# [Анализ графа статей из Википедии](http://cs.mipt.ru/cpp/labs/lab6.html)

## Работа с файлами С++

Для работы с файловыми потоками в C++, необходимо подключить библиотеку <fstream>. В данной библиотеке определены такие файловые потоки, как ifstream — поток, предназначенный для чтения из файла, ofstream — поток для записи и fstream — поток, позволяющий как читать из файла, так и записывать в него. Работа с файловыми потоками абсолютно аналогична работе с потоками cin и cout.

Пример программы, копирующей файл text.txt в copy.txt:

#include <fstream>

#include <string>

#include <iostream>

int main(){

std::ifsteam f\_in("text.txt");

std::ofstream f\_out("copy.txt");

std::string str\_data;

while ( std::getline(f\_in, str\_data) ){

f\_out << str\_data << std::endl;

}

f\_in.close();

f\_out.close();

return 0;

}

## Национальные алфавиты в С++

Работа в С++ с национальными алфавитами в консоли достаточно затруднительна. Существуют различные варианты:

1. Использовать национальную 8-битную кодировку (например KOI8-R);
2. Использовать кодировку UTF-8;
3. Использовать "широкие" строки;

Широкие строки (std::wstring, std::u16string, std::u32string) — строки, в который один символ занимает не один (char) байт, а два (wchar\_t, char16\_t) или четыре (char32\_t). Для инициализации начального значения некоторой строкой, перед ней необходимо использовать префикс L для wchar\_t строк, u для char16\_t строк и U для char32\_t строк.

Для вывода std::wstring, существует поток std::wcout, но посмотрите, как будет работать следующая программа:

#include <string>

#include <iostream>

int main(){

std:: wcout << L"¾" << std::endl;

return 0;

}

Такая программа выведет не один символ ¾, а целых три — 3/4. Дело в том, что заставить консоль нормально воспринимать такой вывод — отдельная задача, а использовать два потока вывода (std::cout и std::wcout) в программе нельзя. Похожая ситуация с 8-битными национальными кодировками.

Поэтому наиболее универсальным методом для работы с национальными алфавитами — использование "широких" строк, для посимвольной работы, а перед выводом в cout --перекодировать в UTF-8 строку. Устройство кодировки UTF-8 можно прочитать в [статье](https://ru.wikipedia.org/wiki/UTF-8). На основании изложенных там материалов можно написать функцию преобразования "широкой" строки (std::wstring) к обычной (std::string), но уже закодированной в UTF-8:

std::string utf16\_to\_utf8 (std::wstring str){

std::string result = ""; // начальное значение для

// результирующей строки

for (int i=0; i < str.length(); ++i){ // Проходим по всем символам

uint16\_t curr\_char = str[i];

if (curr\_char < 0x80) // для кодирования в UTF-8 нужен один байт

{

result.push\_back(curr\_char);

}

else if (curr\_char < 0x800) // два байта

{

result.push\_back(0xC0 | (curr\_char >> 6));

result.push\_back(0x80 | (curr\_char & 0x3F));

}

else // три байта

{

result.push\_back(0xC0 | (curr\_char >> 12));

result.push\_back(0x80 | ((curr\_char >> 6 )& 0x3F));

result.push\_back(0x80 | (curr\_char & 0x3F));

}

}

return result; // возвращаем результат

}

Пример программы:

#include <iostream>

#include <string>

int main(){

std::cout << utf16\_to\_utf8(L"Привет, Мир!!!") << std::endl;

return 0;

}

Стоит отметить, что обычно при работе с файлами в Linux системе, зачастую созданные файлы сохранются именно в кодировке UTF-8. В таком случае, следующая программа будет работать аналогично предыдущей:

#include <iostream>

#include <string>

int main(){

std::string str = "Привет, Мир!!!";

std::cout << str << std::endl;

return 0;

}

При этом важно знать, что длинна str.length() будет равна не 14 а 23 (зависит от компилятора). То есть, если Вам дана UTF-8 строка (например, прочитали из файла) и её необходимо посимвольно обработать (например заменить все буквы ё на е), тогда преобразуйте её в "двухбайтную" (или "четырёхбайтную") строку, проведите все необходимые преобразования, а перед выводом преобразуйте назад в UTF-8. Преобразование из UTF-8 в выглядит следующим образом:

std::wstring utf8\_to\_utf16 (std::string str){

uint8\_t first\_byte, addition\_byte;

std::wstring result = L""; // строка с результатом работы функции

int i = 0;

while (i < str.length()){ // Пока не пройдём все символы

first\_byte = str[i]; // Запоминаем первый байт текущего символа

result.push\_back(first\_byte); // и помещаем в строку результата

if (first\_byte & 0x80){ // если кодировка символа занимает больше одного байта,

first\_byte = first\_byte << 1;// то удалем первую единицу сдвигом

}

uint32\_t cl\_int = 0x3F; // маска для "затирания" начальные единицы

while (first\_byte & 0x80){ // пока не удалим все начальные единицы в 'first\_byte'

addition\_byte = str[++i]; // Считываем очередной байт кодировки

// Изменяем последний символ строки с результатом

int last = result.length()-1;

// затираем первую единицу и дописываем 6 последних бит дополнительного байта

result[last] = ((result[last] & cl\_int) << 6 ) | (addition\_byte & 0x3F);

cl\_int = (cl\_int << 7) | 0xFF;

first\_byte = first\_byte << 1;// удаляем начальную единицу в 'first\_byte'

}

++i; //следующий символ

}

return result; // возвращаем результат

}

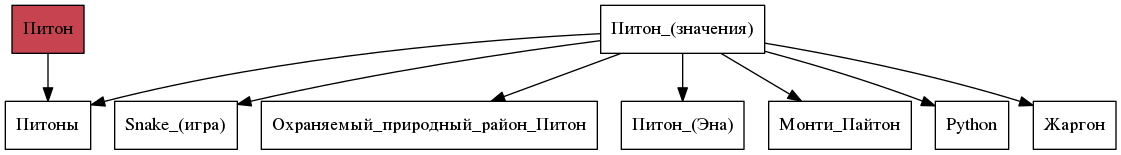
P.S. Аналогичные преобразования можно делать при помощи #include <codecvt>. Для работы UTF-8 вывода в консоли windows — набрать chcp 65001 или mode con cp select=65001.

## Структура графа статей

Граф статей Википедии является ориентированным и имеет следующую структуру:

* вершинами графа являются статьи;
* ребро соединяет вершины A и B, если в статье A есть ссылка на статью B.

При этом фатически ссылки бывают двух типов: обычные ссылки в тексте статьи и ссылки из статей с перенаправлением. На рисунке ниже приведён пример такого графа:



Красным цветом отмечена статья с перенаправлением — если открыть эту статью в браузере, то вместо адреса https://ru.wikipedia.org/wiki/Питон в адресной строке браузера мы увидим https://ru.wikipedia.org/wiki/Питоны. Именно в этом и заключается смысл статей с перенаправлением — они перенаправляют на другую статью. Поэтому вершины графа, соответствующие таким статьям, имеют ровно одно ребро.

### Формат файла с графом

В данной работе граф описывается при помощи текстового файла:

<n\_pages=Количество статей> <n\_links=Количество ссылок>

<Название статьи с номером 0>

<Размер статьи в байтах> <Флаг перенаправления> <n₀=Количество ссылок из статьи>

<Номер статьи, на которую ссылается статья с номером 0>

…

<Номер статьи, на которую ссылается статья с номером 0>

<Название статьи с номером 1>

<Размер статьи в байтах> <Флаг перенаправления> <n₁=Количество ссылок из статьи>

<Номер статьи, на которую ссылается статья с номером 1>

…

<Номер статьи, на которую ссылается статья с номером 1>

…

<Название статьи с номером m-1>

<Размер статьи в байтах> <Флаг перенаправления> <nₘ₋₁=Количество ссылок из статьи>

<Номер статьи, на которую ссылается статья с номером m-1>

…

<Номер статьи, на которую ссылается статья с номером m-1>

При этом Σnᵢ=n\_links.

Таким образом, пример графа статей, рассмотренный выше, описывается следующим файлом (указаны произвольные размеры статей):

9 8

Питон

1 1 1

1

Питоны

7 0 0

Snake\_(игра)

9 0 0

Охраняемый\_природный\_район\_Питон

9 0 0

Питон\_(Эна)

12 0 0

Монти\_Пайтон

99 0 0

Python

45 0 0

Жаргон

23 0 0

Питон\_(значения)

53 0 7

1

2

3

4

5

6

7

## Способ хранения графа в памяти

### Компактный способ хранения графа

Теперь перейдём к вопросу о том, каким образом можно представить граф в памяти компьютера без использования списков, словарей и множеств: для этого достаточно двух массивов. Такой способ хранения называется CSR (Compressed Sparse Row) и используется во многих библиотеках, предназначенных для работы с графами (например, METIS). Рассмотрим процесс построения этих массивов на примере графа, изображённого ниже:

Сначала выпишем все рёбра, отсортировав их по номеру вершины, из которой исходит ребро, и пронумеровав строки:

(0) 0 --> 1

(1) 0 --> 2

(2) 0 --> 3

(3) 0 --> 4

(4) 0 --> 5

(5) 1 --> 4

(6) 2 --> 1

(7) 3 --> 2

(8) 5 --> 1

(9) 5 --> 2

Полученный правый столбец — это содержимое первого массива. Этот массив хранит номера вершин, в которые приходит ребро. Теперь для каждой из пяти вершин выпишем количество рёбер, исходящих из этой вершины:

(0) 5

(1) 1

(2) 1

(3) 1

(4) 0

(5) 2

Теперь построим второй массив по следующему правилу:

* offset₀ = 0
* offsetᵢ = offsetᵢ₋₁ + <количество рёбер, исходящих из вершины с номером i-1> для 1 ≤ i ≤ n\_pages, n\_pages — количество вершин в графе.

Таким образом мы построили два массива. Массив offset содержит индексы в массиве edges, с которых начинаются рёбра, исходящие из соответствующей вершины. Массив edges содержит номера вершин, в которые приходит ребро. Т.е. из вершины k исходят offset[k+1]-offset[k] рёбер в вершины с индексами edges[offset[k]:offset[k+1]].

uint32\_t \*edges, \*offset;

edges = new uint32\_t[10];

offset = new uint32\_t[8];

int j = 0;

for (int i: {1, 2, 3, 4, 5, 4, 1, 2, 1, 2}){

edges[j++] = i;

};

j = 0;

for (int i: {0, 5, 6, 7, 8, 8, 10}){

offset[j++] = i;

}

for (int j = 0; j<6 ; ++j){

std::cout << utf16\_to\_utf8(L"Рёбра из вершины ") << j << ": ";

for (int i = offset[j]; i<offset[j + 1]; ++i){

std::cout << edges[i] << " ";

}

std::cout << std::endl;

}

результат работы:

Рёбра из вершины 0: 1 2 3 4 5

Рёбра из вершины 1: 4

Рёбра из вершины 2: 1

Рёбра из вершины 3: 2

Рёбра из вершины 4:

Рёбра из вершины 5: 1 2

## Анализ графа статей

Перейдём непосредственно к практике. Конечная цель работы — получить некоторые статистические данные по графу статей Википедии.

Перед началом выполнения упражнений сделайте форк [репозитория](https://github.com/mipt-cs/wiki-stats-cpp) с заготовкой для работы. Или скачайте его [здесь](http://127.0.0.1:8000/extra/lab6/wiki-stats-cpp.zip).

### Упражнение №1

Реализуйте метод WikiGraph.load\_from\_file для загрузки графа из текстового файла. Для хранения используйте способ, описанный ранее. В качестве входного файла с описанием графа используйте файл wiki\_small.txt из репозитория.

### Упражнение №2

Реализуйте все оставшиеся методы класса WikiGraph.

### Упражнение №3

При помощи реализованных методов определите следующее:

* путь, по которому можно добраться от статьи [Python](https://ru.wikipedia.org/wiki/Python) до статьи [Список\_файловых\_систем](https://ru.wikipedia.org/wiki/Список_файловых_систем);
* количество статей с перенаправлением;
* минимальное количество ссылок из статьи;
* количество статей с минимальным количеством ссылок;
* максимальное количество ссылок из статьи;
* количество статей с максимальным количеством ссылок;
* статья с наибольшим количеством ссылок;
* среднее количество ссылок в статье;
* минимальное количество ссылок на статью (перенаправление не считается внешней ссылкой);
* количество статей с минимальным количеством внешних ссылок;
* максимальное количество ссылок на статью;
* количество статей с максимальным количеством внешних ссылок;
* статья с наибольшим количеством внешних ссылок;
* среднее количество внешних ссылок на статью;
* минимальное количество перенаправлений на статью;
* количество статей с минимальным количеством внешних перенаправлений;
* максимальное количество перенаправлений на статью;
* количество статей с максимальным количеством внешних перенаправлений;
* статья с наибольшим количеством внешних перенаправлений;
* среднее количество внешних перенаправлений на статью.

Примерно так должны выглядеть результаты работы вашей программы:

Загружаю граф из файла: wiki\_small.txt

Граф загружен

Запускаем поиск в ширину

Поиск закончен. Найден путь:

Python

UNIX

Список\_файловых\_систем

Количество статей с перенаправлением: 50 (4.13%)

Минимальное количество ссылок из статьи: 0

Количество статей с минимальным количеством ссылок: 3

Максимальное количество ссылок из статьи: 356

Количество статей с максимальным количеством ссылок: 1

Статья с наибольшим количеством ссылок: Python

Среднее количество ссылок в статье: 34.34 (ср. откл. 32.55)

Минимальное количество ссылок на статью: 0

Количество статей с минимальным количеством внешних ссылок: 152

Максимальное количество ссылок на статью: 1000

Количество статей с максимальным количеством внешних ссылок: 1

Статья с наибольшим количеством внешних ссылок: Python

Среднее количество внешних ссылок на статью: 32.92 (ср. откл. 68.47)

Минимальное количество перенаправлений на статью: 0

Количество статей с минимальным количеством внешних перенаправлений: 1171

Максимальное количество перенаправлений на статью: 7

Количество статей с максимальным количеством внешних перенаправлений: 1

Статья с наибольшим количеством внешних перенаправлений: Python

Среднее количество внешних перенаправлений на статью: 0.04 (ср. откл. 0.28)

### Упражнение №4

Скачайте [отсюда](https://www.dropbox.com/s/5977miarjdqzcgk/wiki.txt.gz) полный граф статей Википедии в папку репозитория. Распакуйте его при помощи команды: gzip -dc wiki.txt.gz > wiki.txt. Запустите свою программу, передав в качестве входного файла wiki.txt.

1. Получите статистические данные из упражнения №3.
2. Найдите путь, по которому можно добраться от статьи [Python](https://ru.wikipedia.org/wiki/Python) до статьи [Боль](https://ru.wikipedia.org/wiki/Боль).