość.

# przykład

Rozwiązanie dla danych z plików „***przyklad-wiadomosc.txt***” i „***przyklad-klucz.txt***”:

1. z pliku „*przyklad-wiadomosc.txt*” zamieniamy spacje i tabulatory na litery angielskiego alfabetu
   1. ilość wierszy do wczytania to 2 - informacja ta znajduje się w pierwszym wierszu
   2. na początku kolejnego wiersza znajduje się 26 spacji i tabulator, co tłumaczy się na ‘z’, dalej 6 spacji i tabulator, co oznacza f itd. z kolejnymi ciągami spacji
   3. po odczytaniu całego wiersza przechodzimy do kolejnego wiersza - należy wstawić jedną spację oraz przetłumaczyć kolejne ciągi spacji
   4. otrzymany zaszyfrowany tekst to: „ZFNFCARAOJSO RAZWIMYANA”
2. z pliku „*przykład-klucz.txt*” obliczamy klucz k
   1. ani palindrom dokładny, ani przybliżony: 10101010
   2. palindromy dokładne: tylko 10101, zatem
   3. palindromy przybliżone: 10, 100, 1000, 10000, 100000, 1000000; razem 6, zatem
3. teraz należy odszyfrować wiadomość korzystając z kwadratu 5x5 (bo )
   1. rysujemy kwadrat
   2. zaznaczamy „wolne” pola w odwrotny sposób niż przy szyfrowaniu (jeśli są wolne pola, to zaznacza się je od ostatniej kolumny od dołu do góry, itd.) - to ważny krok, aby poprawnie odczytać informację; można wykorzystać zależność:
   3. do kolejnych pól (od lewej do prawej, najpierw w 1. wierszu, potem w kolejnych) wpisujemy kolejne znaki zaszyfrowanego napisu (nie można wpisywać do zakreskowanych pól)



* 1. Wiadomość odszyfrowaną należy odczytywać od góry do dołu, zaczynając od 1. kolumny, pomijając zakreskowane pola; otrzymujemy: „ZASZYFROWANA INFORMACJA”

1. napis „ZASZYFROWANA INFORMACJA” umieszczamy w pliku „*wynik.txt*”

# rozwiązanie wzorcowe - kod

poniższy kod jest również zawarty w pliku „***wiadomosc-biala.cpp***”

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

using namespace std;

// 0 -> not palindrome at all; 1 -> exact palindrome; 2 -> approximate palindrome

int isExactOrApproximatePalindrome(string text, int errors = 1) {

int length = text.length();

int errorsCounter = 0;

for (int i = 0; i < length / 2; i++) {

if (text[i] != text[length - i - 1]) {

errorsCounter++;

}

if (errorsCounter > errors) {

return 0;

}

}

if (errorsCounter == 0) {

return 1;

}

else if (errorsCounter == errors) {

return 2;

}

}

void display2DVector(vector<vector<char>> vec, int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cout << "\'" << vec[i][j] << "\'" << "\t";

}

cout << endl;

}

cout << endl;

}

string decipherTransposition(string encrypted, int key) {

int length = encrypted.length();

int lackingToSquare = key \* key - length;

string answer = "";

cout << "encrypted: \"" << encrypted << "\"" << endl;

cout << "lacking to square: " << lackingToSquare << endl << endl;

vector<vector<char>> chars;

for (int i = 0; i < key; i++) {

vector<char> line(key);

chars.push\_back(line);

}

int col = key - 1; // column index

int line = key - 1; // line index

int counter = 0;

for (int i = 0; i < lackingToSquare; i++) {

if (col == -1) {

line--;

col = key - 1;

counter = 0;

}

chars[col--][line] = '\*';

}

int index = 0;

for (int i = 0; i < key; i++) {

for (int j = 0; j < key; j++) {

if (index < length and chars[i][j] != '\*') {

chars[i][j] = encrypted[index++];

}

}

}

display2DVector(chars, key);

for (int i = 0; i < key; i++) {

for (int j = 0; j < key; j++) {

if (chars[j][i] != '\*') {

answer += chars[j][i];

}

}

}

return answer;

}

int main()

{

fstream messageFile, keyFile, outputFile;

messageFile.open("wiadomosc.txt", ios::in);

keyFile.open("klucz.txt", ios::in);

outputFile.open("wynik.txt", ios::out);

int linesQuantity;

string encrypted = "";

int numsQuantity;

int exactPalindromesCounter = 0; // a

int approximatePalindromesCounter = 0; // b

int key; // k

string answer;

if (messageFile.good() and keyFile.good() and outputFile.good()) {

// wiadomosc.txt

string line;

messageFile >> linesQuantity;

getline(messageFile, line); // to move to next line in the file (skip number)

for (int i = 0; i < linesQuantity; i++) {

getline(messageFile, line);

int lineLength = line.length();

int counter = 0;

if (lineLength > 0) {

for (int j = 0; j < lineLength; j++) {

if (line[j] == ' ') {

counter++;

}

else if (line[j] == '\t') {

encrypted += (char)(64 + counter); // 65 -> 'A' in ASCII

counter = 0;

}

}

if (i + 1 < linesQuantity) {

encrypted += ' ';

}

}

else {

encrypted += ' ';

}

}

cout << "encrypted: \"" << encrypted << "\"" << endl;

// klucz.txt

keyFile >> numsQuantity;

for (int i = 0; i < numsQuantity; i++) {

string num;

keyFile >> num;

int isPalindrome = isExactOrApproximatePalindrome(num);

cout << num << ":" << isPalindrome << " ";

if (isPalindrome == 1) {

exactPalindromesCounter++;

}

else if (isPalindrome == 2) {

approximatePalindromesCounter++;

}

}

cout << endl << endl;

cout << "a = " << exactPalindromesCounter << "\tb = " << approximatePalindromesCounter << endl;

key = approximatePalindromesCounter - exactPalindromesCounter; // b - a

cout << "k = " << approximatePalindromesCounter << " - " << exactPalindromesCounter << " = " << key << endl << endl;

answer = decipherTransposition(encrypted, key);

cout << "wynik: " << endl << "\"" << answer << "\"" << endl;

outputFile << answer;

}

else {

cout << "Problem z plikiem wejsciowym/wyjsciowym";

}

messageFile.close();

keyFile.close();

outputFile.close();

}