Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Лабораторна робота №1

«Реалізація ідеального хешування»

Виконав студент 2-го курсу

Групи ІПС21

Вербицький Артем Віталійович

Київ – 2024

**Завдання**:

Реалізувати ідеальне хешування (тип даних раціональні числа).

**Теорія**

Ідеальна хеш-функція – хеш-функція, яка перетворює завчасно відому статичну множину ключів в діапазоні цілих чисел [0, n-1] без колізій, тобто один ключ відповідає лише одному унікальному значенню.

**Алгоритм**

1. Отримуємо набір вхідних даних (раціональні числа).
2. Перевіряємо дані на повтори.
3. Створюємо хеш-таблицю, розмір якої дорівнює кількості елементів вхідного набору даних.
4. Для кожного елемента рахуємо його індекс у хеш-таблиці за допомогою функції k = h(r) = ((r.num \* a + r.den \* b) mod p) mod m
5. Якщо виникає колізія, тобто до однієї комірки потрапляє кілька елементів створюємо хеш-підтаблицю у цій комірці, розмір якої дорівнює квадрату кількості елементів, що потрапили в одну й ту саму комірку.
6. Хешуємо кожний елемент підтаблиці попередньо обравши нові змінні a і b. Повторюємо цей пункт для всіх підтаблиць та доти, доки у них не буде нових колізій.

**Складність алгоритму**

Алгоритм створення таблиці працює за О(n) в найгіршому випадку, алгоритм пошуку – за час О(1) в найгіршому випадку.

**Мова реалізації алгоритму** С++

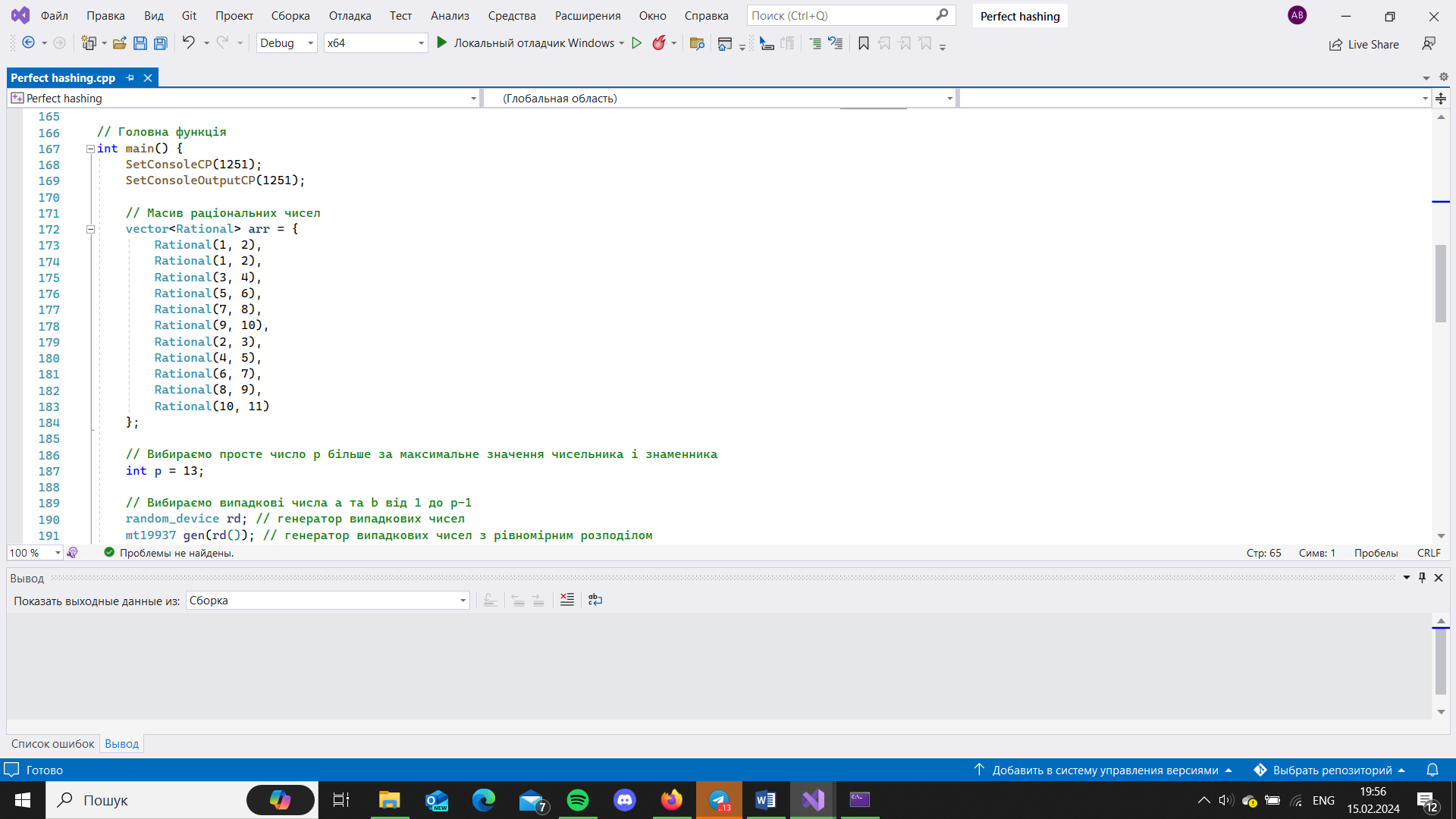
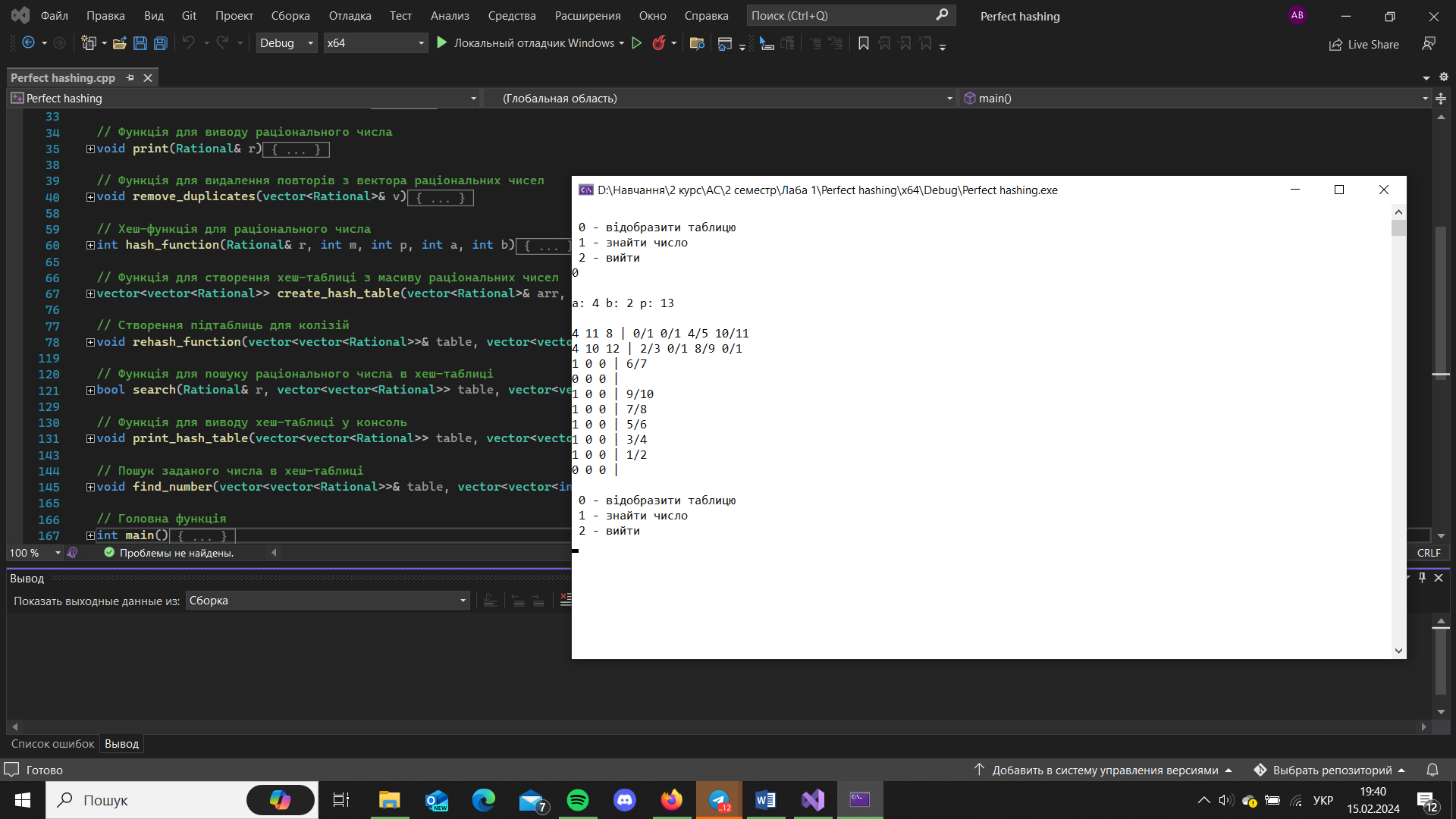
**Модулі програми:**

* struct Rational // Структура для зберігання раціонального числа
* int gcd(int num, int den) // Функція для обчислення найбільшого спільного дільника двох чисел
* void reduce(Rational& r)  // Функція для скорочення раціонального числа
* bool equal(Rational& r1, Rational& r2) // Функція для порівняння двох раціональних чисел
* void print(Rational& r) // Функція для виводу раціонального числа
* void remove\_duplicates(vector<Rational>& v) // Функція для видалення повторів з вектора раціональних чисел
* int hash\_function(Rational& r, int m, int p, int a, int b) // Хеш-функція для раціонального числа
* vector<vector<Rational>> create\_hash\_table(vector<Rational>& arr, vector<vector<int>>& coefficient, int p, int a, int b) // Функція для створення хеш-таблиці з масиву раціональних чисел
* void rehash\_function(vector<vector<Rational>>& table, vector<vector<int>>& coefficient, int p) // Створення підтаблиць для колізій
* bool search(Rational& r, vector<vector<Rational>> table, vector<vector<int>> coefficient, int p, int a, int b) // Функція для пошуку раціонального числа в хеш-таблиці
* void print\_hash\_table(vector<vector<Rational>> table, vector<vector<int>> coefficient) // Функція для виводу хеш-таблиці у консоль
* void find\_number(vector<vector<Rational>>& table, vector<vector<int>> coefficient, int p, int a, int b) // Пошук заданого числа в хеш-таблиці
* int main() // Головна функція

**Інтерфейс користувача**

Вхідні дані вводяться у програмі (в функції int main()) і виводяться в консоль.

**Тестовий приклад**

Маємо вектор раціональних чисел, представлених у вигляді дробу двох дійсних чисел, з 11 елементів, два з яких однакові.

На прикладі числа 2/3 розглянемо роботу хеш-функції.

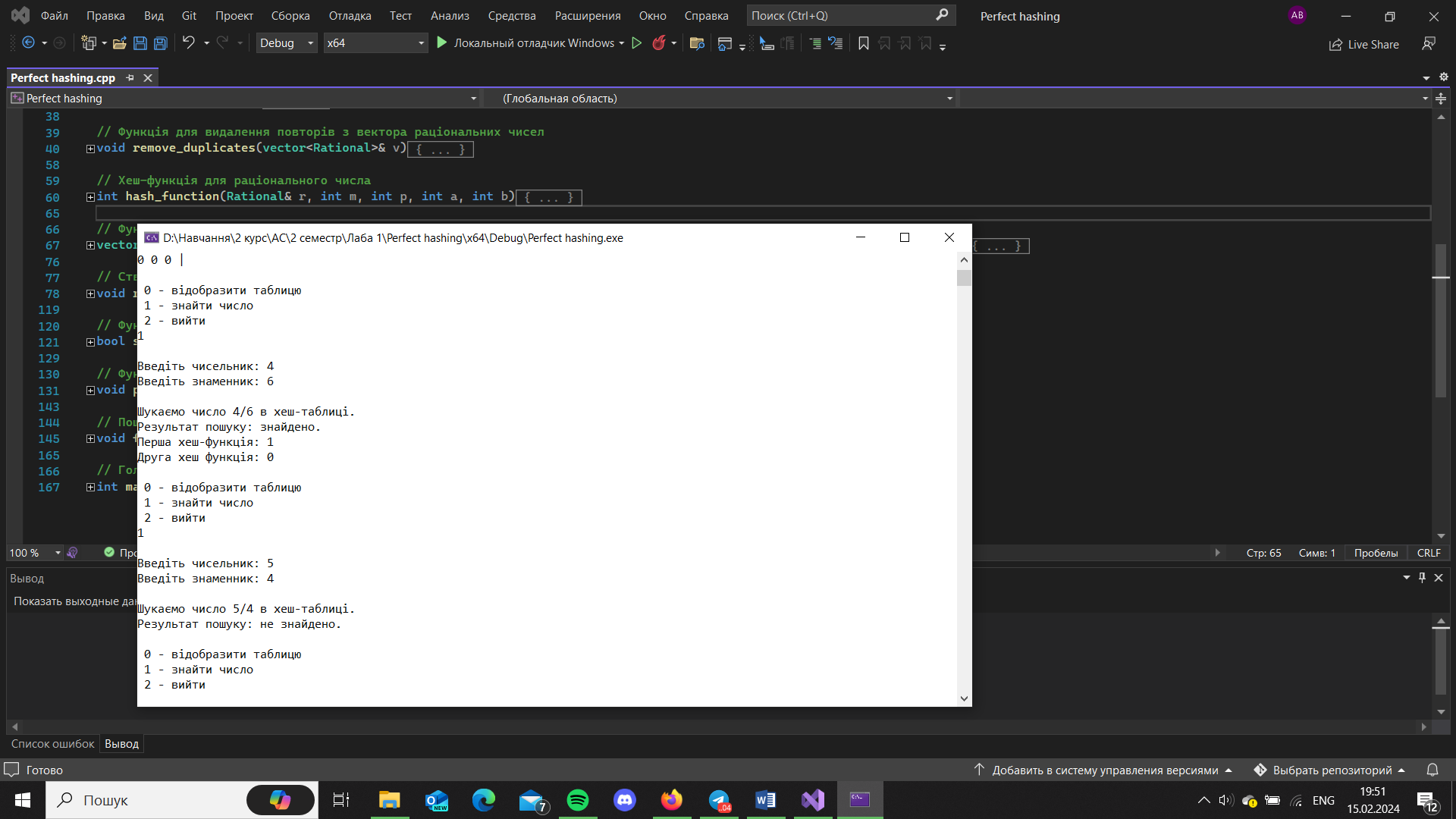
Отримуємо числа a = 4, b = 2, m = 10, p = 13 і за формулою

**h(r) = ((r.numerator \* a + r.denominator \* b) mod p) mod m**маємо

**h(2/3) = (2 \* 4 + 3 \* 2) mod 13 mod 10 = 1**

h(8/9) також дорівнює 2, тому створюємо підтаблицю та визначаємо нову хеш-функцію й маємо a = 10, b = 12, m = 4, p = 13:

**h(2/3) = (2 \* 10 + 3 \* 12) mod 13 mod 4 = 0**



Здійснимо пошук чисел у створеній хеш-таблиці. Для пошуку ми шукаємо результат хеш-функції для числа, яке ввів користувач, а також другої хеш-функції, що визначає підтаблицю, і після цього порівнюємо з числом на відповідній позиції у хеш-таблиці.

Перше число 4/6, що дорівнює 2/3, є у хеш-таблиці. Друге число 5/4 відсутнє у хеш-таблиці. З прикладу вище можна побачити, що число 4/6 має значення 1 для першої хеш-функції 1 та 0 для другої, що відповідає числу 2/3 у хеш-таблиці, власне, тому пошук істинний. Числа 5/4 немає у хеш-таблиці, тому пошук відповідно виявився хибним.

**Висновки**

Ідеальне хешування використовується рідко, проте його алгоритм не є складним та важким для програмування. Це дуже потужний інструмент у правильних руках. Будучи дуже ситуативним, за правильної реалізації, може дати можливість виконувати операції на статичному наборі даних за константу складність, навіть якщо розмір вхідних даних є надзвичайно великим.

**Використані літературні джерела**

* Лекція №1 з предмету «Алгоритми та складність»
* <https://en.wikipedia.org/wiki/Perfect_hash_function>
* Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms. — 3rd. — MIT Press, 2009. (page 311)