СПб ВШЭ, 3-й курс, 4-й модуль 2018/19
Конспект лекций по алгоритмам

Собрано 13 июня 2019 г. в 13:26

Содержание

Лекция #1: Действия над многочленами

8 апреля 2019

1.1. Над $\mathbb{F}_2$

Умножение/деление/gcd можно делать битовым сжатием за $\approx \frac{nm}{w}$, где w – word size.

• Хранение

$A(x) = a_0 + a_1 x + \cdots + a_n x^n \ \to$ `N = n + 1; bitset<N> a`
Обобзначения: $n = \deg A$, $m = \deg B$, $N = \deg A + 1$, $M = \deg B + 1$.

• Умножение

```
1 for i=0..n
2   if a[i]
3     c ^= b << i
```

Время работы: $\mathcal{O}(n \cdot \lceil \frac{m}{w} \rceil)$. Можно n заменить на "*число не нулей в* `a`".

• Деление

```
1 for i=n..m
2   if a[i]
3     a ^= b << (i-m), c[i-m] = 1
```

Результат: в `a` лежит остаток, в `c` частное.
Время работы: $\mathcal{O}((n-m) \cdot \lceil \frac{m}{w} \rceil)$.

• gcd

Запускаем Евклида. Один шаг Евклида – деление.
За $\mathcal{O}((n-m) \cdot \lceil \frac{m}{w} \rceil)$ переходим от n к $n' < m$. Суммарное время работы $\mathcal{O}(\frac{nm}{w} + n + m)$.

1.2. Умножение многочленов

Над произвольным кольцом умеем за $\mathcal{O}(nm)$.
Точнее за "(*число не нулей в* `a`)$\cdot$(*число не нулей в* `b`)".

• Карацуба

Пусть $N = 2^k$. Если нет, дополним нулями.
Делим многочлены на две части: $A = A_0 + x^{N/2} A_1, \ B = B_0 + x^{N/2} B_1$

$$A \cdot B = A_0 B_0 + x^N A_1 B_1 + x^{N/2}(A_0 B_1 + A_1 B_0) = C_0 + x^N C_1 + x^{N/2}((A_0+B_0)(A_1+B_1) - C_0 - C_1)$$

Умножение многочленов длины N – сложение, вычитание и 3 умножения многочленов длины $\frac{N}{2}$.

$$T(n) = 3T(\tfrac{n}{2}) + n = \boxed{\Theta(n^{\log 3})}$$

Умножение работает над произвольным кольцом.

• Оптимальное над $\mathbb{F}_2$

У нас уже есть Карацуба и битовое сжатие. Соединим. Если внутри Карацубы реализовать сложения и вычитания за $\lceil \frac{n}{w} \rceil$, получится $T(n) = 3T(\frac{n}{2}) + \lceil \frac{n}{w} \rceil = \Theta(n^{\log 3})$. То есть, столько же.

Асимптотическая оптимизация: в рекурсии при $n \leqslant w$ умножить за $\mathcal{O}(n)$.
Тогда новое время работы в $w^{\log 3}/w = w^{(\log 3)-1}$ раз меньше.

• **Над $\mathbb{Z}$, над $\mathbb{R}$, над $\mathbb{C}$**

Фурье за $\mathcal{O}(n \log n)$. Смотри главу про Фурье (FFT).

• **Над конечным полем**

Все конечные поля изоморфны $\Rightarrow$ умножить над $\mathbb{F}_p \Leftrightarrow$ умножить над $\mathbb{Z}/p\mathbb{Z}$.
Умножим в $\mathbb{Z}$, затем возьмём по модулю.

Лекция #2: FFT и его друзья

5 сентября

2.1. FFT

2.1.1. Прелюдия

Пусть есть многочлены $A(x) = \sum a_i x^i$ и $B(x) = \sum b_i x^i$.
Посчитаем их значения в точках $x_1, x_2, \ldots, x_n$: $A(x_i) = fa_i, B(x_i) = fb_i$.
Значения $C(x) = A(x)B(x)$ в точках x_i можно получить за линейное время:

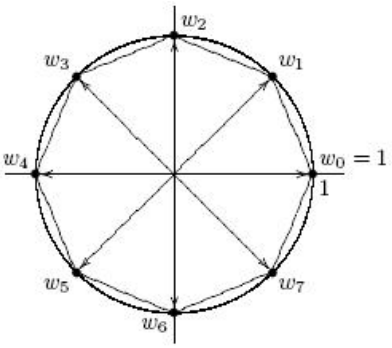
$$fc_i = C(x_i) = A(x_i)B(x_i) = fa_i fb_i$$

Схема быстрого умножения многочленов:

$$a_i, b_i \overset{\mathcal{O}(n\log n)}{\longrightarrow} fa_i, fb_i \overset{\mathcal{O}(n)}{\longrightarrow} fc_i = fa_i fb_i \overset{\mathcal{O}(n\log n)}{\longrightarrow} c_i$$

Осталось подобрать правильные точки x_i.

FFT расшифровывается Fast Fourier Transform и за $\mathcal{O}(n \log n)$ вычисляет значения многочлена в точках $w_j = e^{\frac{2\pi i j}{n}}$ для $n = 2^k$ (то есть, только для степеней двойки).



2.1.2. Собственно идея FFT

$A(x) = \sum a_i x^i = (a_0 + x^2 a_2 + x^4 a_4 + \ldots) + x(a_1 x + a_3 x^3 + a_5 x^5 + \ldots) = B(x^2) + xC(x^2)$ – обозначили все чётные коэффициенты многочлена A многочленом B, а нечётные соответственно C.
Заметим, что $\forall j\ w_j = w_{j \bmod n} \Rightarrow \forall j\ w_j^2 = w_{n/2+j}^2 \Rightarrow B$ и C нужно считать только в $\frac{n}{2}$ точках.

Итого алгоритм:

```
1 def FFT(a):
2     n = len(a)
3     if n == 1: return a[0] # посчитать значение многочлена A(x) ≡ a[0] в x = 1
4     for j=0..n-1: b[j % 2][j / 2] = a[j]
5     b[0], b[1] = FFT(b[0]), FFT(b[1])
6     w = exp(2*pi*i/n)
7     for j=0..n-1: a[j] = b[0][j % (n/2)] + exp(2πi*j/n) * b[1][j % (n/2)]
8     return a
```

Время работы $T(n) = 2T(n/2) + \mathcal{O}(n) = \mathcal{O}(n \log n)$.

2.1.3. Крутая реализация FFT

Чтобы преобразование работало быстро, нужно заранее предподсчитать все $w_j = e^{\frac{2\pi i j}{n}}$.
Заметим, что b и c можно хранить прямо в массиве a. Тогда получается, что на прямом ходу рекурсии мы просто переставляем местами элементы a, а уже на обратном делаем какие-то

полезные действия. Число a_i перейдёт на позицию $a_{rev(i)}$, где $rev(i)$ – перевёрнутая битовая запись i. Кстати, $rev(i)$ мы уже умеем считать динамикой для всех i.
При реализации на C++ можно использовать стандартные комплексные числа `complex<double>`.

```
const int K = 20, N = 1 << K; // N - ограничение на длину результата умножения многочленов
complex<double> root[N];
int rev[N];
void init() {
  for (int j = 0; j < N; j++) {
    root[j] = exp(2πi*j/N); // cos(2πj/N), sin(2πj/N)
    rev[j] = rev[j >> 1] + ((j & 1) << (K - 1));
  }
}
```

Теперь, корни из единицы степени k хранятся в `root[j*N/k]`, $j \in [0, k)$.
Доступ к памяти при этом не последовательный, проблемы с кешом.
Чтобы посчитать все корни, мы $2N$ раз вычисляли тригонометрические функции.

- **Улучшенная версия вычисления корней**

```
  for (int k = 1; k < N; k *= 2) {
    num tmp = exp(π/k);
    root[k] = {1, 0}; // в root[k..2k) хранятся первые k корней степени 2k
    for (int i = 1; i < k; i++)
      root[k + i] = (i & 1) ? root[(k + i) >> 1] * tmp : root[(k + i) >> 1];
  }
```

Теперь код собственно преобразования Фурье может выглядеть так:

```
FFT(a) { // a → f
  vector<complex> f(N);
  for (int i = 0; i < N; i++) // прямой ход рекурсии превратился в один for =)
    f[rev[i]] = a[i];
  for (int k = 1; k < N; k *= 2) // пусть уже посчитаны FFT от кусков длины k
    for (int i = 0; i < N; i += 2 * k) // [i..i+k) [i+k..i+2k) → [i..i+2k)
      for (int j = 0; j < k; j++) { // оптимально написанный главный цикл FFT
        num tmp = root[k + j] * f[i + j + k]; // root[] из "улучшенной версии"
        f[i + j + k] = f[i + j] - tmp; // w_{j+k} = -w_j при n = 2k
        f[i + j] = f[i + j] + tmp;
      }
  return f;
}
```

2.1.4. Обратное преобразование

Обозначим $w = e^{2\pi i/n}$. Нам нужно из
$f_0 = a_0 + a_1 + a_2 + a_3 + \dots$
$f_1 = a_0 + a_1 w + a_2 w^2 + a_3 w^3 + \dots$
$f_2 = a_0 + a_1 w^2 + a_2 w^4 + a_3 w^3 + \dots$
научиться восстанавливать коэффициенты $a_0, a_1, a_2, \dots$

Заметим, что $\forall j \neq 0 \sum\limits_{k=0}^{n-1} w^{jk} = 0$ (сумма геометрической прогрессии).
И напротив при $j = 0$ получаем $\sum\limits_{k=0}^{n-1} w^{jk} = \sum 1 = n$.

Поэтому $f_0 + f_1 + f_2 + \dots = a_0 n + a_1 \sum_k w^k + a_2 \sum_k w^{2k} + \dots = \boxed{a_0 n}$
Аналогично $f_0 + f_1 w^{-1} + f_2 w^{-2} + \dots = \sum_k a_0 w^{-k} + a_1 n + a_2 \sum_k w^k + \dots = \boxed{a_1 n}$
И в общем случае $\sum_k f_k w^{-jk} = \boxed{a_j n}$
Рассмотрим $F(x) = f_0 + x f_1, x^2 f_2 + \dots \Rightarrow F(w^{-j}) = a_j n$, похоже на `FFT(f)`.
Осталось заметить, что множества чисел $\{w_j \mid j = 0..n-1\}$ и $\{w_{-j} \mid j = 0..n-1\}$ совпадают $\Rightarrow$

```
1 FFT_inverse(f) { // f → a
2   a = FFT(f)
3   reverse(a + 1, a + N) // w^j ↔ w^{-j}
4   for (int i = 0; i < N; i++) a[i] /= N;
5   return a;
6 }
```

2.1.5. Два в одном

Часто коэффициенты многочленов – вещественные числа.
Если у нас есть многочлены $A(x), B(x) \in \mathbb{R}[x]$, возьмём числа $c_j = a_j + ib_j$, коэффициенты $C(x) = A(x) + iB(x)$, посчитаем $fc = FFT(c)$. Тогда по f за $\mathcal{O}(n)$ можно восстановить fa и fb.

Для этого вспомним про сопряжения комплексных чисел:
$\overline{x + iy} = x - iy$, $\overline{u \cdot v} = \overline{u} \cdot \overline{v}$, $w^{n-j} = w^{-j} = \overline{w^j} \Rightarrow \overline{fc_{n-j}} = \overline{C(w^{n-j})} = \overline{C}(w^j) = A(w^j) - iB(w^j) \Rightarrow$
$fc_j + \overline{fc_{n-j}} = 2A(w^j) = 2 \cdot fa_j$. Аналогично $fc_j - \overline{fc_{n-j}} = 2B(w^j) = 2i \cdot fb_j$.

Итого для умножения двух многочленов можно использовать не 3 преобразования Фурье, а 2.

2.1.6. Умножение чисел, оценка погрешности

Число длины n в системе счисления 10 $\rightarrow$ система счисления 10^k $\rightarrow$ многочлен длины n/k.
Умножения многочленов такой длины будет работать за $\frac{n}{k} \log \frac{n}{k}$.
Отсюда возникает вопрос, какое максимальное k можно использовать?
Коэффициенты многочлена-произведения будут целыми числами до $(10^k)^2 \cdot \frac{n}{k}$.
Чтобы в типе `double` целое число хранилось с погрешностью меньше 0.5 (тогда мы его сможем правильно округлить к целому), оно должно быть не более 10^{15}.
Получаем при $n \leqslant 10^6$, что $(10^k)^2 \cdot 10^6 / k \leqslant 10^{15} \Rightarrow k \leqslant 4$.
Аналогично для типа `long double` имеем $(10^k)^2 \cdot 10^6 / k \leqslant 10^{18} \Rightarrow k \leqslant 6$.
Это оценка сверху, предполагающая, что само FFT погрешность не накапливает... на самом деле эта оценка очень близка к точной.

2.2. Разделяй и властвуй

2.2.1. Перевод между системами счисления

Задача: перевести число X длины $n = 2^k$ из a-ичной системы счисления в b-ичную.

Разобьём число X на $\frac{n}{2}$ старших цифр и $\frac{n}{2}$ младших цифр: $X = X_0 \cdot a^{n/2} + X_1 \Rightarrow$

$$F(X) = F(X_0) F(a^{n/2}) + F(X_1)$$

Умножение за $\mathcal{O}(n \log n)$ и сложение за $\mathcal{O}(n)$ выполняются в системе счисления b.
Предподсчёт $F(a^1), F(a^2), F(a^4), F(a^8), \dots, F(a^n)$ займёт $\sum_k \mathcal{O}(2^k k) = \mathcal{O}(n \log n)$ времени.
Итого $T(n) = 2T(n/2) + \mathcal{O}(n \log n) = \mathcal{O}(n \log^2 n)$.

2.2.2. Деление многочленов с остатком

Задача: даны $A(x), B(x) \in \mathbb{R}[x]$, найти $Q(x), R(x)$: $\deg R < \deg B \ \wedge\ A(x) = B(x)Q(x) + R(x)$.

Зная Q мы легко найдём R, как $A(x) - B(x)Q(x)$ за $\mathcal{O}(n \log n)$. Сосредоточимся на поиске Q.
Пусть $\deg A = \deg B = n$, тогда $Q(x) = \frac{a_n}{b_n}$. То есть, $Q(x)$ можно найти за $\mathcal{O}(1)$.
Из этого мы делаем вывод, что Q зависит не обязательно от всех коэффициентов A и B.

Lm 2.2.1. $\deg A = m, \deg B = n \Rightarrow \deg Q = m - n$, и Q зависит только от $m{-}n{+}1$ коэффициентов A и $m{-}n{+}1$ коэффициентов B.

Доказательство. У A и $B \cdot Q$ должны совпадать $m{-}n{+}1$ старший коэффициент ($\deg R < n$). В этом сравнении участвуют только $m{-}n{+}1$ старших коэффициентов A. При домножение B на $x^{\deg Q}$, сравнятся как раз $m{-}n{+}1$ старших коэффициентов A и B. При домножении B на меньшие степени x, в сравнении будут участвовать лишь какие-то первые из этих $m{-}n{+}1$ коэффициентов. ■

Теперь будем решать задачу: даны n старших коэффициентов A и B, найти такой C из n коэффициентов, что у A и BC совпадает n старших коэффициентов. Давайте считать, что младшие коэффициенты лежат в первых ячейках массива.

```
1 Div(int n, int *A, int *B)
2   C = Div(n/2, A + n/2, B + n/2) // нашли старших n/2 коэффициентов ответа
3   A' = Subtract(n, A, n + n/2 - 1, Multiply(C, B))
4   D = Div(n/2, A', B + n/2) // сейчас A' состоит из n/2 не нулей и n/2 нулей
5   return concatenate(D, C) // склеили массивы коэффициентов
```

Здесь `Subtract` – хитрая функция. Она знает длины многочленов, которые ей передали, и сдвигает вычитаемый многочлен так, чтобы старшие коэффициенты совместились.

2.2.3. Вычисление значений в произвольных точках

Задача. Дан многочлен $A(x)$, $\deg A = n$ и точки $x_1, x_2, \dots x_n$. Найти $A(x_1), A(x_2), \dots A(x_n)$.

Вспомним теорему Безу: $A(w) = A(x) \bmod (x - w)$.
Обобщение: $B(x) = A(x) \bmod (x - x_1)(x - x_2)\dots(x - x_n) \ \Rightarrow\ \forall j\ B(x_j) = A(x_j)$

```
1 def Evaluate(n, A, x[]): # n = 2^k
2   if n == 1: return list(A[0])
3   return Evaluate(n/2, A mod (x-x_1)·...·(x-x_{n/2}), [x_1, ... , x_{n/2}]) +
4          Evaluate(n/2, A mod (x-x_{n/2+1})·...·(x-x_n), [x_{n/2+1}, ... , x_n])
```

Итого $T(n) = 2T(n/2) + \mathcal{O}(\texttt{div}(n))$. Если деление реализовано за $\mathcal{O}(n \log n)$, получим $\mathcal{O}(n \log^2 n)$.

2.2.4. Интерполяция

Задача. Даны пары $(x_1, y_1), \dots, (x_n, y_n)$. Найти многочлен A: $\deg A = n{-}1,\ \forall i\ A(x_i) = y_i$.

Сделаем интерполяцию по Ньютону методом разделяй и властвуй.
Сперва найдём интерполяционный многочлен B для $(x_1, y_1), (x_2, y_2), \dots, (x_{n/2}, y_{n/2})$.

$$A = B + C \cdot D, \text{ где } D = \prod_{j=1..\frac{n}{2}} (x - x_j), \text{ а } C \text{ нужно найти}$$

Подгоним правильные значения в точках $x_{n/2+1}, \dots, x_n$, вычислим b_j, d_j – значения B и D в точках $x_{n/2+1}, \dots, x_n \Rightarrow C$ – интерполяционный многочлен точек $(x_j, -\frac{b_j}{d_j})$ при $j = \frac{n}{2}+1 \dots n$.

Итого $T(n) = 2T(n/2) + 2\mathcal{O}(\texttt{evaluate}(n/2))$. При $\texttt{evaluate}(n) = \mathcal{O}(n \log^2 n)$ имеем $\mathcal{O}(n \log^3 n)$.

2.2.5. Извлечение корня

Дан многочлен $A(x)$: $\deg A \equiv 0 \bmod 2$. Задача – найти $R(x)$: $\deg(A - R^2)$ минимальна.
Пусть мы уже нашли старшие k коэффициентов R, обозначим их R_k. Найдём $2k$ коэфф-тов: $R_{2k} = R_k x^k + X, R_{2k}^2 = R_k^2 x^{2k} + 2R_k X \cdot x^k + X^2$. Правильно подобрав X, мы можем "обнулить" k коэффициентов $A - R_{2k}^2$, для этого возьмём $X = (A - R_k^2 x^{2k})/(2R_k)$. В этом частном нам интересны только k старших коэффициентов, поэтому переход от R_k к R_{2k} происходит за $\mathcal{O}(\texttt{mul}(k) + \texttt{div}(k))$. Итого суммарное время на извлечение корня – $\mathcal{O}(\texttt{div}(n))$.

2.3. Литература

[sankowski]. Слайды по FFT и всем идеям разделяй и властвуй.
[e-maxx]. Про FFT и оптимизации к нему.
[codeforces]. Задачи на тему FFT.
[vk]. Краткий конспект похожих идей от Александра Кулькова.

Лекция #3: Деление многочленов

12 сентября

3.1. Быстрое деление многочленов

Цель – научиться делить многочлены за $\mathcal{O}(n \log n)$.
Очень хочется считать частное многочленов $A(x)/B(x)$, как $A(x)B^{-1}(x)$. К сожалению, у многочленов нет обратных. Зато обратные есть у рядов, научимся сперва искать их.

• Обращение ряда

Задача. Дан ряд $A \in [[\mathbb{R}]]$, $a_0 \neq 0$. Найти ряд B: $A(x)B(x) = 1$.
Первые n коэффициентов B можно найти за $\mathcal{O}(n^2)$:
$b_0 = 1/a_0$
$b_1 = -(a_1 b_0)/a_0$
$b_2 = -(a_2 b_0 + a_1 b_1)/a_0$
...
А можно за $\mathcal{O}(n \log n)$.
Обозначим $B_k(x) = b_0 + b_1 x + \cdots + b_{k-1}x^{k-1}$. Заметим, что $\forall k$ $A(x)B_k(x) = 1 + x^k C_k(x)$.
$B_1 = b_0 = 1/a_0$. Научимся делать переход $B_k \to B_{2k}$ за $\mathcal{O}(k \log k)$.
$B_{2k} = B_k + x^k Z \Rightarrow A \cdot B_{2k} = 1 + x^k C_k + x^k A \cdot Z = 1 + x^k(C_k - A \cdot Z)$.
Выберем $Z = B_k \cdot C_k \Rightarrow C_k - A \cdot Z = C_k - C_k(A \cdot B_k) = C_k - C_k(1 + x^k C_k) = -x^k C_k^2$.
Итого $B_{2k} = B_k + B_k(x^k C_k) = B_k + B_k(1 - A \cdot B_k) \Rightarrow \boxed{B_{2k} = B_k(2 - A \cdot B_k)}$

Два умножения = $\mathcal{O}(k \log k)$. Общее время работы $n \log n + \frac{n}{2} \log \frac{n}{2} + \frac{n}{4} \log \frac{n}{4} + \cdots = \mathcal{O}(n \log n)$.
Конечно, мы обрежем B_{2k}, оставив лишь $2k$ первых членов.

• Деление многочленов

A^R – reverse многочлена. $a_0 + a_1 x + \cdots + a_n x^n \to a_n + a_{n-1}x + \cdots + a_0 x^n$.
Умножение: $A^R B^R = (AB)^R$ (доказательство: в c_{ij} += $a_i b_j$ поменяли индексы на $n-i$ и $m-j$).
Новое определение деления: по A, B хотим C: $A^R \equiv (BC)^R \bmod x^n$.
Здесь n – число коэффициентов у A, у B ровно столько же.

Обращение ряда нам даёт умение по многочлену Z: $z_0 \neq 0$ строить Z^{-1}: $Z \cdot Z^{-1} \equiv 1 \bmod x^n$.

$$C^R = (B^R)^{-1} A^R$$

Время работы: обращение ряда + умножение = $\mathcal{O}(n \log n)$.
Над кольцом делить странно, а вот над произвольным полем Фурье может не работать, тогда деление работает за $\mathcal{O}(\texttt{mul}(n) + \texttt{mul}(\frac{n}{2}) + \dots) = \mathcal{O}(\texttt{mul}(n))$ для $\texttt{mul}(n) = \Omega(n)$.

3.2. (*) Быстрое деление чисел

Для нахождение частного чисел, достаточно научиться с большой точностью считать обратно.
Рассмотрим метод Ньютона поиска корня функции $f(x)$:
$x_0 =$ `достаточно точное приближение корня`
$x_{i+1} = x_i - f(x_i)/f'(x_i)$
Решим с помощью него уравнение $f(x) = x^{-1} - a = 0$.
$x_0 =$ `обратное к старшей цифре` a
$x_{i+1} = x_i - (\frac{1}{x_i} - a)/(-\frac{1}{x_i^2}) = x_i + (x_i - a \cdot x_i^2) = x_i(2 - ax_i)$.
Любопытно, что очень похожую формулу мы видели при обращении формального ряда...
Утверждение: каждый шаг метода Ньютона удваивает число точных знаков x.
Итого, имея x_i с k точными знаками, мы научились за $\mathcal{O}(k \log k)$ получать x_{i+1} с $2k$ точными знаками. Суммарное время получения n точных знаков $\mathcal{O}(n \log n)$.

3.3. (*) Быстрое извлечение корня для чисел

Продолжаем пользоваться методом Ньютона.

$$x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$$

Если у x_i k_i верных знаков, то $k_{i+1} = k_i + \Theta(1)$, а
$x_i \to x_{i+1}$ вычисляется одним делением многочленов длины k_i за $\mathcal{O}(k_i \log k_i)$.

3.4. (*) Обоснование метода Ньютона

Цель: доказать, что каждый шаг удваивает число точных знаков x.

Сдалем замену переменных, чтобы было верно $f(0) = 0 \Rightarrow$ корень, который мы ищем, – 0.
Сейчас находимся в точке x_i. По Тейлору $f(0) = f(x_i) - x_i f'(x_i) + x_i^2 f''(\alpha)$ $(\alpha \in [0..x_i])$.
Получаем $\frac{f(x_i)}{f'(x_i)} = x_i + x_i^2 \frac{f''(\alpha)}{f'(x_i)}$. Передаём Ньютону $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - x_i - x_i^2 \frac{f''(\alpha)}{f'(x_i)}$.
Величина $\frac{f''(\alpha)}{f'(x_i)}$ ограничена сверху константой C.
Получаем, что если $x_i \leqslant 2^{-n}$, то $x_{i+1} \leqslant 2^{-2n+\log C}$.
То есть, число верных знаков почти удваивается.

3.5. Линейные рекуррентные соотношения

Задача. Дана последовательность $f_0, f_1, \ldots, f_{k-1}, \forall n \geqslant k\ f_n = f_{n-1}a_1 + \cdots + f_{n-k}a_k$, найти f_n.

Вычисления "в лоб" можно произвести за $\mathcal{O}(nk)$.

3.5.1. Через матрицу в степени

$$A = \begin{bmatrix} a_1 & a_2 & \ldots & a_k \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix},\ \forall i\ A \cdot \begin{bmatrix} f_{i-1} \\ f_{i-2} \\ \ldots \\ f_{i-k} \end{bmatrix} = \begin{bmatrix} f_i \\ f_{i-1} \\ \ldots \\ f_{i-k+1} \end{bmatrix} \Rightarrow\ A^n \cdot \begin{bmatrix} f_{k-1} \\ f_{k-2} \\ \ldots \\ f_0 \end{bmatrix} = \begin{bmatrix} f_{n+k-1} \\ f_{n+k-2} \\ \ldots \\ f_n \end{bmatrix}$$

Умножать матрицы умножаем за $\mathcal{O}(k^3) \Rightarrow$ общее время работы $\mathcal{O}(k^3 \log n)$.

3.5.2. Через умножение многочленов

На самом деле можно возводить в степень не матрицу, а многочлен по модулю...
Умножение матриц за $\mathcal{O}(k^3)$ заменяется на умножение многочленов по модулю за $\mathcal{O}(k \log k)$.

Будем выражать f_n через $f_i \colon i < n$. В каждый момент времени $f_n = \sum_j f_j b_j$.

Изначально $f_n = f_n \cdot 1$. Пока $\exists j \geqslant k \colon b_j \neq 0$, меняем $f_j b_j$ на $\left(\sum\limits_{i=1}^{k} f_{j-i}a_i\right) \cdot b_j$. (*)

Посмотрим, как меняются коэффициенты b_j. Пусть $B(x) = \sum b_j x^j$, $A(x) = x^k - \sum\limits_{i=1}^{k} x^{k-i}a_i$.

Тогда (*) – переход от $B(x)$ к $B(x) - A(x)x^{j-k}b_j$.
Изначально $B(x) = x^n \Rightarrow$ наш алгоритм – вычисление $x^n \bmod A(x)$.
Возведение многочлена x в степень n по модулю $A(x)$ – $\mathcal{O}(\log n)$ умножений и взятий по модулю.

Итого: $\mathcal{O}(k \log k \log n)$.

Лекция #4: Применение умножения многочленов

8 апреля 2019

4.1. Факторизация целых чисел

• **Вычисление** $n! \bmod m$

Возьмём $k = \lfloor\sqrt{n}\rfloor$, рассмотрим $P(x) = x(x+k)(x+2k)...(x+k(k-1))$.
$P(1)P(2)P(3)...P(k) = (k^2)!$, чтобы дополнить до $n!$, сделаем руками $\mathcal{O}(k)$ умножений.
Посчитать $P(x)$ мы можем методом разделяй и властвуй за $O(k \log^2 k)$.

• **FFT над** $\mathbb{Z}/m\mathbb{Z}$

Умножим в $\mathbb{Z}$ (то же, что $\mathbb{R}$, что $\mathbb{C}$), в конце возьмём по модулю m.
Если не хватает точности обычного вещественного типа ($m = 10^9 \Rightarrow$ `long double` уже слишком мал), то представим многочлен с коэффициентами до m, как сумму двух многочленов с коэффициентами до $k = \lceil m^{1/2} \rceil$: $P(x) = P_1(x) + k \cdot P_2(x), Q(x) = Q_1(x) + k \cdot Q_2(x)$.
Сделаем 3 умножения, как в Карацубе:

$$P(x)Q(x) = P_1Q_1 + k^2P_2Q_2 + k((P_1{+}Q_1) \cdot (P_2{+}Q_2) - P_1Q_1 - P_2Q_2)$$

• **Факторизация**

Найдём $\min d$: $gcd(d!, n) \neq 1$. Тогда d – минимальный делитель n. Можно искать бинпоиском, можно "двоичными подъёмами", тогда время $= \mathcal{O}(calc(n^{1/2}!)) = \mathcal{O}(n^{1/4} \log^2 n)$.

4.2. CRC-32

Один из вариантов хеширования последовательности бит $a_0, a_1, \ldots, a_{n-1}$ – взять остаток от деления многочлена $A(x) \cdot x^k$ на $G(x)$, где $G(x)$ – специальный многочлен, а $k = \deg G + 1$.

В *CRC-32-IEEE-802.3* (в v.42, mpeg-2, png, cksum) $k = 32$, $G(x) =$ 0xEDB88320 (32 бита).
Если размер машинного слова $\geqslant k$, CRC вычисляется за $\mathcal{O}(n)$, в общем случаем за $\mathcal{O}(n \cdot \lceil \frac{k}{w} \rceil)$.

```
1 for i = n-1..k:
2   if a[i] != 0:
3     a[i..i-k+1] ^= G
```

Упражнение 4.2.1. `CRC(A ^ B)` = `CRC(A) ^ CRC(B)`, `CRC(concat(A, 1))` = (`CRC(A)` $\cdot 2 + 1$) $\bmod G$

4.3. Кодирование бит с одной ошибкой

Контекст. По каналу хотим передать n бит.
В канале может произойти не более 1 ошибки вида "замена бита".

Детектирование ошибки. Передадим $a_1, a_2, \ldots, a_n$ и $b =$ `XOR`$(a_1, a_2, \ldots, a_n)$.
Если b' равно `XOR`$(a'_1, a'_2, \ldots, a'_n)$, ошибки при передачи не было.

Исправление ошибки.
Удвоение бит не работает – и из 0, и из 1 в результате одной ошибки может получиться 01.
Работает утроение бит ($3n$) или "удвоение бит и дополнительно передать `XOR`" ($2n{+}1$).
Раскодирование для $2n{+}1$: $00 \to 0,\ 11 \to 1,\ 01/10 \Rightarrow$ подгоняем, чтобы сошёлся `XOR`.

Исправление ошибки за $\lceil \log_2 n \rceil + 1$ дополнительных бит.
Передадим два раза $\texttt{XOR}(a_1, a_2, \ldots, a_n)$. Теперь мы знаем, есть ли ошибка среди $a_1, a_2, \ldots, a_n$.
Если ошибка есть, её нужно исправить. $\forall b \in [0, \lceil \log_2 n \rceil)$ передадим $\texttt{XOR}_{i:\, bit(i,b)=0}(a_i)$.
В итоге мы знаем все $\lceil \log_2 n \rceil$ бит позиции ошибки.

4.4. Коды Рида-Соломона

Задача. Кодируем n элементов конечного поля $\mathbb{F}_q$.
Канал допускает k ошибок. Хотим научиться исправлять ошибки после передачи.

Замечание 4.4.1. Конченые поля имеют размер p^k ($p \in Prime, k \in \mathbb{N}$). Поле размера p – $\mathbb{Z}/p\mathbb{Z}$.
Поле размера p^k – остатки по модулю неприводимого многочлена над $\mathbb{F}_p$ степени k.
Для кодирования битовых строк удобно использовать $q = 2^k$ при $k \in \{32, 64, 4096\}$.

Lm 4.4.2. Если обозначить расстояние Хэмминга как $D(s,t)$, а $f(s)$ – код строки s, канал передачи допускает k ошибок, то исправить ошибки можно iff $\forall s \neq t \;\; D(f(s), f(t)) \geqslant 2k{+}1$.

Доказательство. С учётом ошибок строка s может перейти в любую точку шара радиуса k с центром в $f(s)$. Если такие шары пересекаются, $\forall$ точку пересечения не раскодировать. ■

Код Рида-Соломона. Данные $a_0, \ldots a_{n-1}$ задают многочлен $A(x) = \sum a_i x^i$. Передадим значения многочлена в произвольных $n{+}2k{+}1$ различных точках (здесь мы требуем $p \geqslant n$).

Теорема 4.4.3. Корректность кодов Рида-Соломона.

Доказательство. $A(x) \neq B(x) \Rightarrow (A-B)(x)$ имеет не более n корней $\Rightarrow A(x)$ и $B(x)$ имеют не более n общих значений $\Rightarrow$ хотя бы $2k{+}1$ различных $\Rightarrow D(f(A), f(B)) \geqslant 2k{+}1$. ■

Выбор точек и q: хочется применить FFT. На практике можно отправлять данные такими порциями, что $n{+}2k{+}1 = 2^b$. Нужно q вида $a \cdot 2^b + 1$ и x: $ord(x) = 2^b$, точка $w_i = x^i$, $i \in [0, 2^b)$.

• Декодирование

Мы доказали возможность однозначного декодирования, осталось предъявить алгоритм.
Имели $A(x)$, $\deg A = n{-}1$, передали $A(w_0), A(w_1), \ldots A(w_{n+2k})$. Получили на выходе данные с ошибками $A'(w_0), A'(w_1), \ldots A'(w_{n+2k})$. Посчитали интерполяционный многочлен $A'(x)$.

Утверждение 4.4.4. Если у $A'(x)$ старшие $2k{+}1$ коэффициентов нули, ошибок не было.

Итого, если ошибок нет, декодирование при желании можно сделать за $\mathcal{O}(n \log n)$ через FFT.

Обозначим позиции ошибок $e_1, e_2, \ldots, e_k$.
Может быть, ошибок меньше. Главное, что в позициях кроме e_i ошибок точно нет.
Многочлен ошибок $E(x) = (x - e_1)(x - e_2) \ldots (x - e_k)$, $B(x) = A(x)E(x)$, $B'(x) = A'(x)E(x)$.
$\forall i \notin \{e_j\}\ A(w_i) = A'(w_i) \Rightarrow \forall i\ B(w_i) = B'(w_i)$. Запишем это, как СЛАУ $\forall i\ B(w_i) = A'(w_i)E(w_i)$, где неизвестные – коэффициенты многочленов B ($n{+}k{+}1$ штук) и E (k штук).

Теорема 4.4.5. Для записанной СЛАУ $\exists!$ решение.

Следствие 4.4.6. Декодирование: решим СЛАУ, найдём $A(x) = \frac{B(x)}{E(x)}$.

Асимптотика декодирования: Гаусс $\mathcal{O}((n+k)^3)$. Бэрликэмп-Мэсси $\mathcal{O}((n+k)^2)$.
Ссылка на более крутые алгоритмы декодирования в разд. 4.6.

4.5. Счастливые билеты

Задача: билет – последовательность из $2n$ цифр из множества $d_1, d_2, \ldots, d_k$, билет называется счастливым, если сумма первых n цифр совпадает с суммой последних n. Найти число счастливых билетов из $2n$ цифр.

Рассмотрим $P(x) = (x^{d_1} + x^{d_2} + \cdots + x^{d_k})^n$. Ответ – сумма квадратов коэффициентов P.
Время возведения в степень – $\mathcal{O}(\log n)$ умножений.
Точнее $\mathcal{O}(mul(N) + mul(\frac{N}{2}) + mul(\frac{N}{4}) + \ldots) = \mathcal{O}(N \log N)$, где $\deg P = N = n \cdot \max d_i$.

4.6. Литература

[Shuhong Gao'2002]. Декодирование Рида-Соломона через расширенного Евклида.

Лекция #5: Автоматы

10 апреля 2019

5.1. Определения, детерминизация

Def 5.1.1. *Детерминированный автомат* – $\langle V, s, T, \Sigma, E\rangle, s \in V, T \subseteq V, D \subseteq V \times \Sigma,\ E\colon D \to V$

Def 5.1.2. *Детерминированный автомат называется полным, если* $D = V \times \Sigma$.

Замечание 5.1.3. Чтобы сделать автомат полным, добавим фиктивную вершину и все $\nexists$ рёбра направим туда.

Def 5.1.4. *Недетерминированный автомат* – $\langle V, s, T, \Sigma, E\rangle,\ s \in V,\ T \subseteq V,\ E \subseteq (V \times \Sigma) \times V$

Def 5.1.5. *Автомат принимает строку* w, *если* $\exists s = v_0, v_1, \dots v_{|w|}\colon \forall i\ (v_i, s_i, v_{i+1}) \in E$.

• Детерминизация

Принимая строку w, недетерминированный автомат в каждый момент времени t находится в одной из вершин множества A_t. "Множества вершин" и будут состояниями детерминированного автомата.

Можно взять $V' = 2^V$, можно оптимальнее – dfs-ом выбрать достижимые множества вершин.

Время детерминизации $\mathcal{O}(|V'| \cdot |E|)$.

5.2. Эквивалентность

• Простейший алгоритм

Проверяем эквивалентность детерминированных автоматов $\langle V_1, s_1, T_1, E_1\rangle$ и $\langle V_2, s_2, T_2, E_2\rangle$.
Найдём все пары состояний $v_1 \in V_1, v_2 \in V_2\colon v_1 \not\equiv v_2$. Все пары будем помещать в очередь.
База: $(v_1 \in T_1) \neq (v_2 \in T_2) \Rightarrow$ помечаем и помещаем $\langle v_1, v_2\rangle$ в очередь.
Переход: $v_1 \not\equiv v_2 \Rightarrow \forall c, x_1, x_2\colon E(x_1, c) = v_1, E(x_2, c) = v_2 \quad x_1 \not\equiv x_2$.
Реализация: `(v1,v2)= q.pop(); for c: for x1 in from(v1,c): for x2 in from(v2,c): toQueue(x1, x2)`
Каждую пару (v_1, v_2) переберём не более одного раза $\Rightarrow$ каждую пару рёбер $\Rightarrow \mathcal{O}(E^2)$.
Для корректности алгоритма автоматы должны быть полными.

• Через минимизацию

Чтобы проверить эквивалентность $A_1 = \langle V_1, E_1, T_1, s_1\rangle$, $A_2 = \langle V_2, E_2, T_2, s_2\rangle$, запустим минимизацию для $\langle V_1 \cup V_2, E_1 \cup E_2, T_1 \cup T_2\rangle$, и посмотрим попали ли s_1 и s_2 в один класс эквивалентности.

5.3. Минимизация

Задача: построить автомат, минимальный по числу вершин, эквивалентный данному.
Перед тем, как рассматривать решения, поймём, как устроен минимальный автомат.

Def 5.3.1. $R_A(w)$ – *правый контекст строки* w. $R_A(w) = \{x \mid A \text{ принимает } wx\}$.

Теорема 5.3.2. Минимальный автомат, эквивалентный A, есть $A_{min} = \langle V, E, s, T\rangle$, где
$V = \{R_A(w) \text{ по всем } w\},\ E = \{R_A(w) \xrightarrow{c} R_A(wc)\},\ s = R_A(\varepsilon),\ T = \{\varnothing\}$.

Для недетерминированных есть алгоритм Бржозовского [wiki][pdf]:

$$A_{min} = d(r(d(r(A)))) = drdrA$$

Где d – детерминизация автомата, r – разворот всех рёбер автомата и `swap(S, T)`.

Для детерминированных обычно пользуются алгоритмом Хопкрофта.

5.4. Хопкрофт за $\mathcal{O}(VE)$

```
# дополняем автомат до полного (next[v, char] - или конец ребра, или -1)
fictive = newVertex() # next[fictive, *] = -1, isTerminal[fictive] = 0
for v in [0, vertexN):
  for char:
    if next[v, char] == -1:
      next[v, char] = fictive

# строим обратные рёбра, инициализируем классы
for v in [0, vertexN):
  for char:
    prev[next[v, char], char].add(v)
  type[v] = isTerminal[v] # тип/класс вершины
  A[type[v]].add(v)       # A[type] - множество вершин типа type
typeN = 2 # изначально есть только терминалы и нетерминалы

# основной цикл с очередью
queue q; q.push(0); # любой из классов 0, 1
while !q.isEmpty():
  t = q.pop()
  for char:
    Split(t, char) # самая сложная процедура
```

Функция `Split(t, c)` должна разделить во всех существующих классах разделить вершины по предикату "*ведёт ли ребро по символу c в класс t*?" и положить в очередь новые классы.

Её несложно реализовать за $\mathcal{O}(E)$, тогда суммарное время работы алгоритма $\mathcal{O}(VE)$, так как `Split` вызовется не более $V-2$ раз.

5.5. Хопкрофт за $\mathcal{O}(E \log V)$

Можно реализовать `Split` оптимальнее. Главная идея:
1. реализовать `Split(t)` за $\mathcal{O}$(просматра входящих рёбер в t),
2. при разбиении класса на два добавлять в очередь только меньшую половину.

```
1  def Split(t, char):
2    cc++ # очищаем vertexMark[] за O(1)
3    allTypes = []
4    types = []
5    for v in A[t]:
6      for u in prev[v, char]:
7        if vertexMark[u] != cc:
8          vertexMark[u] = cc
9        t0 = type[u]
10       allTypes.add(t0)
11       B[t0].add(u) # для каждого типа t0 помним посещённую половину
12       if B[t0].size == 0:           types.add(t0)
13       if B[t0].size == A[t0].size: types.remove(t0)
14
15   for t0 in types: # те классы, которые поделились относительно (t, char)
16     if B[t0].size * 2 > A[t0].size: # если B[t0] - большая половина
17       B[t0].clear()
18       for u in A[t0]: # тратим времени O(B[t0].size), то есть, O(уже потраченного)
19         if vertexMark[u] != cc:
20           B[t0].add(u)
21     # теперь B[t0] - точно меньшая половина
22     for u in B[t0]: # перекрашиваем половину B[t0]
23       type[u] = typeN # номер нового класса - автоинкремент
24       A[typeN].add(u)
25       A[t0].add(u)
26     # Теперь старый класс t0 разбит на два новых (t0, typeN), кладём в очередь меньшую половину
27     q.add(typeN++)
28
29   for t0 in allTypes: # быстрое обнуление массивов B[]
30     B[t0].clear()
```

Время работы. Блок строк [15-26] работает за $\mathcal{O}$([5-13]). Блок [5-13] – перебор вершин $v \in A[t]$ и входящих в них рёбер. Если вершина $v \in A[t]$ на строке (5), то следующий раз мы положим её в очередь в составе в два раза меньшего класса $\Rightarrow$ переберём её не более $\log V$ раз $\Rightarrow$ каждое входящее рёбро переберём $\log V$ раз $\Rightarrow$ время работы $\mathcal{O}(E \log V)$.

Замечание 5.5.1. Здесь E – количество рёбер в автомате, дополненном до полного $\Rightarrow E = V \cdot |\Sigma|$.

5.6. Литература

[hopcroft,motwani,ulman'2001]. Книжка про автоматы. Минимизация в разделе 4.4, стр. 154.

Лекция #6: Суффиксный автомат

19 сентября

6.1. Введение, основные леммы

Будем обозначать "v – суффикс u" как $\boxed{v \subseteq u}$

Def 6.1.1. *Суффиксный автомат строки s, $SA(s)$* – min *по числу вершин детерминированный автомат, принимающий ровно суффиксы строки s, включая пустой.*

Def 6.1.2. $R_s(u)$ – *правый контекст строки u относительно строки s.*

$$R_s(u) = \{x \mid ux \subseteq s\}$$

Пример: $s = abacababa \Rightarrow R_s(ba) = \{cababa, ba, \epsilon\}$
Мы будем рассматривать правые контексты только от подстрок $s \Rightarrow R_s(v) \neq \varnothing$.

Def 6.1.3. $V_A = \{u \mid R_s(u) = A\}$ – *все строки с правым контекстом A.*

Def 6.1.4. $V(w)$ – *вершина автомата, в которой заканчивается строка w ($w \in V_A$).*

Утверждение 6.1.5. $\forall A$ все строки V_A заканчиваются в одной вершине суффавтомата.
Собственно вершины автомата, как и в 5.3.2 – классы V_A.

Следствие 6.1.6. Рёбра между вершинами проводятся однозначно:
$(\exists\ x \in V_A,\ xc \in V_B) \Leftrightarrow$ (между вершинами V_A и V_B есть ребро по символу "`c`").

Lm 6.1.7. $R_s(v) \cap R_s(u) \neq \varnothing,\ |v| \leqslant |u| \ \Rightarrow\ v \subseteq u$.

Доказательство. Возьмём $w \in R_s(v) \cap R_s(u)$, строки vw и uw – суффиксы s, отрежем w. ■

Lm 6.1.8. $R_s(v) = R_s(u),\ |v| \leqslant |u| \ \Rightarrow\ v \subseteq u$.

Lm 6.1.9. v – суффикс $u \Rightarrow R_s(u) \subseteq R_s(v)$ (у суффикса правый контекст шире).

Lm 6.1.10. $v \subseteq w \subseteq u,\ R_s(v) = R_s(u) \Rightarrow R_s(v) = R_s(w) = R_s(u)$ (непрерывность отрезания).

Следствие 6.1.11. $\forall A$ класс V_A определяется парой $s_{min} \subseteq s_{max}$: $V_A = \{w \mid s_{min} \subseteq w \subseteq s_{max}\}$.

Def 6.1.12. *Суффиксная ссылка $V(w)$ – вершина $V(z)$: $z \subseteq w,\ R_s(z) \neq R_s(w),\ |z| = \max$.*

`suf[V]` – суффиксная ссылка V_A
`len[V]` $= |s_{max}(V_A)|$

Lm 6.1.13. $|s_{min}(V_A)| =$ `len[suf[A]]` $+ 1$

Замечание 6.1.14. `suf[A]` корректно определена iff `len[A]` $\neq 0$.

Lm 6.1.15. У $SA(s)$ терминальными являются вершины `V(s)`, `suf[V(s)]`, `suf[suf[V(s)]]`, ...

Из 6.1.5 (вершины), 6.1.6 (рёбра), 6.1.15 (терминалы) мы представляем устройство суфавтомата.

Из лемм и 6.1.11 (вершина = отрезок суффиксов) 6.1.12 (суффссылка) мы получили инструменты для построения линейного алгоритма.

6.2. Алгоритм построения за линейное время

Алгоритм будет онлайн наращивать строку s. Начинаем с пустой строки $s = \varepsilon$.
Осталось научиться, дописывая к s символ a, от $SA(s)$ переходить к $SA(sa)$.

Будем в каждый момент времени поддерживать:

(a) `start` – $V(\varepsilon)$ (стартовая вершина)
(b) `last` – $V(s)$ (последняя вершина)
(c) `suf[V]` – для каждой вершины автомата суффссылку
(d) `len[V]` – для каждой вершины автомата максимальную длину строки
(e) `next[A,c]` – рёбра автомата

База: $s = \varepsilon$, `start` = `last` = 1.

Для того, чтобы понять, как меняется автомат, нужно понять, как меняются его вершины – правые контексты. Переход: $s \to sa \Rightarrow R_s(v) = \{z_1, \dots, z_k\} \to R_{sa}(v) = \{z_1 a, \dots, z_k a\}$ `+?` ε.

Пример 6.2.1. $s = abacabx,\ R_s(ab) = \{acabx, x\},\ R_{sa}(ab) = \{acabxa, xa\}$

Пример 6.2.2. $s = abacab,\ R_s(ba) = \{cab\},\ R_{sa}(ba) = \{caba, \varepsilon\}$

Lm 6.2.3. $(\varepsilon \in R_{sa}(v)) \Leftrightarrow (v \subseteq sa)$.

TODO

6.3. Реализация

TODO

6.4. Линейность размера автомата, линейность времени построения

TODO

Лекция #7: Линейное программирование

29 апреля 2019

7.1. Применение LP и ILP

Def 7.1.1. *Задача LP. Поиск* x: $Ax \leqslant b,\ x \geqslant 0,\ \langle c, x\rangle \to \max,\ x \in \mathbb{R}^n$

Def 7.1.2. *Задача ILP. Поиск* x: $Ax \leqslant b,\ x \geqslant 0,\ \langle c, x\rangle \to \max,\ \boxed{x \in \mathbb{Z}^n}$

При этом мы помним, как приводить к виду LP несколько похожих задач.

• LP: Кратчайшее расстояние в графе (от s до всех)

x_i — расстояние до вершины i, $x_s = 0$
Для каждого ребра $a \xrightarrow{w} b$ добавляем условие $x_b \leqslant x_a + w$
$\sum_i x_i \to \max$

• LP: Максимальный поток $s \to t$

Обратные рёбра не создаём, x_e — поток по ребру e.
$0 \leqslant x_e \leqslant capacity_e$ (второе добавляем в список неравенств)
Для каждой вершины $v \neq s, t$ добавляем условие $\sum x_{e \in in(v)} = \sum x_{e \in out(v)}$
$\sum_{e \in out(s)} x_e \to \max$

Немного магии: в симплекс-методе можно $0 \leqslant x_e$ бесплатно заменить на $0 \leqslant x_e \leqslant c_e$.

• LP: Поток размера k минимальной стоимости $s \to t$

$\sum_{e \in out(s)} x_e = k$
$\sum_e x_e cost_e \to \min$

• LP: Мультипродуктовый поток

Мы решаем на одной сети одновременно k задач "пустить из s_i в t_i F_i единиц потока". По каждому ребру e течёт одновременно $f_{e_1}, f_{e_2}, \ldots, f_{e_k}$, и должно выполняться общее ограничение $\forall e \sum_i f_{e_i} \leqslant capacity_e$. В отличии от предыдущих проще чем "сведём к LP" задача не решается.

• ILP: Два непересекающихся пути $A \to B, C \to D$

Задача NP-трудна, через поток размера два (склеить A и B, C и D) не делается.
Зато является частным случаем *целочисленного мультипродуктового потока.*

• ILP: Паросочетание в произвольном графе максимального веса

Каждому ребру сопоставляем переменную x_e — взяли ли мы ребро в паросочетание. $x_e \geqslant 0$.
Каждой вершине условие $\sum_{e \in adj(v)} x_e \leqslant 1$.
$\sum_e x_e cost_e \to \max$.

Замечание 7.1.3. На самом деле эту задачу можно решить за полином через LP

• LP: Паросочетание в двудольном графе максимального веса

Та же сеть, что в предыдущей, но благодаря двудольности матрица A задачи LP будет обладать свойством *тотальной унимодулярности*, из чего следует, что симплекс автоматически найдёт целочисленное решение.

• LP: Вершинное покрытие минимального веса

$0 \leqslant x_v$ – взяли ли вершину v, для каждого ребра (a, b) имеем $x_a + x_b \geqslant 1$. Цель: $\sum_v x_v w_v \to \min$.
$x_v \in \mathbb{Z} \Rightarrow$ решаем ILP, возвращаем $\{i \mid x_i = 1\}$, получили точное решение.
$x_v \in \mathbb{R} \Rightarrow$ решаем LP, возвращаем $\{i \mid x_i \geqslant \frac{1}{2}\}$, получили 2-приближение.

7.2. Сложность задач LP и ILP

Симплекс метод решает LP на большинстве тестов за полином, но в худшем всё же за экспоненту. Есть полиномиальные решения LP. Одно из них – *метод эллипсоидов*, им мы займёмся сегодня.

Lm 7.2.1. ILP $\in$ NP-hard

Доказательство. Сведём SAT. $0 \leqslant x_i \leqslant 1$, для каждого дизъюнкта $\sum x_{i_j} \geqslant 1$. ■

7.3. Геометрия и алгебра симплекс-метода

Система неравенств $Ax \leqslant b$ — пересечение полупространств. Такой объект называется полиэдром. Полиэдр выпукл, максимум любой линейной функции на нём достигается в вершине.

Lm 7.3.1. Для системы $Ax = b, x \geqslant 0, \langle c, x\rangle \to \max$ из m уравнений $\exists$ оптимальное решение, содержащее не более m ненулевых x_i.

Доказательство. Рассмотрим оптимальное решение x^* с максимальным числом нулевых компонент. Пусть число нулей $k \geqslant m+1$. Мы хотим подвигать x^* так, чтобы нули остались нулями, а x^* осталось решением. Решение системы "m уравнений $Ax = b$ и $n - k$ уравнений $x_i = 0$" или пусто, или хотя бы прямая ($dim = n - m - (n - k) = k - m \geqslant 1$). x^* — решение $\Rightarrow \exists$ прямая, пойдём по ней в сторону увеличения $\langle c, x\rangle$, пока не упрёмся в ограничение $x_j = 0$.
Получили решение, у которого $\langle c, x\rangle$ не хуже, а нулей больше. Противоречие. ■

Теорема 7.3.2. Корректность симплекса

Доказательство. Чем занимается симплекс? Перебирает наборы из m столбцов, которые соответствуют ненулевым переменным. Зафиксировав эти m столбцов и занулив оставшиеся переменные, мы однозначно получаем кандидата на оптимальное решение.
Конечность алгоритма: есть всего $\binom{m}{n}$ выборок, важно, чтобы они не повторялись.
Для этого мы используем правило Блэнда (доказательство) — брать min столбец.
Оптимальность решения после остановки симплекса: $\forall i\ c_i \leqslant 0 \Rightarrow \langle c, x\rangle \leqslant 0 \Rightarrow 0$ – оптимум. ■

• Симплекс перебирает вершины полиэдра

Если исходная задача имела форму $Ax \leqslant b, x \geqslant 0$, то область допустимых решений — полиэдр, а оптимум $\langle c, x\rangle$ достигается в вершине полиэдра. Симплексу мы скармливаем систему $Ax + y = b, x \geqslant 0, y \geqslant 0$. При этом и $y_i = 0$, и $x_i = 0$ означает, что одно из исходных неравенств обратилось в равенство (полупространство $\to$ плоскость). Из $n + m$ переменных x_i, y_i n обратятся в ноль, чем породят n плоскостей, пересечение которых — вершина полиэдра.

• Когда у симплекса есть вероятность застрять?

Если вершина полиэдра – не оптимум, из неё есть ребро, по которому функция увеличивается. Но симплекс может остаться на месте, лишь поменяв множество столбцов. Это значит, что вершина полиэдра есть одновременное пересечение больше чем n плоскостей.

7.4. Литература, полезные ссылки

[cormen]. Доступное для школьников описание симплекса.
[test]. Тест, на котором симплекс работает за экспоненту (Klee–Minty cube).
[efficiency]. Чуть подробнее про время работы симплекса (Hirsch Conjecture and so on).
[bland]. Правило Блэнда.
[max-babenko]. Хороший видео курс про линейное программирование от Максима Бабенко.

7.5. Перебор базисных планов

Понимание симплекса дало нам простейшее решение для LP – перебрать все $\binom{m}{n}$ базисных планов и для каждого пустить Гаусса. Такое решение можно использовать для тестирование. Заметьте, что до этого мы не знали вообще никаких решений задачи LP кроме симплекса.

7.6. Обучение перцептрона

В фундаменте нейронных сетей живёт один нейрон, он же перцептрон. Для решения некоторых задач классификации достаточно сети, состоящей лишь из одного нейрона.

Задача: даны точки $a_i \in \mathbb{R}^n$ и значения $y_i \in \{\pm 1\}$, найти гиперплоскость $b, x_1, \dots x_n$, что $\forall i\ sign(b + a_{i_1}x_1 + a_{i_2}x_2 + \dots + a_{i_n}x_n) = sign(y_i)$

Сразу упростим задачу
1. Добавим $\forall i\ a_{i_0} = 1$, обозначим $x_0 = b$.
2. Для всех $y_i = -1$ заменим a_i на $-a_i$, а y_i на 1. 3. Нормируем все a_i.

Теперь мы просто ищем такой вектор x, что $\forall i\ \langle x, a_i\rangle > 0$.

Решение: начнём с $x_0 = 0$, пока $\exists i_k\ \langle x_k, a_{i_k}\rangle \leqslant 0$, переходим к $x_{k+1} = x_k + a_{i_k}$.

$|a_i| = 1,\ |x_k|^2 = (x_{k-1} + a_{i_{k-1}})^2 = x_{k-1}^2 + 1 + 2\langle x_{k-1}, a_{i_{k-1}}\rangle \leqslant |x_{k-1}|^2 + 1 \leqslant k.$ (1)

x^* – искомый ответ, $|x^*| = 1,\ \langle x_k, x^*\rangle = \sum_j \langle a_{i_j}, x^*\rangle \geqslant k\alpha$, где $\alpha = \min_i \langle a_i, x^*\rangle > 0.$ (2)

$k\alpha \overset{(2)}{\leqslant} \langle x_k, x^*\rangle \leqslant |x_k| \overset{(1)}{\leqslant} \sqrt{k} \Rightarrow \sqrt{k} \leqslant \frac{1}{\alpha} \Rightarrow \boxed{k \leqslant \alpha^{-1/2}}$ ■

Проблема: уже при $n \geqslant 2$ мы можем построить тесты с α близким к нулю.

7.7. Метод эллипсоидов (Хачаян'79)

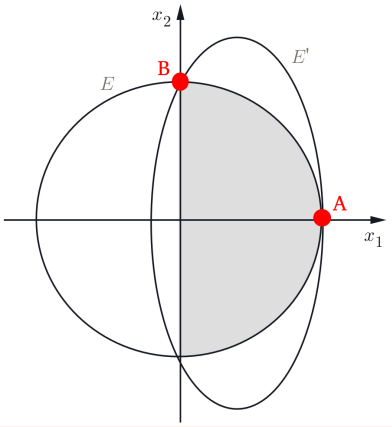
Решаем задачу $P = \{x \mid Ax \geqslant b\}$, найти $x \in P$, ограничение $P \neq \varnothing \Rightarrow Volume(P) > 0$.
Кроме A и b нам должны дать шар $E_0(x_0, r_0) \supseteq P$.

• Алгоритм

В каждый момент времени у нас есть эллипсоид $E_k(x_k, R_k)$: $x^T R x \leqslant 1$, содержащий P.
Если $x_k \in P$, счастье. Иначе выберем i_k: $\langle a_{i_k}, x\rangle < b$. $P \subseteq E_k \cap H_k$, где $H_k = \{x \mid \langle a_{i_k}, x\rangle \geqslant b\}$ (в половине эллипсоида по направлению a_{i_k} от центра). Теперь выпишем переход к $(k+1)$-му эллипсоиду в предположении, что E_k – единичная сфера, и $\forall i\ |a_i| = 1$.

$$x_{k+1} = x_k + \frac{1}{n+1} a_{i_k}$$
$$r_{k+1} = \left(\frac{n}{n+1}, \frac{n}{\sqrt{n^2-1}}, \dots, \frac{n}{\sqrt{n^2-1}}\right)$$
$$R_{k+1} = \begin{bmatrix} \frac{(n+1)^2}{n^2} & 0 & 0 \\ 0 & \frac{n^2-1}{n^2} & 0 \\ 0 & 0 & \ddots \end{bmatrix}$$

Где первая координата радиуса указана по направлению a_{i_k}.



• Математика: эллипсоиды

Эллипс: $\frac{x^2}{a^2} + \frac{y^2}{b^2} \leqslant 1$.
Добавим поворот и переход к $\mathbb{R}^n$: $z^t R z \leqslant 1$, где $z \in \mathbb{R}^n, R \in \mathbb{R}^{n\times n}, R \succ 0$.
Случай из $\mathbb{R}^2$ без поворота: $z = (x, y),\ R = \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{bmatrix}$.

• Математика: матрицы

$R \succ 0$ (положительно определена) $\Rightarrow R = B^t D B$ (здравствуй, Жорданова форма), где
B – ортонормированная матрица собственных векторов,
D – диагональная матрица собственных чисел (с точки зрения геометрии $D_{ii} = \frac{1}{r_i^2}$).
Более того $B^t = B^{-1}$ (это нормально для ортонормированных матриц).

Lm 7.7.1. Растяжение системы координат по направлению **z**

(1) Важно помнить, что если $f(x, R) = x^t R x$, то
$f(x + y, R) = f(x, R) + f(y, R),\ f(x, R + \Delta R) = f(x, R) + f(x, \Delta R)$.
(2) Фокус: $\forall z \in \mathbb{R}^n$: $|z| = 1$ матрица $S = zz^t$ обладает свойством $f(z, S) = 1$, $\forall v \perp z\ f(v, S) = 0$.
(1),(2) $\Rightarrow \forall A, |z|{=}1, \alpha{\in}\mathbb{R}, S = A + \alpha z^t z \quad f(z, S) = f(z, A) + \alpha$, $\forall v \perp z\ f(v, S) = f(z, S)$

• Математика: системы координат

Почему мы предполагали, что E_k – единичная сфера?
Это удобно, и мы всегда можем перейти к такой системе координат, где это верно:
$R = B^t D B, x^t R x = 1 \Rightarrow S = B^T D^{-1/2}, x = Sy, y^2 = 1$ (y – точка на единичной сфере).

• Обоснование алгоритма

Радиус $r_1 = \frac{n}{n+1}$ подобран так, чтобы точка A была покрыта: $\frac{1}{n+1} + \frac{n}{n+1} = 1$.
Радиус $r_2 = \frac{n}{\sqrt{n^2-1}}$ подобран так, чтобы точка B была покрыта: $\frac{x_1^2}{r_1^2} + \frac{x_2^2}{r_2^2} = \left(\frac{1}{n+1}\right)^2 \left(\frac{n+1}{n}\right)^2 + \frac{n^2-1}{n^2} = 1$.
Посмотрим на частное объёмов. Объём эллипсоида равен $f(n) r_1 r_2 \ldots r_n$, поэтому $\frac{V_{k+1}}{V_k} = \frac{r_1 r_2 \ldots r_n}{1 \cdot 1 \cdot \ldots 1} =$
$= \frac{n}{n+1} \left(\frac{n}{\sqrt{n^2-1}}\right)^{n-1} = \ldots \leqslant e^{-1/2(n+1)} \Rightarrow$ за $2(n{+}1)$ шагов объём уменьшится хотя бы в e раз.

$\Rightarrow$ количество шагов не более $2(n{+}1) \cdot \ln(Volume(E_0)/Volume(P))$.
Хорошая новость: это полином от n.
Плохая новость: в худшем это $\mathcal{O}(n^4 \log(nU))$.
Пояснение: $Volume(E_0) \leqslant (nU)^{\Theta(n^2)}$, $Volume(P) \geqslant (nU)^{-\Theta(n^3)}$. U – ограничение на $a_{ij}, b_i \in \mathbb{Z}$.

• Время работы алгоритма

Один шаг работает за $mn + n^2$.
m – количество проверок $\langle a_i, x_k \rangle \geqslant b_i$.
n^2 – время пересчёта матрицы система координат.

Требуемая точность вычислений - $n^3 \log U$ цифр.

Итого: $\mathcal{O}^*(n^9)$, где $9 = 2 + 3 + 4$.

• **Псевдокод**

```
# Наш эллипсоид задаётся уравнением (x − x_0)^t D^{-1} (x − x_0) = 1
# Напомним, что D ≻ 0 ⇒ D = A^T R A, D^{-1} = A^T R^{-1} A, где R - диагональная
x_0 = [0,0,0,...,0]
D[i,i] = 10^20 # ∞
while True: # Предполагаем, что ответ существует
  find i : ∑_j a[i,j]x_0[j] < b[i] # O(nm)
  if i does not exist : return x_0 # Нашли точку, удовлетворяющую всем неравенствам
  z = D*a_i # O(n^2), z - нормаль a_i в другой системе координат
  len = (a_i^t*D*a_i)^{1/2} # O(n^2)
  x_0 += 1/(n+1) z/len # O(n)
  D = n^2/(n^2-1)*(D - 2/(n+1) z*z^t/len^2) # O(n^2), растянули всё в n/√(n^2-1) раз и
                                            # поменяли радиус по направлению z, см. 7.7.1
```

• **Обобщение**

Можно применить не только к ∩ полупространств, но и к ∩ ∀ выпуклых множеств P_i.
Нам достаточно уметь проверять $x_k \in P_i$ и искать опорную плоскость к P_i.

7.8. Литература, полезные ссылки

[MIT'09 | ellipsoid]. Краткое, но полное описание метода эллипсоидов.
[Florida'07 | ellipsoid]. Строгое описание метода эллипсоидов с кучей ссылок по теме.
[Кормен'2013, разделы 29.2 и 35.4]. Примеры задач на LP.

Лекция #8: Паросочетание в произвольном графе

13 мая 2019

8.1. Полезные данные из прошлого

Lm 8.1.1. Лемма о дополняющем пути. Если паросочетание M не максимально, то существует дополняющий, чередующийся относительно M, путь (ЧДП), увеличивающий M.

Lm 8.1.2. Корректность Куна. Алгоритм последовательно увеличивает M дополняющими путями. $\forall v$ если в какой-то момент $\nexists$ дополняющего чередующегося пути из v, то это навсегда.

Lm 8.1.3. Симметрические разности.
M_1, M_2 – паросочетания, тогда $M_1 \triangle M_2$ – набор путей и циклов,
причём в $M_1 \triangle M_2$ есть хотя бы $\big||M_1| - |M_2|\big|$ ЧДП.
P – чередующийся относительно M_1 путь (не важно, дополняющий ли), тогда $M_1 \triangle P$ – тоже паросочетание, причём $|M_1 \triangle P| = |M_1| + \Delta P$, где $\Delta P = +1$ для ЧДП, 0 для чётного пути.

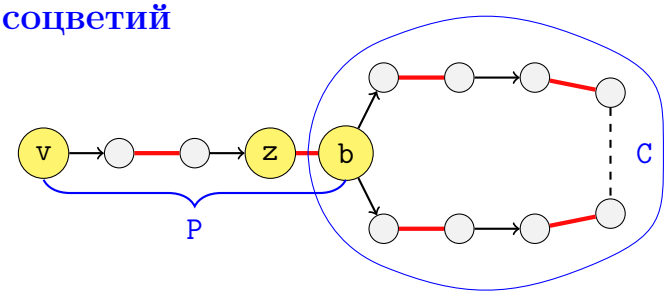
Lm 8.1.4. Поиск дополняющего пути в двудольном графе.
Чтобы из свободной вершины v 1-й доли найти дополняющий путь, запустим dfs/bfs на орграфе, где из 1-й доли ведут все рёбра, а из 2-й доли только рёбра паросочетания.

- Первые три леммы верны для произвольного графа.
- Последнюю мы сможем применить, добавив в алгоритм случай “найден нечётный цикл”.

Упражнение 8.1.5. G – произвольный граф. M_1, M_2 – паросочетания в нём. Есть ли в $M_1 \triangle M_2$ нечётные циклы? *(конечно, нет, в сим.разности только чередующиеся циклы)*

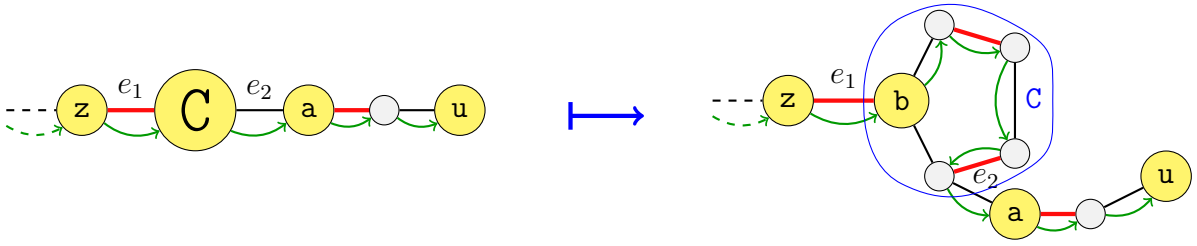
8.2. Алгоритм Эдмондса сжатия соцветий

Берём Куна и добавляем в `dfs/bfs` случай “найден нечётный цикл”. Кстати, мы нашли не просто нечётный цикл, а чередующийся нечётный цикл C со стеблем P.



Теорема 8.2.1. Теорема Эдмондса⁺ [1]. Пусть стебель P начинается в v.
$G' = G/_C$ (стянули цикл C в одну вершину) $\Rightarrow$ в G есть ЧДП из v iff в G' есть ЧДП из v

Доказательство. В G' есть путь $T \Rightarrow$ в G есть.
Если путь T не проходит через вершину C, не меняем его.
Иначе заметим, что в T вершине C смежны два ребра – $e_1 = (z, b) \in M$ и $e_2 \notin M$. Расжимаем.



■

[1] На e-maxx используется слишком слабая версия теоремы, нам нужен путь именно из v.

Доказательство. В G есть путь $T\colon v \rightsquigarrow u \Rightarrow$ в G' есть путь $T'\colon v \rightsquigarrow u$.
Сперва пусть $P = \{b\}$. Возьмём в T последнее ребро, исходящее из цикла ($e\colon z \to w, z \in C$).
b не покрыта $M \Rightarrow$ все рёбра из $C \notin M \Rightarrow e \notin M$. Отрежем от T кусок до e, получили путь T'.

В общем случае применим предыдущую идею к паросочетанию $M_1 = M \triangle P$ и пути T_1, взятого из $(M \triangle T) \triangle M_1$. Получили T_1'. T' найдётся в $(M_1 \triangle T_1') \triangle M$.

Путь	Относительно	Где	Куда	Как появился?
T	M	G	$v \to u$	Дан по условию.
T_1	$M_1 = M \triangle P$	G	$b \to u$	Найден в $(M \triangle T) \triangle M_1$.
T_1'	$M_1 = M \triangle P$	G'	$C \to u$	Отрезали от T_1 часть до *последнего ребра, исходящего из* C.
T'	M	G'	$v \to u$	Найден в $(M_1 \triangle T_1') \triangle M$.

Lm 8.2.2. Для тех, чей мозг плавится под воздействием симметрических разностей.
Пусть $A(M)$ – свободные вершины относительно паросочетания M.
Концы всех путей в $M_1 \triangle M_2$ лежат в $A(M_1) \triangle A(M_2)$.

Таблица, чтобы понять, почему именно такие пути мы нашли в $(M \triangle T) \triangle M_1$ и $(M_1 \triangle T_1') \triangle M$:

Паросочетание	Свободные вершины	Покрытые вершины	Граф	Как получили
M	$v\ u$	$b\,(C)$	G/G'	Дано по условию.
M_1	$b\,(C)\ u$	v	G/G'	Поксорили M со стеблем.
$M \triangle T$		$v\ b\ u$	G	Увеличили паросочетание M путём T.
$M_1 \triangle T_1'$		$v\ C\ u$	G'	Увеличили паросочетание M_1 путём T_1'. ■

8.3. Реализация за $\mathcal{O}(V^3)$

1. `for v=1..n` запустим `dfs/bfs(v)`, доставшийся нам от Куна, для поиска пути.
2. Поиск пути нашёл путь или нечётный цикл со стеблем (соцветие = цветок + стебель).
3. Сожмём нечётный цикл в одну вершину, продолжим поиск. На слове *продолжим* мы понимаем, что из `dfs/bfs` удобнее `bfs`.
   (a) Сжимаем нечётный цикл в новую вершину z за $\mathcal{O}(V \cdot$ `cycleLen`$)$.
   (b) Кидаем z в очередь, удаляем все вершины цикла из очереди.
   (c) Делаем рекурсивный вызов `continueBfs`, который возвращает путь в сжатом графе.
   (d) Если путь проходит через z, делаем расжатие пути. Возвращаем путь в исходном графе.

`continueBfs` – тот же `bfs`, но не обнуляем очередь и пометки.
Время работы = $V \cdot$ `bfs` = $V(E+V^2)$: E – просмотрели рёбра по разу, $\mathcal{O}(V^2) = \mathcal{O}(V \cdot \sum_i$ `cycleLen`$_i)$

8.4. Красивая простая реализация Эдмондса (Габов'1976)

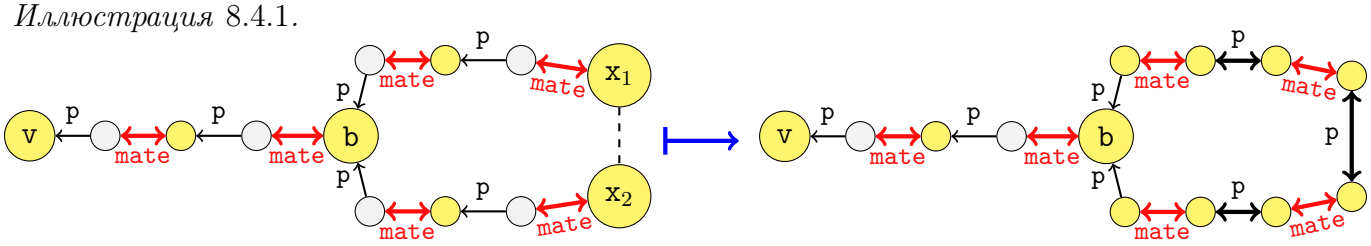
Основная схема та же, что в Куне:

```
1 mate = [-1,-1,...,-1] # mate[v] - пара в паросочетании
2 for v in 1..n
3   if mate[v] == -1
4     endOfPath = bfs(v)
5     if endOfPath != -1
6       mate = mate △ recoverPath(endOfPath)
```

`bfs` по ходу работы помечает вершины 1-й доли `used[v] = 1`.
Вершины 2-й доли – пары вершин 1-й доли.
Для $\forall$ вершины `w` 2-й доли храним предыдущую `p[w]` в 1-й доли.
Таким образом `w`, `p[w]`, `mate[p[w]]`, `p[mate[p[w]]]`, ... – путь до `w` $\rightsquigarrow$ `v`.
Рёбра $\{$`w` $\to$ `mate[p[w]]` $\mid$ `used[w]` $= 1\}$ образуют дерево T с корнем в `mate[v]`.

Интересен случай, когда мы пытаемся пойти из помеченной вершины $\mathtt{x}_1$ в помеченную вершину $\mathtt{x}_2$. Тогда налицо нечётный цикл C, причём до всех вершин этого цикла кроме основания мы научились доходить, как до вершин 1-й доли.

Иллюстрация 8.4.1.



Замечание 8.4.2. Здесь жёлтым отмечены вершины, до которых мы умеем доходить, как до вершин первой доли. Они же лежат в очереди.

• Главная идея

Сжимая цикл C, вместо того, чтобы создавать новую вершину и видоизменять граф, лишь запомним, что все вершины C лежат именно в C. Для этого будем $\forall$`v` поддерживать `base[v]` – основание цикла, в котором лежит `v`.

• Алгоритм действий

1. За $\mathcal{O}(n)$ выделить в дереве T вершину `b` $=$ `lca`$(\mathtt{x}_1, \mathtt{x}_2)$.
2. Пусть $\mathtt{b}_i$ – первая вершина на пути $\mathtt{x}_i \rightsquigarrow$ `b`: `base[`$\mathtt{b}_i$`]` $=$ `base[b]`.
   Пройти по путям P_1: $\mathtt{x}_1 \rightsquigarrow \mathtt{b}_1$ и P_2: $\mathtt{x}_2 \rightsquigarrow \mathtt{b}_2$ и проставить новые ссылки `p[]`.
   $\forall$`z` $\in P_1 \cup P_2$: `used[z] = 0` пометить z и добавить в очередь `bfs`-а.
   Важно: менять `p[`$\mathtt{b}_1$`]` и `p[`$\mathtt{b}_2$`]` нельзя, также нужно идти именно до b_1 и b_2 (до b нельзя, иначе мы можем случайно поменять `p[`$\mathtt{b}_1$`]`).
3. $\forall$`u` $\in P_1 \cup P_2$ $\forall$`w`: `base[w]` $=$ `base[u]` сделать `base[w] := b`.

8.5. Оптимизации

Как и для Куна, основные оптимизации – не удалять пометки и жадно инициализировать паросочетание. Важно, что второе толком не работает без первого.

- Удаляем пометки, пока не нашли ЧДП $\Rightarrow$ $\mathcal{O}(V \cdot$ `bfs`$)$ превратилось в $\mathcal{O}(|M| \cdot$ `bfs`$)$.
- Жадно находим за $\mathcal{O}(E)$ паросочетание размером $\geqslant \frac{|M|}{2}$ $\Rightarrow$ ускорили ещё в два раза.

8.6. DSU и $\mathcal{O}(VE \cdot \alpha)$

Научимся случай "нечётный цикл" обрабатывать за $\mathcal{O}(k \cdot \alpha)$, где k – длина цикла в сжатом графе. Тогда суммарное время работы `bfs` будет $\mathcal{O}((V+E) \cdot \alpha)$.

• Поиск LCA

Во-первых, мы будем переходить не `v` $\to$ `p[mate[v]]`, а сразу `v` $\to$ `p[mate[base[v]]]`.
Во-вторых, LCA(x_1, x_2) можно искать не за $\mathcal{O}(n)$, а за $\mathcal{O}(k)$, идя от x_1 и x_2 двумя указателями.

• **Собственно сжатие цикла**

`base[v]` $\to$ `DSU.get(v)`. Какие множества нужно передать `DSU.join`? Повторим код LCA, только теперь идём ровно до базы b. Объединяем все пройденные `base[v]`, `mate[base[v]]` с b.

• **Добавление вершин в очередь**

Если вершина лежит в уже сжатом цикле, она точно в очереди.
Иначе это `mate[base[v]]`. Каждую такую попробуем добавить.

• **Обновление ссылок `p[]`**

Если мы хотим просто проверить наличие дополняющего пути, то ссылки `p[]` вообще не нужны. Для восстановлении пути основная идея – хранить только остовное дерево `bfs`, то есть, не переприсваивать `p[]` и не проставлять ссылки `p[]` вершинам, которые мы изначально определили во 2-ю долю, и лишь при сжатии цикла были добавлены в очередь. Для таких вершин y 2-й доли мы запомним, что добавили их в очередь, когда сжимали цикл, образованный ребром (v, x), и откатываясь по пути из v до базы цикла. Тогда чтобы продолжить восстанавливать путь из y, пройдём кусок $v \to y$, развернём его, а затем продолжим восстанавливать путь из x.

• **Реализация**

[rkolganov]. `dfs`-реализация за $\mathcal{O}(VE \cdot \alpha)$.

8.7. Реализации через `dfs`

Если мы хотим вместо `bfs` использовать `dfs`, то при сжатии все вершины цикла нужно положить в `vector`, пометить, а потом по очереди сделать от каждой рекурсивный вызов.

У `dfs` есть огромный плюс – когда мы идём из `v` в `x` и обнаруживаем нечётный цикл, то `LCA(v, x) = inTime[v] < inTime[x] ? base[v] : base[x]`; (т.е. ищем LCA за $\mathcal{O}(1)$).

8.8. Альтернативное понимание реализации

Попробуем вообще не думать про нечётные циклы. Будем поддерживать массивы `p[]`, `mate[]`.

```
1 v = q.pop()
2 for e : v --> x
3   if mate[x] == -1: return # нашли путь, молодцы
4   if !used[mate[x]]:
5     p[x] = v
6     used[mate[x]] = 1
7     q.push(mate[x])
```

Проблема такого подхода только в том, что при восстановлении пути по ссылкам `p[]`, мы можем получить не простой путь. Эту проблему можно попробовать костыльно разрешить, в явном виде проверяя при `p[x] = v`, что `x` нет в пути от `v` до корня. Окажется, что тогда мы некоторые пути не найдём, но в простых случаях, как 8.4.1, отработаем корректно. Можно попробовать более мощный костыль, обработать таки случай “нечётный цикл”, пройдя ровно до LCA и проставив ссылки. Это как раз пример **TODO**, на котором мы показывали, почему важно идти до b_i, но не b. Так вот: массив `base[]`, имеющий тайный смысл “база нечётного цикла”, – простейший корректный способ разрешить сложности с `p[]`.

8.9. Литература, полезные ссылки

[e-maxx]. Классическое описание алгоритма Эдмондса и реализации Габова.
[Galil'1986]. Экскурс во всё известное об алгоритмах для паросочетаний на 86-й год. В нашем курсе мы не выходим за рамки этих результатов и даже не затрагиваем самые крутые.

8.10. Исторический экскурс

[Gabow'1976]. Ph.D. Гарольда Габова'72. Простая и красивая реализация Эдмондса за $\mathcal{O}(V^3)$.

• **Попытки обобщить Хопкрофта-Карпа для произвольного графа.**
[Even&Kariv'1975]. Научились делать одну фазу Хопкрофта-Карпа за $\mathcal{O}(V^2 + E \log V)$. Получили $\mathcal{O}(\min(V^{5/2}, EV^{1/2} \log V))$. Достойный претендент на *ACM Longest Paper Award*. Ивен был научником Харива, который через год оформил эту работу своим Ph.D.
[Micali&Vazirani'1980]. MV80. Научились одну фазу делать за $\mathcal{O}(E)$, получили $\mathcal{O}(EV^{1/2})$.
[Peterson'1988]. Автор утверждает, что смог сделать "понятное изложение" MV80.
[Vazirani'2014]. Ребята наконец оформили строгое доказательство своего мегаалгоритма =)
[Gabow'2017]. Современная реализация алгоритма за $\mathcal{O}(EV^{1/2})$ на github.

Лекция #9: Двойственность, целочисленность

15 мая 2019

• **Общая идея**

Если у нас есть уравнения/неравенства, их можно умножать на константу и складывать, получая новые. Если именно неравенства, следует использовать положительные коэффициенты, чтобы неравенства не меняли знак. Если i-ю строку A умножаем на y_i, получаем y^tA.

9.1. Формулировка задачи

• $Ax = b, x \geqslant 0,\ \langle c, x\rangle \to \max$
Возьмём i-е уравнение с коэффициентом y_i так, чтобы получилось $y^tA \geqslant c$,
тогда из $x \geqslant 0$ имеем $\langle c, x\rangle \leqslant \langle y, b\rangle$.

• $Ax \leqslant b, x \geqslant 0,\ \langle c, x\rangle \to \max$
Возьмём i-е неравенство с коэффициентом $y_i \geqslant 0$ так, чтобы получилось $y^tA \geqslant c$,
тогда из $x \geqslant 0$ имеем $\langle c, x\rangle \leqslant \langle y, b\rangle$.

• $Ax \leqslant b,\ \langle c, x\rangle \to \max$
Возьмём i-е неравенство с коэффициентом $y_i \geqslant 0$ так, чтобы получилось $y^tA = c$,
тогда имеем $\langle c, x\rangle \leqslant \langle y, b\rangle$.

Теорема 9.1.1. Слабая теорема двойственности: во всех трёх случаях $\min\langle y, b\rangle \geqslant \max\langle c, x\rangle$.

Теорема 9.1.2. Сильная теорема двойственности: во всех трёх случаях $\min\langle y, b\rangle = \max\langle c, x\rangle$.

Прямая	Двойственная
$\langle c, x\rangle \to \max,\ Ax \leqslant b, x \geqslant 0$	$\langle b, y\rangle \to \min,\ y^tA \geqslant c, y \geqslant 0$
$\langle c, x\rangle \to \max,\ Ax \leqslant b$	$\langle b, y\rangle \to \min,\ y^tA = c, y \geqslant 0$
$\langle c, x\rangle \to \max,\ Ax = b, x \geqslant 0$	$\langle b, y\rangle \to \min,\ y^tA \geqslant c$

Замечание 9.1.3. Обозначим за A^* двойственную к задаче A, тогда $A^{**} = A$.

9.2. Доказательство сильной двойственности

Посмотрим внимательно на симплекс-метод, будем тащить за собой решения и прямой, и двойственной задач. Заметим, что (а) симплекс корректен, (б) в конце решения равны.

Подробнее рассмотрим на задаче $Ax = b,\ x \geqslant 0,\ \langle c, x\rangle \to \max$.
В каждый момент в строке c' хранится $c - \sum y_i a_i$. Здесь a_i – i-я строка A.
Алгоритм завершается, когда $\forall i\ c'_i \leqslant 0 \Leftrightarrow c - \sum y_i a_i \leqslant 0 \Leftrightarrow y^tA = \sum y_i a_i \geqslant c$.
В момент завершения $x^* = (b_1, b_2, \dots b_m, 0, \dots, 0)$, $y^* = (0, \dots, 0)$. $\langle x^*, c\rangle = 0$, $\langle y^*, b\rangle = 0$.
По слабой теореме двойственности решения x^* и y^* оптимальны.
При переходе (смена базиса) мы $\langle x, c\rangle$ и $\langle y, b\rangle$ меняем на одинаковую величину $\Rightarrow$
у изначальной задачи оба решения оптимальны, и верна 9.1.2. ■

Замечание 9.2.1. Мы научили симплекс искать решения сразу двух задач.
Для этого нужно для каждой строки текушей матрицы A и строки c поддерживать вектор коэффициентов исходной матрицы A. Теперь мы за то же время находим сразу пару (x, y).

9.3. Целочисленность решения

Def 9.3.1. *Минор матрицы A – определитель квадратной подматрицы A (выбрали произвольное k, k строк и k столбцов матрицы A, получили $B\colon k \times k$, $\det B$ называется минором A).*

Def 9.3.2. *Унимодулярной называется матрица $A \in \mathbb{Z}^{n\times n}\colon \det A = \pm 1$.*

Lm 9.3.3. Если $A \in \mathbb{Z}^{n\times n}$ унимодулярна, то $\forall b \in \mathbb{Z}^n$ у системы $Ax = b$ $\exists!$ решение x^* и $x^* \in \mathbb{Z}^n$

Доказательство. $\det A \neq 0 \Rightarrow \exists!$. По Крамеру $x_i^* = \frac{\Delta_i}{\det A} \in Z$. ■

Def 9.3.4. *Тотально унимодулярной (TU) называется матрица $A \in \mathbb{Z}^{n\times m}$, если все её миноры принимают значения из $\{0, \pm 1\}$*

Теорема 9.3.5. $Ax = b,\ x \geqslant 0,\ \langle c, x\rangle \to \max$, A – TU $\Rightarrow$ симплекс найдёт $x \in \mathbb{Z}^n$.

Доказательство. Симплекс находит одну из вершин полиэдра, все вершины есть решения систем $A'x = b'$, где A' – невырожденный набор столбцов A. $\det A' = \pm 1 \Rightarrow x \in \mathbb{Z}^n$. ■

Теорема 9.3.6. $Ax \leqslant b,\ x \geqslant 0,\ \langle c, x\rangle \to \max$, A – TU $\Rightarrow$ симплекс найдёт $x \in \mathbb{Z}^n$

Доказательство. Приводя задачу к форме $A'x = b$, мы лишь добавили столбцы, содержащие одну единицу и нули. $\det \left[\begin{array}{c|c} a_1 \\ a_2 \\ \cdots \\ a_m \end{array} \begin{array}{c} \\ \\ \\ \\ \end{array} Q\right] = \sum_i (-1)^i a_i Q_i$, где Q_i – минор Q, полученный удалением i-й строки $\Rightarrow$ миноры A' есть (миноры A) $\cdot \pm 1$. ■

9.4. Паросочетания

Вспомним формулировку задачи через линейное программирование.

$$0 \leqslant x_e,\ \forall v \sum_{e\in N(v)} x_e \leqslant 1,\ \sum_e w_e x_e \to \max \tag{1}$$

Запишем двойственную.

$$0 \leqslant y_v,\ \forall a \xrightarrow{e} b\ \ y_a + y_b \geqslant w_e,\ \sum_v y_v \to \min \tag{2}$$

Для двудольного графа мы решали эти задачи так:
x найдёт Венгерка, y найдётся из потенциалов Венгерки.
Теперь мы знаем, что задачи связаны сильной теоремой двойственности.

Теорема 9.4.1. Матрица инцидентности I двудольного графа тотально унимодулярна.

Доказательство. Пусть столбцы I – рёбра, строки – вершины. Рассмотрим квадратную подматрицу A матрицы I. Выбранные множества вершин и рёбер обозначим U и X. Если есть столбец из нулей, $\det A = 0$. Если есть столбец из одной единицы, в $\det A$ есть лишь одна ненулевая перестановка... удалим столбец и строку, соответствующую единице.

Остался случай "в каждом столбце ровно 2 единицы" $\Rightarrow$ матрице A соответствует набор циклов. $\det A = \sum_\pi (-1)^{sign(\pi)} \prod a_{i\pi_i}$. Если $\prod a_{i\pi_i} \neq 0 \Rightarrow$ мы каждой вершине из U выбрали ребро из X. Для цикла это можно сделать двумя способами (ориентируем цикл), при этом, если развернуть цикл, для чётного цикла $sign(\pi)$ поменяется, для нечётного не поменяется.
Итого, если A – чётный цикл, $\det A = 0$, если же A – нечётный цикл, $|\det A| = 2$.
Если A – набор циклов, то $\det A = \prod_i \det C_i$, где C_i – матрица i-го цикла. ■

Замечание 9.4.2. (A тотально унимодулярна) $\Leftrightarrow$ (A^t тотально унимодулярна)

Теорема 9.4.3. Симплекс найдёт целочисленное решение для задачи о взвешенном паросочетании в двудольном графе.

Доказательство. Запишем задачу в форму ЛП (1), применим теоремы 9.4.1 и 9.3.6. ■

9.5. Частные случаи ЛП

• **Уравнений больше чем неизвестных,** $Ax = b,\ m > n$

Гаусс уменьшит размер задачи, оставит $m' \leqslant n$ уравнений и приведёт задачу к начальному виду для симплекса.

• **Есть очень много переменных, нет неотрицательности,** $Ax \leqslant b,\ m < n$

Во-первых, мы умеем переходить к двойственной задаче $A^t y = c, y \geqslant 0$.
После транспонирования A мы получаем $m' > n'$ и можем применить предыдущую идею.

Во-вторых, можно Гаусса натравить на столбцы A и сделать их линейно независимыми.
При этом важно что, когда мы находим линейную зависимость столбцов $\sum_j col_{i_j} = 0$,
если $\sum_j c_{i_j} \neq 0$, то ($\exists$ решение $\Rightarrow$ max неограничен).

• **Число переменных** $\mathcal{O}(1)$

Тогда есть рандомизированный линейный от m алгоритм для решения задачи ЛП.

9.5.1. Рандомизированный алгоритм пересечения полупространств

Пересекаем k d-мерных полупространств $\sum a_{ij}x_j \leqslant b_i$. Ищем максимум $\sum x_j c_j$.
Чтобы ограничить ответ, предположим, что он лежит в *bounding box*: $\forall j\ |x_j| \leqslant C$.
Изначально берём одну из 2^d вершин куба, на которой достигается максимум.
После этого в порядке `randomShuffle` по одному добавляем полупространства.

```
1  def solve(k, d, halfspaces):
2    p = (0,..., 0)
3    for v in vertices(±C): # O(2^d)
4      if p*c < v*c:
5        v = p
6    randomShuffle(halfspaces)
7    for i = 1..halfspaces.n:
8      a, b = halfspaces[i]
9      if p*a <= b:
10       continue # точка p оптимальна для первых i-1, лежит во всех i
11     # иначе оптимальная p* обязательно лежит на halfspaces[i]
12     p = solve(i-1, d-1, проекции первых i − 1 полупространств на halfspaces[i])
```

Lm 9.5.1. На строку 12 мы попадаем с вероятностью $\frac{d}{i}$.

Доказательство. Оптимальная точка образована пересечением d плоскостей. Как только мы переберём их всех, ответ будет посчитан корректно и в процессе перебора первых i плоскостей уже не поменяется. Значит, событие "*мы пересчитываем точку p на i-м шаге*" случается iff "на i-й позиции в `halfspaces` лежит одна из тех d плоскостей". Вероятность этого ровно $\frac{d}{i}$. ■

• Матожидание времени работы

База $d = 1$, пересечь k лучей мы сможем за $\mathcal{O}(k)$. =)
Для $d = 2$ получили формулу $T = \sum_i (1 + \frac{2}{i} \cdot i) = 3k$.
По индукции для $\forall d$ получаем $\mathcal{O}(k \cdot d!)$: $T(k, d) = \sum_i (1 + \frac{d}{i} \cdot T(i, d{-}1)) = \sum_i (1 + \frac{d}{i} \cdot (i \cdot (d{-}1)!))$.

Замечание 9.5.2. Есть аналогичный алгоритм для пересечения шаров. Тоже за $\mathcal{O}(k \cdot d!)$.

9.6. Матричные игры

• Правила игры

Дана матрица $A \in \mathbb{R}^{n \times m}$.
Первый выбирает строку i, второй столбец j, оба не знают выбор противника.
Результат игры – ячейка a_{ij}. Первый $a_{ij} \to \max$. Второй $a_{ij} \to \min$.

• Детерминированные стратегии

При поиске оптимальной стратегии важно зафиксировать пространство решений.
Начнём с простейшего: стратегии первого – $\{i\}$, стратегии второго – $\{j\}$.
На каждую конкретную стратегию есть контрстратегия.

Пример: $\begin{bmatrix} 10 & \boxed{1} \\ 2 & 4 \end{bmatrix} \Rightarrow$ первый хочет получить 10, выбирает 1-ю строку, если второй узнает, он подсунет 2-й столбец.

• Осторожная жадная стратегия

Первый игрок гарантирует себе $\max_i(\min_j a_{ij}) = \alpha$. При $\forall$ игре второго он получит $\geqslant \alpha$.
Второй игрок гарантирует себе $\min_j(\max_i a_{ij}) = \beta$. При $\forall$ игре первого он получит $\leqslant \beta$.

Пример: $\begin{bmatrix} 10 & 1 \\ 2 & 4 \end{bmatrix} \Rightarrow$ первый гарантирует себе $\alpha = 2$, второй $\beta = 4$.

• Вероятностная стратегия

Введём новое пространство решений.

Def 9.6.1. *Вероятностная стратегия 1-го игрока – вероятности строк* $p_1, \ldots, p_n$: $\sum p_i = 1$

Def 9.6.2. *Стратегия 2-го игрока – вероятности столбцов* $q_1, \ldots, q_m$: $\sum q_j = 1$

Если игроки играют несколько раз и каждый раз делают выбор строки/столбца в соответствии со своим вектором вероятностей, у каждой игры будет свой результат, в среднем столько:

Def 9.6.3. *Результат игры* $F(p, q) = \sum_{ij} p_i a_{ij} q_j$ *(матожидание* a_{ij}*).*

Пойдём опять по пути осторожной жадной стратегии.
Первый хочет найти такую стратегию p^*: $\min_q(\sum_{ij} p_i^* a_{ij} q_j) = \min_j(\sum_i p_i^* a_{ij}) \to \max = \alpha$.
Второй в свою очередь ищёт q^*: $\max_i(\sum_j a_{ij} q_j^*) \to \min = \beta$.
В результате первый гарантирует себе α: $\forall q\ F(p^*, q) \geqslant \alpha$, а второй β: $\forall p\ F(p, q^*) \leqslant \beta$.

Пример: $\begin{bmatrix} 10 & 1 \\ 2 & 4 \end{bmatrix} \Rightarrow \begin{cases} p = (\frac{2}{11}, \frac{9}{11}),\ q = (\frac{3}{11}, \frac{8}{11}) \\ \alpha = \min(10p_1 + 2p_2, 1p_1 + 4p_2) = \min(\frac{38}{11}, \frac{38}{11}) = \frac{38}{11} \\ \beta = \min(10q_1 + 1q_2, 2q_1 + 4q_2) = \min(\frac{38}{11}, \frac{38}{11}) = \frac{38}{11} \end{cases}$

Итого, пользуясь вероятностной стратегией, оба игрока гарантируют себе результат $\frac{38}{11} \approx 3.5$.

Теорема 9.6.4. Для $\forall$ матрицы A $\alpha = \beta$

Теорема утверждает, что полученный результат $\sum_{ij} p_i a_{ij} q_j$ оптимален для обоих игроков.

Доказательство. Запишем задачу первого игрока в форме ЛП:

$$\forall j \sum p_i a_{ij} \geqslant t,\ p_i \geqslant 0,\ \sum p_i = 1,\ t \to \max$$

$$A' = \left[\begin{array}{c|c} -A^t & \begin{matrix}1_1\\ \vdots \\ 1_m\end{matrix} \\ \hline & \end{array}\right], \quad A' \cdot \begin{bmatrix} p_1 \\ \vdots \\ p_n \\ t \end{bmatrix} \leqslant b = \begin{bmatrix} 0_1 \\ \vdots \\ 0_m \end{bmatrix}, \quad c = \begin{bmatrix} 0_1 \\ \vdots \\ 0_n \\ 1 \end{bmatrix}$$

Запишем двойственную задачу. Неравенства умножаем на $q_j \in \mathbb{R}^{\geqslant 0}$, равенство на $s \in \mathbb{R}$.
Для первых n компонент c получаем неравенство (т.к. $p_i \geqslant 0$), для последней равенство.

$$\sum_j q_j \cdot 1 = 1,\ \forall i = 1..n \sum_j q_j \cdot (-a_{ij}) + s \geqslant 0 \Leftrightarrow \sum_j q_j a_{ij} \leqslant s$$
$$q_1 b_1 + \ldots q_m b_m + s \cdot 1 = 0 + \cdots + 0 + s \to \min$$

■

Теорема 9.6.5. Равновесие Нэша. $\exists$ стратегии p^*, q^*: $\forall p\ F(p, q^*) \leqslant F(p^*, q^*), \forall q\ F(p^*, q) \geqslant F(p^*, q^*)$

"Есть такие вероятностные стратегии для первого и второго игроков, что ни один из них не может улучшить для себя результат игры, изменив только свою стратегию".

Доказательство: по теореме 9.6.4 подойдут p и q, полученные осторожной жадной стратегией.

■

Замечание 9.6.6. Достижение Джона Нэша в том, что он доказал существование хотя бы одного равновесия для произвольных дискретных некооперативных игр (wiki) для n игроков.

Лекция #10: Факторизация

22 мая 2019

10.1. Метод Крайчика

Как обычно, наша задача – найти любой нетривиальный делитель x.
Метод Ферма: найдём v, u: $v^2 - u^2 = n \Rightarrow v \pm u$ – нетривиальный делитель n.
Обобщим: v, u: $v^2 - u^2 \equiv 0 \bmod n,\ v \pm u \not\equiv 0 \bmod n \Rightarrow gcd(n, v \pm u)$ – нетривиальный делитель n.

Суть Крайчика: взять $x_i = \lfloor \sqrt{n} \rfloor + i,\ y_i = x_i^2 - n$, найти подмножество y-ков, дающих в произведении точный квадрат, получается $x_{i_1}^2 \cdot \dots \cdot x_{i_k}^2 \equiv y_{i_1}^2 \cdot \dots \cdot y_{i_k}^2 \bmod n$, применяем Ферма.

Задача: дано множество $\{y_i\}$, найти подмножество, являющееся точным квадратом.
Решение: число является квадратом iff в его факторизации все степени простых чётны $\Rightarrow$ факторизуем y_i, получаем вектора z_i "чётности степеней простых", находим подмножество векторов z_i, в сумме над $\mathbb{F}_2$ дающее 0.
Реализация: Гаусс над разреженной матрицей.

Небольшая проблема – мы как раз учимся факторизовать, а тут нужно факторизовать y_i.
Чтобы разрешить эту неловкость, будем брать только b-гладкие y_i.

Def 10.1.1. *Число x называется b-гладким, если все простые в разложении x не больше b.*

Замечание 10.1.2. b-гладкое число x легко факторизовать за $\mathcal{O}(\frac{b}{\log b} + \log x)$ делений.

• Алгоритм `factorize(n)`

1. Фиксируем константу k ($b = \texttt{prime}[k]$).
2. Генерируем числа y_i, сразу факторизуем их за $\mathcal{O}(k + \log y_i)$.
3. Для всех y_i, оказавшихся b-гладкими, скармливаем Гауссу z_i, вектор над $\mathbb{F}_2$ длины k.
4. Каждый раз, когда Гаусс видит, что очередной вектор – линейная комбинация предыдущих, подставляем в тест Ферма $v = \prod x_{i_j}$ и $u = \left(\prod y_{i_j}\right)^{1/2}$.

• Время работы

Алгоритм зависнет для простого n и имеет шанс найти делитель для составного n.
При успешном завершении время работы $\mathcal{O}(k^3 + k^2 t + km)$, где t – число запусков теста Ферма, а m – общее число всех y_i.

Чтобы сделать алгоритм конечным и вероятностным, можно скармливать вектора Гауссу с вероятностью $\frac{1}{2}$ и останавливаться после первого же теста Ферма.
Тогда время $\mathcal{O}(k^3 + km)$, и мы верим в оценку на вероятность p: $0 < \varepsilon \leqslant p \leqslant 1$.

• Оптимизация

Можно для всех чисел $p_i = \texttt{prime}[i]$ заранее посчитать r_i: $(\pm r_i)^2 - n \equiv 0 \bmod p_i$.
Тогда множество таких j, что $p_i \mid y_j$ равно $\{\pm r_i + k \cdot p_i - \lfloor \sqrt{n} \rfloor \mid k \in \mathbb{Z}\}$. Факторизация:

```
1 for i for j while (p_i | y_j) do { y_j /= p_i; v_j[i]++; }
```

Научились факторизовать все y_j за $\sum\limits_{i=1..k} \frac{m}{i \log i} = \Theta(m \log \log k)$.

Предположим, что $\mathcal{O}(m \log \log k)$ единиц по m векторам распределены равномерно, тогда в выбранных $(k+1)$-ом векторе будет лишь $\mathcal{O}(k \log \log k)$ не нулей. Можно написать разреженного Гаусса, который хранит в строках лишь списки ненулевых элементов и операции над

строками-списками длин a и b делает за $\mathcal{O}(a+b)$. Можно воспользоваться алгоритмом Видеманна (Wiedemann), который решит систему за $\mathcal{O}(k \cdot z)$, где z – число не нулей в матрице.

Итого время $\mathcal{O}(k^3 + km)$ можно улучшить до $\mathcal{O}((k^2+m)\log\log k)$.
Чем меньше k, тем больше m. Нужно подобрать k: $k^2 = \Theta(m)$. Это можно сделать, например, последовательным удвоением (`for` $k \in \{1, 2, 4, 8, \dots\}$ `do` запускаем решение для $m = k^2$).

Теорема 10.1.3. Для числа n оптимальное $k = e^{\frac{1}{2}\sqrt{\log n \log\log n}}$

Следствие 10.1.4. Время Крайчика со всеми оптимизациями $T = e^{\sqrt{\log n \log\log n}}$, $\forall \varepsilon > 0 \;\; T = o(n^\varepsilon)$.

Замечание 10.1.5. Это первый, изученный нами, субэкспоненциальный алгоритм факторизации.
Раньше мы умели за $\mathcal{O}(n^{1/4})$. Длина ввода есть $l = \log n \Rightarrow$ мы умели только за $exp(\Theta(l))$.

10.2. Литература

[logic.pdmi.ras]. Слайды от Николенко про Крайчика и Видемана.
[wiki|Wiedemann]. Короткое внятное описание решения системы $Ax = 0$ над $\mathbb{F}_p$ за $\mathcal{O}(n \cdot z)$.
[Pomerance]. Доказательства, связанные с b-гладкими числами и алгоритмом Крайчика.
[Факторизация]. Много про факторизацию на русском. И про Крайчика тоже.

Лекция #11: Планарность

27 мая 2019

11.1. Основные определения и теоремы

Def 11.1.1. *Планарным (planar) называется граф, который может быть изображён на плоскости без пересечения рёбер.*

Def 11.1.2. *Плоским (plane) граф – изображение графа на плоскости без пересечений рёбер.*

Плоский граф разбивает плоскость на области, которые называются гранями.
Внешней называется грань, содержащая точку на бесконечности.

Теорема 11.1.3. У графа есть укладка на сфере iff граф планарен.
Доказательство. Плоскость гомеоморфна проколотой сфере. Осталось проткнуть сферу. ■

Следствие 11.1.4. Для ∀ грани планарного графа есть укладка, в которой именно она внешняя.
Доказательство. Уложим на сфере. Проткнём нужную грань, получилась плоскость. ■

Теорема 11.1.5. Эйлера. Для плоского графа $E + C + 1 = V + G$, где V – число вершин, E – число рёбер, G – число граней, C – число компонент связности.
Доказательство. Индукция: удалим ребро, или увеличилось число граней, или уменьшилось число компонент связности. В конце $C = V$, $G = 1$. ■

Следствие 11.1.6. Для планарных графов $E \leqslant 3V - 6$, $E = \mathcal{O}(V)$.
Доказательство. У каждой грани хотя бы 3 ребра $\Rightarrow E \geqslant \frac{3}{2}G \Rightarrow E + 2 \leqslant V + \frac{2}{3}E$. ■

Упражнение 11.1.7. Покажите, что для двудольных планарных $E \leqslant 2V - 4$.

Lm 11.1.8. Графы $K_{3,3}$ и K_5 не планарны.
Доказательство. В K_5 вершин 5, рёбер $10 > 3{\cdot}5 - 6$. В $K_{3,3}$ вершин 6, рёбер $9 > 2{\cdot}6 - 4$. ■

Def 11.1.9. *Стягивание графа. Процесс, в котором каждая операция – удаление вершины, ребра или стягивание двух вершин, соединённых ребром, в одну.*

Теорема 11.1.10. Вагнер'1937. Граф планарен iff он не стягивается в $K_{3,3}$ или K_5.

Теорема 11.1.11. Куратовский'1930. Граф планарен iff он не является подразбиением $K_{3,3}$, K_5.

Подразбиение – процесс обратный стягиванию, так что теоремы равносильны.
Короткое доказательство теоремы от Скопенкова [eng|arxiv] [rus].

Теорема 11.1.12. Фари'1948. ∀ планарный граф можно уложить прямыми отрезками.

Теорема 11.1.13. Schnyder'1989. ∀ планарный граф при $V \geqslant 3$ можно уложить с использованием только прямых отрезков на гриде размера $(V-2) \times (V-2)$. Укладку можно найти за $\mathcal{O}(V)$.

Теорема 11.1.14. Koebe–Andreev–Thurston. [wiki].
Можно нарисовать вершины непересекающимися кругами: ребро ⇔ круги касаются.

11.2. Алгоритмы проверки на планарность

11.2.1. Исторический экскурс

[tarjan'1974]. Первый линейный алгоритм дан Тарьяном и Хопкрофтом.

Более современные алгоритмы развивались в направлении уменьшения сложности реализации, сложности доказательства, константы времени работы. Среди них можно выделить работы

[boyer'1999]. Boyer, Myrvold: A simplified O(n) planar embedding.
[brandes'2010]. Ulrik Brandes: The Left-Right Planarity Test.

11.2.2. Алгоритм Демукрона

Простейший алгоритм за $\mathcal{O}(n^2)$ в худшем, $\mathcal{O}(n \log n)$ в среднем при аккуратной реализации.

1. Выделим любой цикл C, уложим. Топологически есть ровно один способ уложить цикл.

2. После удаления рёбер и вершин C оставшиеся рёбра графа делятся на компоненты связности. Два ребра связны, если у них есть общий конец, не являющийся вершиной C.

3. Построим граф противоречий – для каждых двух компонент можно ли их уложить по одну сторону цикла. Время работы $\sum_{ij}(k_i + k_j) = 2m \sum_i k_i \leqslant 2mE$.
k_i – число вершин, которыми i-я компоненты зацепляется с циклом, m – число компонент.

TODO Картинка

4. Раскрасим граф противоречий в два цвета за $\mathcal{O}(m^2)$.
Если не красится, исходный граф не планарен.

5. Все компоненты укладываем независимо. Если компонента цепляет C не более чем одной вершиной, укладываем независимо от C. Иначе пусть цепляет вершинами a, b, ищем и рисуем любой путь $P\colon a \to b$ (топологически это можно сделать единственным образом). Оставшиеся рёбра компоненты делятся на подкомпоненты относительно P. $C \cup P$ образуют два цикла C_1 и C_2, каждая подкомпонента лежит в одном из них. Переходим к (3), не забываем, что про подкомпоненты, цепляющие $C \setminus P$ заранее известно в C_1 или C_2 они должны лежать.

Время работы: $T(E) = \mathcal{O}(E) + \mathcal{O}(mE) + T(e_1) + \cdots + T(e_m) = \mathcal{O}(E^2) = \mathcal{O}(V^2)$, $e_1 + \cdots + e_m \leqslant E$.

11.3. Алгоритмы отрисовки графа прямыми отрезками

Пусть у нас уже есть укладка графа.

Можно триангулировать граф: в грань из k вершин добавляем или $k - 3$ диагонали, или вершину-центр и k рёбер в центр.

Lm 11.3.1. Если все грани связного планарного графа – треугольники, граф трёхсвязен.
Доказательство. Пусть есть разрез $\{a, b\}$, удалим a и b, получим две компоненты связности G_1, G_2. Уложим графы $G_1 + a + b + (a, b)$ и $G_2 + a + b + (a, b)$ так, чтобы ребро (a, b) лежало на внешней грани. Соединим укладки. Получим внешнюю грань из $\geqslant 4$ вершин. Противоречие. ■

Теорема 11.3.2. Tutte'1963. Spring Theorem. Для $\forall$ трёхсвязного планарного графа.
Зафиксируем любую грань внешней, уложим её в виде выпуклого многоугольника.
Тогда остальная часть графа однозначно укладывается на плоскость таким образом, что каждая грань – выпуклый многоугольник, а любая внутренняя вершина – центр масс соседей.

• **Алгоритм отрисовки графа прямыми отрезками**

0. Вход: список рёбер. Выход: координаты вершин.
1. Укладываем граф Демукроном, получаем грани.
2. Триангулируем граф: в каждую грань добавляем центр и рёбра в центр.
3. Выбираем $\forall$ грань-$\triangle$, называем внешней, укладываем как $\triangle$ (-1,-1), (1,-1), (0,2).
4. Записываем систему уравнений из 11.3.2.
5. Устанавливаем координаты всех остальных вершин $(0,0)$, решаем систему методом итераций.
6. Удаляем лишние рёбра, добавленные в (2).

Замечание 11.3.3. Если изначально граф уже трёхсвязен, можно пропустить (1), (2), (6). Внешней гранью тогда можно выбрать, например, кратчайший цикл.

11.4. Планарный сепаратор

[lipton'1977]. Почти сразу после успешной проверки на планарность и укладки графа Тарьян и Липтон показали, что в $\forall$ планарном графе за $\mathcal{O}(n)$ можно найти сепаратор размера $2\sqrt{2}\sqrt{n}$.

Def 11.4.1. *S – сепаратор графа $G = \langle V, E\rangle$, если*
$V = A \sqcup B \sqcup S$: $\max(|A|, |B|) \leqslant \frac{2}{3}n$, $n = |V|$, *между A и B нет рёбер.*

Lm 11.4.2. Если при удалении S все компоненты связности имеют размер $\leqslant \frac{2}{3}n$, S – сепаратор.

Доказательство. Жадно набираем множества A и B. Просматриваем компоненты от большей к меньшей, кидаем очередную компоненту в меньшее из A, B. ■

Lm 11.4.3. В планарном графе есть остов высоты $h \Rightarrow$ есть A: $|A| \leqslant 2h$, $A \cup \{root\}$ – сепаратор.

Доказательство. Триангулируем граф.
Каждое ребро e не из остова образует цикл, который делит множество вершин на A "внутри", B "снаружи". Выберем $e(u,v)$: $\max(|A|,|B|) \to \min = m$. Пусть $m > \frac{2}{3}n$. Пусть большая часть внутри цикла. Ребро e смежно двум граням-треугольникам. Возьмём тот, что внутри цикла, пусть его третья вершина x. Рассмотрим случаи – какие из рёбер (u,x), (v,x) лежат на цикле. В самом интересном "оба не лежат" покажем, что у одного из рёбер (u,x), (v,x) величина $\max(|A|,|B|)$ меньше. Противоречие. ■

Теорема 11.4.4. Тарьян-Липтон. В $\forall$ планарном графе G $\exists$ сепаратор S: $|S| \leqslant 2\sqrt{2}\sqrt{n}$.

Доказательство. Запустим bfs из любой вершины v, получим слои по расстоянию до v – $L_0, L_1, \ldots, L_d$. Найдём i: $|L_0 \cup \cdots \cup L_{i-1}| \leqslant \frac{n}{2}$, $|L_0 \cup \cdots \cup L_i| > \frac{n}{2}$.
Если бы $|L_i| \leqslant f(n)$, счастье уже наступило бы. Но нет $\Rightarrow$

Ищем $j_0 < i$: $|L_{j_0}| + 2(i - j_0 - 1) \leqslant 2\sqrt{k}$, где $k = |L_0| + \cdots + |L_{i-1}|$.
Пусть $\nexists j_0 \Rightarrow |L_{i-1}| > \sqrt{k}$, $|L_{i-2}| > \sqrt{k} - 2, \ldots \Rightarrow |L_0| + \cdots + |L_{i-1}| > k$. Противоречие.
Аналогично ищем $j_1 \geqslant i$: $|L_{j_1}| + 2(j_1 - i) \leqslant 2\sqrt{n-k}$.
Пусть $\nexists j_1 \Rightarrow |L_i| > \sqrt{n-k}$, $|L_{i+1}| > \sqrt{n-k} - 2, \ldots \Rightarrow |L_i| + \cdots + |L_n| > n-k$. Противоречие.
Максимум величины $|L_{j_0}| + |L_{j_1}| + 2(j_1 - j_0 + 1)$ достигается при $k = \frac{n}{2}$ и равен $2\sqrt{2}\sqrt{n}$.

Обозначим $L_a \cup L_{a+1} \cup \cdots \cup L_b$, как $L[a,b]$.
Множество $S_1 = L_{j_0} \sqcup L_{j_1}$ делит граф на 3 части: $A = L[0, j_0 - 1]$, $B = L[j_0 + 1, j_1 - 1]$, $C = L[j_1 + 1, n]$.
$|A|, |C| \leqslant \frac{n}{2} \Rightarrow$ если $|B| \leqslant \frac{2n}{3}$, вернём сепаратор S_1.
Иначе рассмотрим граф $G' = L[j_0, j_1 - 1]/_{L_{j_0}}$ (стянули L_{j_0} в одну вершину).

В G' остовное дерево bfs имеет высоту $j_1 - j_0 \overset{11.4.3}{\Rightarrow} \exists$ сепаратор $X \cup \{root\} = X \cup L_{j_0}$: $|X| \leqslant 2(j_1 - j_0)$.
Итого $|(L_{j_0} \sqcup L_{j_1}) \cup X| \leqslant 2\sqrt{2}\sqrt{n}$ и является планарным сепаратором. ■

Замечание 11.4.5. В доказательстве теоремы планарность мы пользуемся только в лемме 11.4.3.

Замечание 11.4.6. [holzer'2009]. В журнале по экспериментальным алгоритмам говорят, что если оставить только последнюю часть алгоритм "*взять циклы, образованные не древесными рёбрами триангуляции*", то лучший из рассмотренных циклов на большинстве графов даст оценки лучше чем в теореме.

11.4.1. Решение NP-трудных задач на планарных графах

Lm 11.4.7. В ∀ планарном графе есть множество вершин S размера $\Theta(\sqrt{n})$:
$V = A \sqcup B \sqcup S$, $\max(|A|, |B|) \leqslant \frac{n}{2}$, между A и B нет рёбер.

Доказательство. У нас есть две корзины $A, B = \varnothing$ и множество $X = V$. В каждый момент времени $\max(|A|, |B|) \leqslant \frac{n}{2}$. Пока $\min(|A|, |B|) + |X| > \frac{n}{2}$ делим X его сепаратором на Y и Z, к меньшему из A и B добавляем меньшее из Y и Z, большее из Y и Z кладём в X.

Оценим суммарный размер сепараторов: $2\sqrt{2}(\sqrt{n} + \sqrt{\frac{2}{3}n} + \sqrt{\frac{4}{9}n} + \dots) = \frac{2\sqrt{2}}{1-\sqrt{2/3}}\sqrt{n}$. ■

• Поиск max IS в планарном графе за $2^{\Theta(n)}$.

Разделяй и властвуй. Выделим по лемме $(\frac{n}{2}, \frac{n}{2})$-сепаратор S.
Переберём $2^{|S|}$ вариантов, какие вершины сепаратора лежат в независимом множестве.
Для каждого из $2^{|S|}$ вариантов сделаем два рекурсивных вызова от половин.
Итого $T(n) \leqslant 2 \cdot 2^{C\sqrt{n}} \cdot T(n/2) \leqslant 2 \cdot 2^{C\sqrt{n}} \cdot 2 \cdot 2^{C\sqrt{n/2}} \cdot 2 \cdot 2^{C\sqrt{n/4}} \cdot \ldots = 2^{\Theta(\sqrt{n})}$.

• Раскраска в k цветов планарного графа за $2^{\Theta(n \log k)}$.

Аналогично: переберём все $k^{|S|}$ раскрасок вершин сепаратора.

Замечание 11.4.8. Научившись искать ещё меньший сепаратор, мы ускорим решение многих NP-трудных задач. К сожалению, для планарных графов оценка $\Theta(\sqrt{n})$ достигается: грид $\sqrt{n} \times \sqrt{n}$.

11.5. Системы уравнений

11.5.1. k-диагональная матрица

Если запустить обычного Гаусса, учитывающего, что строки – отрезки длины не более $2k$, автоматически получится решение за $\mathcal{O}(k^2 n)$. В частности метод для трёхдиагональной матрицы делает то же и работает за $\mathcal{O}(n)$.

Аналогично для верхнетреугольной матрицы + k диагоналей Гаусс будет работать за $\mathcal{O}(kn^2)$.

11.5.2. Правило Кирхгофа

Рассмотрим задачу нахождение сопротивления между вершинами a и b в неорграфе, где у каждого ребра e есть сопротивление R_e.

Для всех вершин v обозначим потенциал вершины $u[v]$. Пусть $u[a] = 0$, $u[b] = 1$. Сила тока $I_{e\colon a \to b} = \frac{u[b]-u[a]}{R_e}$. Правило Кирхгофа гласит "*в каждую вершину втекает тока ровно столько, сколько вытекает*", т.е. $\forall v \neq a, b \sum\limits_{e\colon v \to x} I_e = \sum\limits_{e\colon v \to x} \frac{u[v]-u[x]}{R_e} = 0 \Leftrightarrow \left(\sum_e \frac{1}{R_e}\right) u[v] = \sum_e \frac{1}{R_e} u[x]$.

Получили систему линейных уравнений над $u[v]$. Её можно решить Гауссом за $\mathcal{O}(V^3)$.
Если схема задаётся планарном графом, то есть сепаратор размера $\mathcal{O}(V^{1/2}) \Rightarrow$
систему можно решить за $\mathcal{O}(V^{3/2})$ используя nested dissection. Подробно в [tarjan'1986].

11.6. Выделение граней плоского графа

• **Задача**

Даны координаты вершин плоского графа, уложенного прямыми отрезками.
Выделить грани графа.

• **Решение за $\mathcal{O}(sort + V)$**

Для каждого ребра $e(a, b)$ создадим две ориентированные копии $e_1 \colon a \to b,\ e_2 \colon b \to a$.
Проставим $rev[e_1] = e_2, rev[e_2] = e_1$. Добавим рёбра в списки смежности $g[a].add(e_1)$, $g[b].add(e_2)$.
Для каждой вершины v отсортируем $g[v]$ по углу. Сохраним позиции рёбер в списках смежности:

```
1 for v=1..V:
2   for i=0..g[v].size-1:
3     index[g[v][i]] = i
4 def next(e): # e: a -> b
5   retrun g[a[e]][(index[e] + 1) % g[a[e]].size]
```

Тогда для каждого ориентированного ребра e следующим в грани будет $f_e = next(rev[e])$.
Граф $e \to f_e$ – ровно набор ориентированных циклов, осталось их выделить.

Замечание 11.6.1. Все грани кроме внешней будут ориентированы по часовой стрелке.

11.7. Литература

[Luca Vismara | graphbook]. Подробно про отрисовку графов.
[thomassen'2004]. Доказательство пружинной теоремы Татта, её физическое понимание.
[tarjan'1986]. Гаусс для планарных графов за $n^{3/2}$ и не только.
[tarjan'1977]. Существование и построение за $\mathcal{O}(n)$ планарного сепаратора.
[skopenkov]. Доказательство теоремы Куратовского.
[boyer'1999]. Короткий (2.5 страниц в 2 столбца) алгоритм проверки на планарность за $\mathcal{O}(n)$.