

Математические основы алгоритмов

Тискин Александр Владимирович

Факультет математики и компьютерных наук СПбГУ

- 1 Дискретное преобразование Фурье и его применения
- 2 Параллельные алгоритмы
- 3 Оптимизационные задачи и приближенные алгоритмы

Дискретное преобразование Фурье и его применения

Дискретное преобразование Фурье

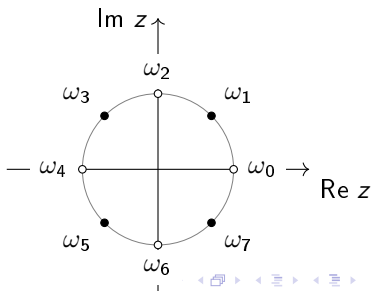
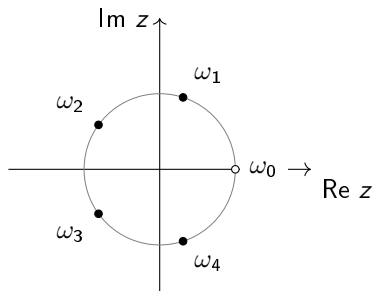
R — коммутативное кольцо без делителей нуля $n \geq 2$

$\omega \in R$ — **первообразный корень степени n из единицы** ($\sqrt[n]{1}$) если $\omega, \omega^2, \dots, \omega^{n-1} \neq 1, \omega^n = 1$

В частности, $R = \mathbb{C}$

$\omega = \omega_k = e^{\frac{2\pi i}{n}k}$ — первообразный $\sqrt[n]{1}$, если $\gcd(k, n) = 1$

$\langle \omega \rangle = \{1, \omega, \omega^2, \dots, \omega^{n-1}\}$: циклическая группа корней из единицы



Дискретное преобразование Фурье и его применения

Дискретное преобразование Фурье

R — коммутативное кольцо без делителей нуля $n \geq 2$

Пусть $n = 1 + \dots + 1$ (n раз) обратим в R : $\exists n^{-1} \in R, n^{-1} \cdot n$

Пусть $\exists \omega \in R$ — первообразный $\sqrt[n]{1}$


$$a = [a_0, \dots, a_{n-1}]^T \in R^n \quad b = [b_0, \dots, b_{n-1}]^T \in R^n$$

Дискретное преобразование Фурье степени n (DFT_n): $F_{n,\omega} \cdot a = b$

$$F_{n,\omega} = [\omega^{ij}]_{i,j=0}^{n-1} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \dots & \omega \end{bmatrix}$$

$$\sum_j \omega^{ij} a_j = b_i \quad 0 \leq i, j < n$$

Трудоёмкость: наивный алгоритм — время $O(n^2)$

Приложения: обработка сигналов, изображений, сжатие данных, ... 

Дискретное преобразование Фурье и его применения

Дискретное преобразование Фурье

Обратное DFT_n: $\frac{1}{n}F_{n,\omega^{-1}} \cdot b = \frac{1}{n}F_{n,\omega^{-1}} \cdot F_{n,\omega} \cdot a = a$

$F_{n,\omega^{-1}} \cdot F_{n,\omega} = n \cdot I$ (упражнение)

$a(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$

$C = \{1, \omega, \omega^2, \dots, \omega^{n-1}\}$

$F_{n,\omega} \cdot a$: **вычисление значений** полинома $a(x)$ на C

$\frac{1}{n}F_{n,\omega^{-1}} \cdot a$: **интерполяция** полинома $a(x)$ по значениям на C

$\frac{1}{\sqrt{n}}F_{n,\omega^{-1}} \cdot a$: разложение a по **базису Фурье** (строкам $\frac{1}{\sqrt{n}}F_{n,\omega}$)

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

Быстрое преобразование Фурье (FFT)

Идея: использовать разложение n на множители и структуру множества корней из единицы для ускорения вычислений

Пусть $n = n' n''$; $1 < n', n'' < n$. В алгоритме FFT

- вектор a записывается в виде $n' \times n''$ -матрицы по строкам
- вектор b записывается в виде $n'' \times n'$ -матрицы по строкам
- DFT_n выражается через $\text{DFT}_{n'}$, $\text{DFT}_{n''}$ над столбцами матриц
- $\text{DFT}_{n'}$, $\text{DFT}_{n''}$ вычисляются рекурсивно

Классические варианты:

- $n = \frac{n}{2} \cdot 2$ — FFT с прореживанием по времени (FFT-DIT)
- $n = 2 \cdot \frac{n}{2}$ — FFT с прореживанием по частоте (FFT-DIF)
- общий случай (например, $n = (n^{1/2})^2$) — шестиэтапное FFT

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

Шестиэтапное FFT

Пусть $n = n' n''$ $1 < n', n'' < n$

$$A = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n''-1} \\ a_{n''} & a_{n''+1} & \cdots & a_{2n''-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-n''} & a_{n-n''+1} & \cdots & a_{n-1} \end{bmatrix} \quad B = \begin{bmatrix} b_0 & b_1 & \cdots & b_{n'-1} \\ b_{n'} & b_{n'+1} & \cdots & b_{2n'-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-n'} & b_{n-n'+1} & \cdots & b_{n-1} \end{bmatrix}$$

$$A_{u,v} = a_{n''u+v} \quad B_{s,t} = b_{n's+t} \quad 0 \leq u, t < n' \quad 0 \leq s, v < n''$$

$$B_{s,t} = \sum_{u,v} \omega^{(n's+t)(n''u+v)} A_{u,v} = \sum_{u,v} \omega^{n'sv+tv+n''tu} A_{u,v} = \sum_v ((\omega^{n'})^{sv} \cdot \omega^{tv} \cdot \sum_u (\omega^{n''})^{tu} A_{u,v})$$

Таким образом, $B = F_{n'', \omega^{n'}} \cdot (G_{n', n'', \omega} \circ (F_{n', \omega^{n''}} \cdot A))^T$

$G_{n', n'', \omega} = [\omega^{tv}]_{t,v}$: $n' \times n''$ -матрица **поворотных множителей**

Оператор \circ — **умножение Адамара** (поэлементное умножение матриц)

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

$$B = F_{n'', \omega^{n'}} \cdot (G_{n', n'', \omega} \circ (F_{n', \omega^{n''}} \cdot A))^T$$

$F_{k, \zeta} \cdot X$: l независимых DFT_k над столбцами $k \times l$ -матрицы X

Пусть $n = 2^r = n' \cdot n''$

Имеем шестиэтапную **схему FFT** для DFT_n :

- вектор a записывается в матрицу по строкам
- n'' независимых $\text{DFT}_{n'}$ над столбцами, вычисляются рекурсивно
- транспозиция; применение поворотных множителей (2 этапа)
- n' независимых $\text{DFT}_{n''}$ над столбцами, вычисляются рекурсивно
- вектор b считывается из матрицы по строкам

$$\text{База рекурсии: } F_{-1,2} \cdot a = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} a_0 + a_1 \\ a_0 - a_1 \end{bmatrix} = b$$

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

Трудоёмкость FFT-DIT ($n' = \frac{n}{2}$, $n'' = 2$)

$$T(n) = O(n) + 2T(n/2) = O\left(n \cdot \left(1 + \frac{2}{2} + \left(\frac{2}{2}\right)^2 + \dots + \left(\frac{2}{2}\right)^{\log_2 n}\right)\right) = O(n \log n)$$

Трудоёмкость FFT-DIF ($n' = 2$, $n'' = \frac{n}{2}$) — аналогично

Трудоёмкость симметричного FFT ($n = 2^{2^s}$, $n' = n'' = n^{1/2} = 2^{2^{s-1}}$)

$$T(n) = O(n) + 2 \cdot n^{1/2} \cdot T(n^{1/2}) = O(1 \cdot n \cdot 1 + 2 \cdot n^{1/2} \cdot n^{1/2} + 4 \cdot n^{3/4} \cdot n^{1/4} + \dots + \log n \cdot n \cdot 1) = O(n + 2n + 4n + \dots + \log n \cdot n) = O(n \log n)$$

Дискретное преобразование Фурье и его применения

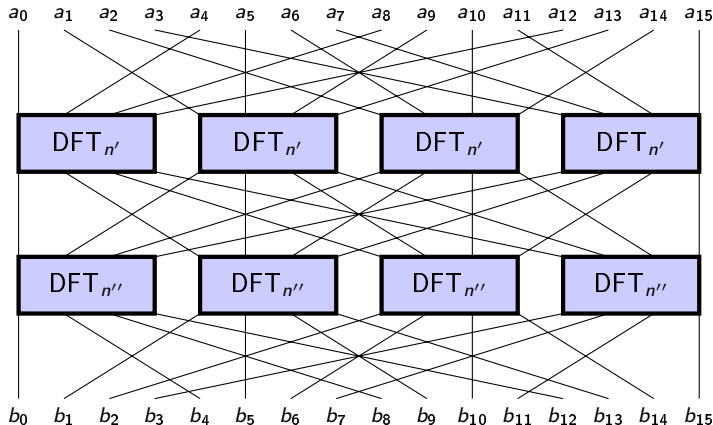
Быстрое преобразование Фурье

Схема FFT и граф-бабочка (шестиэтапное FFT)

DFT_n

$n = 16$

$n' = n'' = 4$



Поворотные множители не указаны

Дискретное преобразование Фурье и его применения

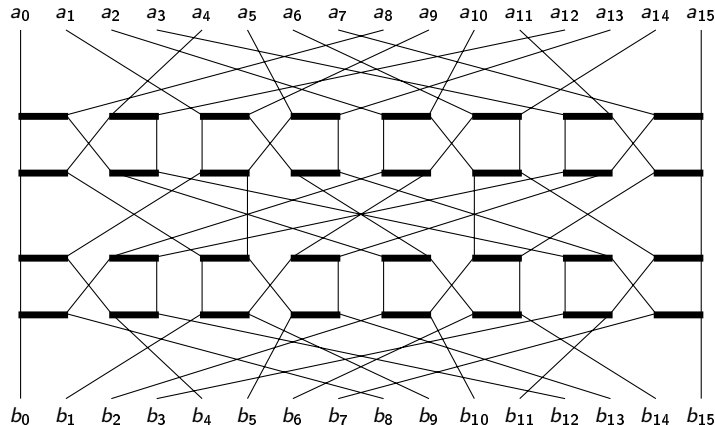
Быстрое преобразование Фурье

Схема FFT и граф-бабочка (шестиэтапное FFT)

DFT_n

$n = 16$

$n' = n'' = 4$

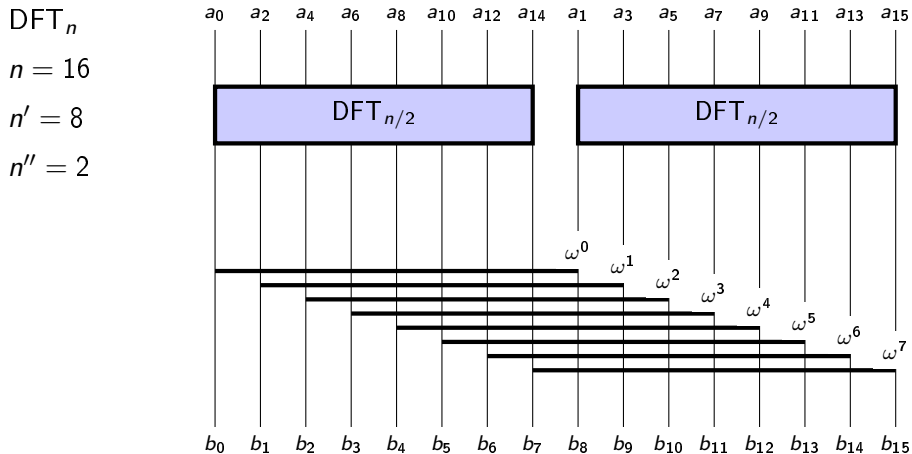


Поворотные множители не указаны

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

Схема FFT и граф-бабочка (FFT-DIT)

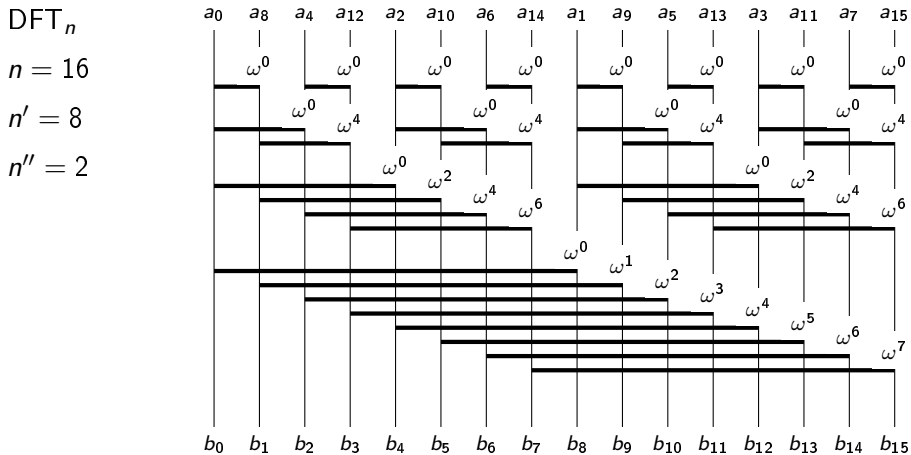


Вход: четно-нечетная перестановка a

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье

Схема FFT и граф-бабочка (FFT-DIT)



Вход: бит-реверсивная перестановка a

Дискретное преобразование Фурье и его применения

Умножение полиномов

Задача **умножения полиномов**

$$a, b, c \in \mathbb{C}[x]$$

$$a(x) = \sum_{0 \leq i < n} a_i x^i \quad b(x) = \sum_{0 \leq i < n} b_i x^i \quad c(x) = \sum_{0 \leq i < 2n-1} c_i x^i$$

Предполагаем $a_i = b_j = 0$ при $i, j \geq n$

$$a(x) \cdot b(x) = c(x) = \sum_{0 \leq k < 2n-1} c_k x^k = \sum_{0 \leq k < 2n-1} \left(\sum_{0 \leq i \leq k} a_i b_{k-i} \right) x^k$$

Трудоемкость: наивный алгоритм — время $O(n^2)$

Алгоритм Карацубы

$$a(x) = \sum_i a_i x^i \quad b(x) = \sum_j b_j x^j \quad 0 \leq i, j < n$$

Пусть $n = 2m$

$$a(x) = \sum_i a_i x^i + \left(\sum_i a_{m+i} x^i\right) x^m = a'(x) + a''(x) x^m \quad 0 \leq i < m$$

Аналогично, $b(x) = b'(x) + b''(x) x^m$

$$c(x) = (a' + a'' x^m)(b' + b'' x^m) = a'b' + (a'b'' + a''b')x^m + a''b''x^n = a'b' + ((a' + a'')(b' + b'') - a'b' - a''b'')x^m + a''b''x^n$$

Вместо 4 умножений полиномов с m членами получилось 3

Дискретное преобразование Фурье и его применения

Умножение полиномов

Пусть n — степень 2, иначе возьмем минимальную степень 2 большую n и дополним a , b нулевыми старшими членами до n членов

- $a'b'$, $a''b''$, $(a' + a'')(b' + b'')$ вычисляются рекурсивно
- вычисляется $(a' + a'')(b' + b'') - a'b' - a''b''$

База рекурсии: $n = 1$ $ab = a_0 b_0$

Трудоемкость: $T(n) = O(n) + 3T(n/2) =$

$$O\left(n \cdot \left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \dots + \left(\frac{3}{2}\right)^{\log_2 n}\right)\right) = O\left(n \cdot \frac{1}{n} \cdot 3^{\log_2 n}\right) = O(n^{\log_2 3}) = O(n^{1.59})$$

Интерполяция полиномов

$$x_0, x_1, \dots, x_{n-1} \in \mathbb{C}$$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a(x_0) \\ a(x_1) \\ a(x_2) \\ \vdots \\ a(x_{n-1}) \end{bmatrix}$$

$$\sum_j x_i^j a_j = a(x_i) \quad 0 \leq i, j < n$$

Если все x_0, x_1, \dots, x_{n-1} различны, то матрица невырождена

Следовательно, n коэффициентов полинома однозначно определяются его $n - 1$ значениями

Быстрое умножение полиномов

- возьмем $N \geq 2n - 1$ различных $x_0, x_1, \dots, x_{N-1} \in \mathbb{C}$
- вычислим a, b на всех x_i , получим $a(x_i), b(x_i), i = 0, 1, \dots, N - 1$
- перемножим попарно $c(x_i) = a(x_i) \cdot b(x_i)$
- проинтерполируем $c(x)$ по $c(x_i)$

Идея: взять в качестве x_i корни из единицы и применить FFT

Пусть N — степень 2, т.ч. $2n - 1 \leq N < 4n - 1$

$$x_i = \omega^i = e^{\frac{2\pi i}{N}i} \quad 0 \leq i < N$$

Вычисление $a(x_i), b(x_i)$: DFT степени N

Интерполяция $c(x)$: обратное DFT степени N

Трудоемкость: $O(n \log n)$

Дискретное преобразование Фурье и его применения

Свертки

$$a = [a_0, a_1, \dots, a_{n-1}]^T \quad b = [b_0, b_1, \dots, b_{n-1}]^T$$

$$a' = [a_0, a_1, \dots, a_{n-1}, 0, \dots, 0]^T \quad b' = [b_0, b_1, \dots, b_{n-1}, 0, \dots, 0]^T$$

(дополнение нулями до длины $2n$)

(**Линейная**) **свертка** $c = a * b$

$$c_i = \sum_{0 \leq j \leq i} a'_j b'_{i-j} = \begin{cases} \sum_{0 \leq j \leq i} a_j b_{i-j} & 0 \leq i \leq n \\ \sum_{i-n \leq j < n} a_j b_{i-j} & n \leq i < 2n \end{cases}$$

Быстрое вычисление линейной свертки (= умножение полиномов)

ω — первообразный $\sqrt[n]{1}$

$$c = \frac{1}{n} F_{2n, \omega^{-1}} ((F_{2n, \omega} a') \circ (F_{2n, \omega} b'))$$

Дискретное преобразование Фурье и его применения

Свертки

$$a = [a_0, a_1, \dots, a_{n-1}]^T \quad b = [b_0, b_1, \dots, b_{n-1}]^T$$

Циклическая и косоциклическая свертка $c^\pm = a \circledast_\pm b$:

$$c_i^\pm = c_i \pm c_{i+n} = \sum_{0 \leq j \leq i} a_j b_{i-j} \pm \sum_{i < j < n} a_j b_{n+i-j} \quad 0 \leq i < n$$

Быстрое вычисление циклической свертки

$$c^+ = \frac{1}{n} F_{n,\omega^{-1}} ((F_{n,\omega} a) \circ (F_{n,\omega} b))$$

Быстрое вычисление косоциклической свертки

$$\psi \text{ — первообразный } \sqrt[n]{1} \quad \psi^2 = \omega$$

$$w_{n,\psi} = [1, \psi, \dots, \psi^{n-1}]^T$$

$$c^- = w_{n,\psi^{-1}} \circ \left(\frac{1}{n} F_{n,\omega^{-1}} ((F_{n,\omega} (w_{n,\psi} \circ a)) \circ (F_{n,\omega} (w_{n,\psi} \circ b))) \right)$$

Доказательство: упражнение

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Пусть $R = \mathbb{Z}_N$

$\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N^+ : \gcd(x, N) = 1\}$ — мультипликативная группа mod N

$$|\mathbb{Z}_N^*| = \phi(N)$$

ξ — первообразный $\phi(N)\sqrt[n]{1}$, если $\mathbb{Z}_N^* = \langle \xi \rangle$. Существует только при $N = 2, 4, p^k, 2p^k$, где $p > 2$ — простое.

Тогда для любого $n \mid \phi(N)$, $\omega = \xi^{\frac{\phi(N)}{n}}$ — первообразный $\sqrt[n]{1}$

С другой стороны, по заданным степеням двойки n , ω несложно найти N такое, что ω — первообразный $\sqrt[n]{1}$ в \mathbb{Z}_N

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Теорема Пусть n, ω — положительные степени 2, $N = \omega^{n/2} + 1$. Тогда

- n обратимо в \mathbb{Z}_N
- ω — первообразный $\sqrt[n]{1}$ в \mathbb{Z}_N

Доказательство

Имеем $n \mid \omega^{n/2}$, $n^{-1} \equiv -\frac{\omega^{n/2}}{n} \pmod{N}$

Действительно, $(-\frac{\omega^{n/2}}{n}) \cdot n = -\omega^{n/2} \equiv 1 \pmod{\omega^{n/2} + 1 = N}$

ω — это $\sqrt[n]{1}$ в \mathbb{Z}_N

Действительно, $\omega^n = (\omega^{n/2})^2 \equiv (-1)^2 = 1 \pmod{\omega^{n/2} + 1 = N}$

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Доказательство (продолжение)

Предположим, что $\omega^m \equiv 1 \pmod{N}$, $0 < m < n$. Рассмотрим наименьшее такое m .

Имеем $\omega^k \equiv 1 \pmod{N}$, т. и т.т. когда $m \mid k$. Следовательно, $m \mid n$.

Поскольку n — степень двойки, из $m < n$, $m \mid n$ следует $m \mid n/2$

Имеем $\omega^{n/2} \equiv 1 \pmod{N}$. Однако $\omega^{n/2} \equiv -1 \pmod{\omega^{n/2} + 1 = N}$ — противоречие

Следовательно, ω — первообразный $\sqrt[n]{1}$ в \mathbb{Z}_N



Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Альтернативное доказательство

Пусть $n = 2^k$, $N = \omega^{2^{k-1}} + 1$.

Предположим, что $\omega^m \equiv 1 \pmod{N}$ $0 < m < n$

Рассмотрим $\sum_{0 \leq i < n} \omega^{im}$

$$\begin{aligned} \sum_i \omega^{im} &= \frac{\omega^{2^k m} - 1}{\omega^m - 1} = (\omega^{2^{k-1} m} + 1) \frac{\omega^{2^{k-1} m} - 1}{\omega^m - 1} = \\ &= (\omega^{2^{k-1} m} + 1)(\omega^{2^{k-2} m} + 1) \frac{\omega^{2^{k-2} m} - 1}{\omega^m - 1} = \dots = \\ &= (\omega^{2^{k-1} m} + 1)(\omega^{2^{k-2} m} + 1) \dots (\omega^m + 1) \frac{\omega^m - 1}{\omega^m - 1} = \\ &= (\omega^{2^{k-1} m} + 1)(\omega^{2^{k-2} m} + 1) \dots (\omega^m + 1) \end{aligned}$$

Пусть $m = 2^t s$ s нечетно $0 \leq t < k$ (поскольку $m < n$)

$$\omega^{2^{k-1} s} + 1 = \omega^{2^{k-t-1} 2^t s} + 1 = \omega^{2^{k-t-1} m} + 1 \mid \sum_i \omega^{im}$$

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Альтернативное доказательство (окончание)

Утверждение: для нечетного s , $x \in \mathbb{Z}$, выполняется $x + 1 \mid x^s + 1$

Действительно, пусть $x^s + 1 = (x + 1) \cdot q + r$ в $\mathbb{Z}[x]$

При $x = -1$ имеем $0 = 0 \cdot q + r$, следовательно $r = 0$

Согласно утверждению, $N = \omega^{2^{k-1}} + 1 \mid \omega^{2^{k-1}s} + 1 \mid \sum_i \omega^{im} \quad 0 \leq i < n$

Таким образом, $\sum_i \omega^{im} \equiv 0 \pmod{N}$

Но в то же время $\sum_i \omega^{im} \equiv \sum_i 1^i = n \not\equiv 0 \pmod{N}$ — противоречие!

Следовательно, ω — первообразный $\sqrt[n]{1}$ в \mathbb{Z}_N



Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Пусть $n, \omega = 2^r$ — положительные степени 2

$$N = \omega^{n/2} + 1 = 2^{\frac{nr}{2}} + 1$$

Для записи элемента \mathbb{Z}_N достаточно $\lceil \log N \rceil = \frac{nr}{2} + 1$ бит

Приведение $a \in \mathbb{N}$ в \mathbb{Z}_N : запишем a по основанию $\omega^{n/2}$

$$a = \sum_i a_i \omega^{ni/2} \equiv \sum_i a_i (-1)^i \pmod{N} \quad 0 \leq a_i < \omega^{n/2}$$

Если в записи a всего $O(nr)$ бит, трудоемкость $O(nr)$ битовых операций

$$\omega^{-1} \equiv \omega^{n-1} \quad \omega^{n-1} \cdot \omega = \omega^n \equiv 1 \pmod{N}$$

Дискретное преобразование Фурье и его применения

Быстрое преобразование Фурье в \mathbb{Z}_N

Операции FFT для прямого и обратного DFT_n в \mathbb{Z}_N

- сумма двух значений: в результате $(\frac{nr}{2} + 1) + 1 = \frac{nr}{2} + 2$ бит
- умножение на поворотный множитель $\omega^q = 2^{qr}$, $0 \leq q < n$: сдвиг на $qr < nr$ бит: в результате $\leq (\frac{nr}{2} + 1) + nr - 1 = \frac{3nr}{2}$ бит
- приведение значений с записью из $\frac{3nr}{2} = O(nr)$ бит

Трудоёмкость каждой операции над значениями: $O(nr)$ битовых операций

Трудоёмкость DFT_n в \mathbb{Z}_N :

- вход/выход $n \cdot O(nr) = O(n^2 r)$ бит
- $O(n \log n)$ сложений и умножений на поворотные множители $= O(n^2 r \log n)$ битовых операций

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Задача **умножения многозначных чисел**

a , b , c в записи по основанию 2

$$a = [a_0, a_1, \dots, a_{n-1}] \quad b = [b_0, b_1, \dots, b_{n-1}] \quad c = [c_0, c_1, \dots, c_{2n-1}]$$

$$a_i, b_j, c_k \in \{0, 1\}$$

Определим полиномы $a, b, c \in \mathbb{Z}[x]$

$$a(x) = \sum_{0 \leq i < n} a_i x^i \quad b(x) = \sum_{0 \leq i < n} b_i x^i \quad c(x) = \sum_{0 \leq i < 2n} c_i x^i$$

$$a(2) \cdot b(2) = (ab)(2) = c(2)$$

Трудоемкость: наивный алгоритм — $O(n^2)$ битовых операций

Замечание: Вообще говоря, $a(x) \cdot b(x) = (ab)(x) \neq c(x)$: в отличие от умножения полиномов, при умножении многозначных чисел происходит **перенос** из младших разрядов в старшие. В частности, в $(ab)(x)$ всего $2n - 1$ член, а в $c(x)$ — уже $2n$.

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Алгоритм Карацубы — аналогично одноименному для полиномов

$$a(2) = \sum_i a_i 2^i \quad b(2) = \sum_j b_j 2^j \quad 0 \leq i, j < n$$

Пусть $n = 2m$

$$a(2) = \sum_i a_i 2^i + (\sum_i a_{m+i} 2^i) 2^m = a'(2) + a''(2) 2^m \quad 0 \leq i < m$$

Аналогично, $b(x) = b'(x) + b''(x) 2^m$

$$c(2) = (a' + a'' 2^m)(b' + b'' 2^m) = a'b' + (a'b'' + a''b') 2^m + a''b'' 2^n = a'b' + ((a' + a'')(b' + b'') - a'b' - a''b'') 2^m + a''b'' 2^n$$

Вместо 4 умножений чисел с m разрядами получилось 3

В числах a' , b' , a'' , b'' по m бит

В числах $a' + a''$, $b' + b''$ по $m + 1$ бит

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Пусть n — степень 2, иначе возьмем минимальную степень 2 большую n и дополним a , b нулевыми старшими разрядами до n разрядов

- $a'b'$, $a''b''$, $(a' + a'')(b' + b'')$ вычисляются рекурсивно
- вычисляется $(a' + a'')(b' + b'') - a'b' - a''b''$

База рекурсии: $n = 1$ $ab = a_0b_0$

Трудоемкость $T(n) = O(n^{\log_2 3}) = O(n^{1.59})$ битовых операций

Перенос из младших разрядов в старшие:

- несущественен для алгоритма Карацубы (скорректировать анализ с учетом лишнего бита в $a' + a''$, $b' + b''$ — упражнение)
- существенен для метода FFT: попарные умножения придется выполнять рекурсивно

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Алгоритм Шенхаге–Штрассена

Идея: разбить a , b на разряды правильно выбранной длины l и рассматривать каждый разряд как коэффициент полинома, всего n/l коэффициентов в каждом полиноме. Использовать FFT для умножения полиномов. Парное умножение значений полиномов выполнять рекурсивно. Затем учесть переносы.

$$0 \leq a, b < 2^n$$

Пусть также $0 \leq c = ab < 2^n$ (иначе возьмем большее n и дополним a , b нулями). Достаточно вычислить c по модулю $2^n + 1$.

Обозначим $M(n)$ общую трудоемкость алгоритма в битовых операциях

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Пусть $n = 2^k = lm$ $l = 2^{\lceil k/2 \rceil}$ $m = 2^{\lfloor k/2 \rfloor}$ $m \mid l$

A, B, C в записи по основанию 2^l

$$A = [A_0, \dots, A_{m-1}]^T \quad B = [B_0, \dots, B_{m-1}]^T \quad C = [C_0, \dots, C_{2m-1}]^T$$

$$0 \leq A_i, B_i, C_i < 2^l$$

Определим полиномы $A, B, C \in \mathbb{Z}[x]$

$$A(x) = \sum_{0 \leq i < m} A_i x^i \quad B(x) = \sum_{0 \leq i < m} B_i x^i \quad C(x) = \sum_{0 \leq i < m} C_i x^i$$

$$a(2) = A(2^l) \quad b(2) = B(2^l) \quad c(2) = C(2^l)$$

$$A(2^l) \cdot B(2^l) = (AB)(2^l) = C(2^l)$$

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Пусть $D(x) = A(x)B(x)$ $D = A * B = [D_0, \dots, D_{2m-2}, D_{2m-1} = 0]^T$

$C(2^l) = D(2^l)$, то есть C — результат выполнения переносов в D

$$D_i = \begin{cases} \sum_{0 \leq j \leq i} A_j B_{i-j} & 0 \leq i < m \\ \sum_{i-m \leq j < m} A_j B_{i-j} & m \leq i < 2m \end{cases}$$

$$D(2^l) = \sum_{0 \leq i < 2m} D_i 2^{li} = \sum_{0 \leq i < m} D_i 2^{li} + \sum_{m \leq i < 2m} D_i 2^{li} = \\ \sum_{0 \leq i < m} D_i 2^{li} + \sum_{0 \leq i < m} D_{m+i} 2^{l(m+i)} \equiv$$

(используем $2^{lm} = 2^n \equiv -1 \pmod{2^n + 1}$)

$$\sum_{0 \leq i < m} (D_i - D_{m+i}) 2^{li} \equiv \sum_{0 \leq i < m} E_i 2^{li} \pmod{2^n + 1}$$

Пусть $E = A \circledast B = [D_0 - D_m, \dots, D_{m-1} - D_{2m-1}]^T$

Имеем $0 \leq D_i \leq (i+1)2^{2l}$ $0 \leq D_{m+i} \leq (m-i-1)2^{2l}$

Следовательно, $-(m-i-1)2^{2l} \leq E_i = D_i - D_{m+i} \leq (i+1)2^{2l}$

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Диапазон E_i равен $(i+1)2^{2l} - (-(m-i-1)2^{2l}) = m2^{2l}$,
следовательно, E_i можно вычислять вместо модуля $2^n + 1$ по
меньшему модулю $m(2^{2l} + 1) > m2^{2l}$

Пусть $E_i \equiv X \pmod{m(2^{2l} + 1)}$, $0 \leq X < m(2^{2l} + 1)$, тогда

$$E_i = \begin{cases} X - m(2^{2l} + 1) & \text{если } X > (i+1)2^{2l} \\ X & \text{иначе} \end{cases}$$

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Для вычисления $E_i \pmod{m(2^{2^l} + 1)}$, вычислим отдельно в \mathbb{Z}_m и $\mathbb{Z}_{2^{2^l}+1}$ (описано позднее), затем восстановим в $\mathbb{Z}_{m(2^{2^l}+1)}$

$m = 2^{\lfloor k/2 \rfloor}$; $2^{2^l} + 1$ нечетно; следовательно, $\gcd(m, 2^{2^l} + 1) = 1$

Китайская теорема об остатках

Пусть $x \equiv x' \pmod{p}$, $x \equiv x'' \pmod{q}$, $\gcd(p, q) = 1$

Тогда $x \equiv x'q\bar{q} + x''p\bar{p} \pmod{pq}$, где $p\bar{p} \equiv 1 \pmod{q}$, $q\bar{q} \equiv 1 \pmod{p}$

Доказательство в курсе теории чисел □

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Пусть $E_i \equiv E'_i \pmod{m}$, $E_i \equiv E''_i \pmod{2^{2l} + 1}$,

Имеем $(2^{2l} + 1) \cdot 1 \equiv 1 \pmod{m}$, поскольку $m \mid 2^{2l}$

Имеем $m \cdot (2^{2l} + 1 - 2^{2l}/m) = m(2^{2l} + 1) - 2^{2l} \equiv 1 \pmod{2^{2l} + 1}$

По китайской теореме об остатках:

$$\begin{aligned} E_i &\equiv E'_i \cdot (2^{2l} + 1) \cdot 1 + E''_i \cdot m \cdot (2^{2l} + 1 - 2^{2l}/m) = \\ E'_i \cdot (2^{2l} + 1) \cdot 1 - E''_i \cdot 2^{2l} &= (2^{2l} + 1)(E'_i - E''_i) + E''_i \pmod{m(2^{2l} + 1)} \end{aligned}$$

Пусть $E'_i - E''_i = Qm + R \equiv R \pmod{m}$, $0 \leq R < m$

$$\begin{aligned} (2^{2l} + 1)(E'_i - E''_i) + E''_i &= Qm(2^{2l} + 1) + R(2^{2l} + 1) + E''_i \equiv \\ R(2^{2l} + 1) + E''_i &\pmod{m(2^{2l} + 1)} \end{aligned}$$

Следовательно, для вычисления $E'_i - E''_i \pmod{m(2^{2l} + 1)}$ требуются только приведение \pmod{m} (где m — степень двойки), сдвиг и сложение

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Вычисление $E' \equiv A \circledast_m B \pmod{m}$

$$A = [A_0, \dots, A_{m-1}]^T \equiv A' = [A'_0, \dots, A'_{m-1}]^T \pmod{m} \quad 0 \leq A'_i < m$$

$$B = [B_0, \dots, B_{m-1}]^T \equiv B' = [B'_0, \dots, B'_{m-1}]^T \pmod{m} \quad 0 \leq B'_i < m$$

Поскольку $m = 2^{\lfloor k/2 \rfloor}$, каждый разряд A', B' имеет длину $\lfloor k/2 \rfloor$ бит

Всего в A', B' по $m \lfloor k/2 \rfloor = \Theta(n^{1/2} \log n)$ бит. Наивное вычисление $A' * B'$ или $A' \circledast_m B'$ требует $\Theta((n^{1/2} \log n)^2) = \Theta(n(\log n)^2)$ битовых операций. Нужно уложиться в $O(n)$ битовых операций.

Добавляем к каждому разряду A', B' по два нулевых разряда, формируя расширенные разряды длины $3 \lfloor k/2 \rfloor$ бит

$$A'' = [A'_0, 0, 0, A'_1, 0, 0, \dots, A'_{m-1}, 0, 0] \quad A''(m) = A'(m^3)$$

$$B'' = [B'_0, 0, 0, B'_1, 0, 0, \dots, B'_{m-1}, 0, 0] \quad B''(m) = B'(m^3)$$

Вычисляем $A''(m) \cdot B''(m) = C''(m)$ алгоритмом Карацубы

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Вычисление $E' \equiv A \circledast B \pmod{m}$ (окончание)

В каждом расширенном разряде $C''(m)$ сумма $\leq m$ чисел $< m^2$, значение разряда $< m \cdot m^2 = m^3 = 2^{3\lfloor k/2 \rfloor}$. Следовательно, переносов из расширенных разрядов нет: $A''(x) \cdot B''(x) = C''(x)$

Расширенные разряды C'' — элементы $A' * B'$, отсюда получаем $A' \circledast B'$ попарным вычитанием, $E' \equiv A' \circledast B' \pmod{m}$ усечением

Трудоемкость этого этапа $O((3m\lfloor k/2 \rfloor)^{\log_2 3}) = O((n^{1/2} \log n)^{\log_2 3}) = O((n^{1/2} \log n)^{1.59}) = O(n^{1.60/2}) = O(n)$

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Вычисление $E'' \equiv A \circledast B \pmod{2^{2l} + 1}$ методом FFT

В векторах A, B по m элементов

Нужен ψ — первообразный $\sqrt[2m]{1}$ в $\mathbb{Z}_{2^{2l}+1}$ $\omega = \psi^2$

$\psi = 2^{2l/m}$ подходит по теореме о первообразном корне для $\mathbb{Z}_{2^{2l}+1}$:
 $m \mid l$, $\psi^{2m/2} + 1 = (2^{2l/m})^m + 1 = 2^{2l} + 1$

Прямое и обратное DFT: трудоемкость $O(m^2 \log m \cdot 4l/m) = O(n \log n)$

Поэлементное умножение результатов прямого DFT: m операций умножения $2l$ -битовых чисел, вычисляется рекурсивно; трудоемкость $m \cdot M(2l)$

Общая трудоемкость этого этапа

$O(m^2 \log m \cdot 4l/m + m \cdot M(2l)) = O(n \log n + m \cdot M(2l))$

Дискретное преобразование Фурье и его применения

Умножение многозначных чисел

Трудоёмкость вычисления E в $\mathbb{Z}_{2^{2l}+1}$: $O(n \log n + m \cdot M(2l))$

Трудоёмкость вычисления E в \mathbb{Z}_m и прочих вычислений: $O(n)$ — пренебрежима

Итоговая трудоёмкость $M(n) = O(n \log n + m \cdot M(2l))$

Пусть $M'(x) = \frac{M(x)}{x}$

$$M'(n) = c \log n + \frac{mM(2l)}{n} = c \log n + \frac{2mM(2l)}{2ml} = c \log n + 2M'(2l)$$

Утверждение: $M'(x) = c' \log x \log \log x$ удовлетворяет этому рекуррентному соотношению при некотором c' (упражнение)

Следовательно, $M(n) = O(n \log n \log \log n)$

- 1 Дискретное преобразование Фурье и его применения
- 2 Параллельные алгоритмы**
- 3 Оптимизационные задачи и приближенные алгоритмы

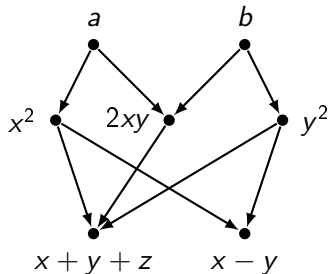
Параллельные алгоритмы

Вычисления схемами

Вычислительная **схема** как базовая модель параллелизма

$$a^2 + 2ab + b^2$$

$$a^2 - b^2$$



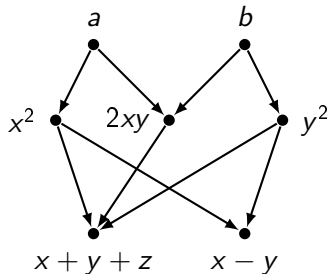
Параллельные алгоритмы

Вычисления схемами

Вычислительная **схема** как базовая модель параллелизма

$$a^2 + 2ab + b^2$$

$$a^2 - b^2$$



Ориентированный ациклический граф (dag), фиксированное количество входов/выходов. Вычисления неадаптивны (oblivious): последовательность операций не зависит от входа.

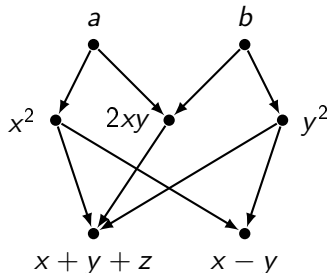
Параллельные алгоритмы

Вычисления схемами

Вычислительная **схема** как базовая модель параллелизма

$$a^2 + 2ab + b^2$$

$$a^2 - b^2$$



Ориентированный ациклический граф (dag), фиксированное количество входов/выходов. Вычисления неадаптивны (oblivious): последовательность операций не зависит от входа.

Вычисления над переменным количеством входов: (бесконечное) **семейство схем**. Семейство схем с конечным описанием — алгоритм.

Параллельные алгоритмы

Вычисления схемами

В семействе схем, входящая/выходящая степень узлов может быть ограничена (константой) или неограничена: например, сумма двух аргументов vs сумма n аргументов

Возможные операции в узлах

- арифметические/булевы/сравнения
- каждая (обычно) выполняется за константное время

Параллельные алгоритмы

Вычисления схемами

В семействе схем, входящая/выходящая степень узлов может быть ограничена (константой) или неограничена: например, сумма двух аргументов vs сумма n аргументов

Возможные операции в узлах

- арифметические/булевы/сравнения
- каждая (обычно) выполняется за константное время

размер = количество узлов

глубина = максимальная длина пути от входа до выхода

Параллельные алгоритмы

Вычисления схемами

В семействе схем, входящая/выходящая степень узлов может быть ограничена (константой) или неограничена: например, сумма двух аргументов vs сумма n аргументов

Возможные операции в узлах

- арифметические/булевы/сравнения
- каждая (обычно) выполняется за константное время

размер = количество узлов

глубина = максимальная длина пути от входа до выхода

Параллельные алгоритмы

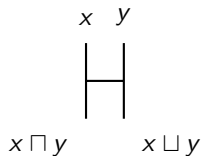
Схемы сравнения

Схема сравнения: схема, в которой узлы — **элементы сравнения**

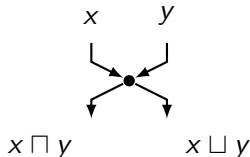
Параллельные алгоритмы

Схемы сравнения

Схема сравнения: схема, в которой узлы — **элементы сравнения**



обозначает



$\sqcap = \min$

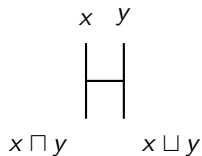
$\sqcup = \max$

Вход/выход: последовательности одинаковой длины, их члены — элементы линейно упорядоченного множества

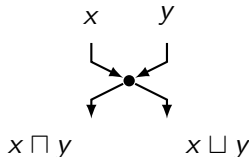
Параллельные алгоритмы

Схемы сравнения

Схема сравнения: схема, в которой узлы — **элементы сравнения**



обозначает

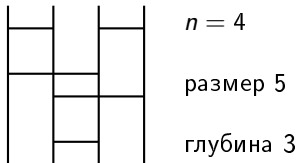


$\sqcap = \min$

$\sqcup = \max$

Вход/выход: последовательности одинаковой длины, их члены — элементы линейно упорядоченного множества

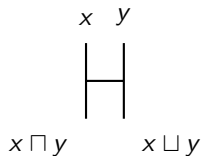
Примеры:



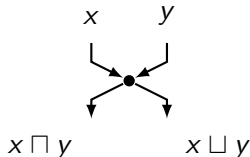
Параллельные алгоритмы

Схемы сравнения

Схема сравнения: схема, в которой узлы — **элементы сравнения**



обозначает

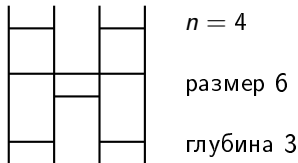
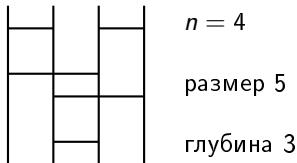


$\sqcap = \min$

$\sqcup = \max$

Вход/выход: последовательности одинаковой длины, их члены — элементы линейно упорядоченного множества

Примеры:



Параллельные алгоритмы

Схемы сравнения

Схема слияния — схема сравнения, у которой на входе две отсортированные последовательности длины n' , n'' , а на выходе — отсортированная последовательность длины $n = n' + n''$

Схема сортировки — схема сравнения, у которой на входе произвольная последовательность, а на выходе — отсортированная последовательность

Параллельные алгоритмы

Схемы сравнения

Схема слияния — схема сравнения, у которой на входе две отсортированные последовательности длины n' , n'' , а на выходе — отсортированная последовательность длины $n = n' + n''$

Схема сортировки — схема сравнения, у которой на входе произвольная последовательность, а на выходе — отсортированная последовательность

Семейство схем с конечным описанием — алгоритм сортировки. Их размер и глубина — последовательная и параллельная сложность алгоритма.

Параллельные алгоритмы

Схемы сравнения

Схема слияния — схема сравнения, у которой на входе две отсортированные последовательности длины n' , n'' , а на выходе — отсортированная последовательность длины $n = n' + n''$

Схема сортировки — схема сравнения, у которой на входе произвольная последовательность, а на выходе — отсортированная последовательность

Семейство схем с конечным описанием — алгоритм сортировки. Их размер и глубина — последовательная и параллельная сложность алгоритма.

Общее (адаптивное) слияние: $O(n)$

Общая (адаптивная) сортировка: $O(n \log n)$ сравнений (алгоритм сортировки слиянием)

Какова сложность неадаптивной сортировки?

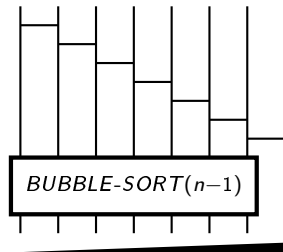
Параллельные алгоритмы

Сортировка пузырьком

BUBBLE-SORT(n)

размер $n(n-1)/2 = O(n^2)$

глубина $2n-3 = O(n)$



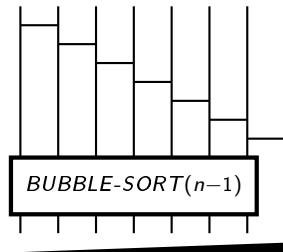
Параллельные алгоритмы

Сортировка пузырьком

BUBBLE-SORT(n)

размер $n(n-1)/2 = O(n^2)$

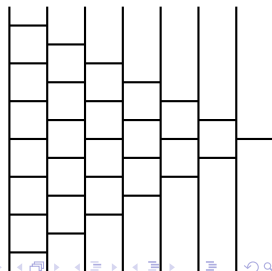
глубина $2n-3 = O(n)$



BUBBLE-SORT(8)

размер 28

глубина 13



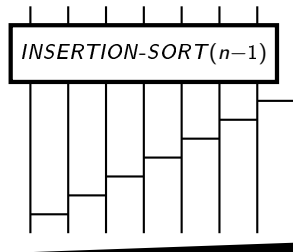
Параллельные алгоритмы

Сортировка пузырьком

INSERTION-SORT(n)

размер $n(n-1)/2 = O(n^2)$

глубина $2n - 3 = O(n)$



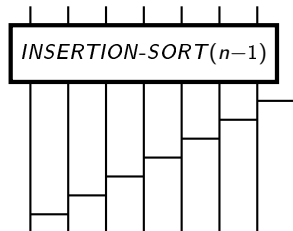
Параллельные алгоритмы

Сортировка пузырьком

INSERTION-SORT(n)

размер $n(n-1)/2 = O(n^2)$

глубина $2n-3 = O(n)$

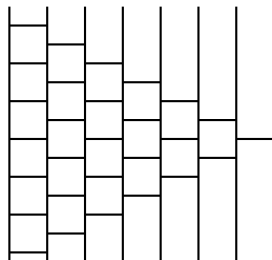


INSERTION-SORT(8)

размер 28

глубина 13

Идентична *BUBBLE-SORT*!



Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Доказательство.

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Доказательство. “Только тогда”: тривиально.

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Доказательство. “Только тогда”: тривиально. “Тогда”: от противного.

Предположим, что заданная сеть не сортирует вход $x = \langle x_1, \dots, x_n \rangle$

$$\langle x_1, \dots, x_n \rangle \mapsto \langle y_1, \dots, y_n \rangle \quad \exists k, l : k < l : y_k > y_l$$

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Доказательство. “Только тогда”: тривиально. “Тогда”: от противного.

Предположим, что заданная сеть не сортирует вход $x = \langle x_1, \dots, x_n \rangle$

$$\langle x_1, \dots, x_n \rangle \mapsto \langle y_1, \dots, y_n \rangle \quad \exists k, l : k < l : y_k > y_l$$

Положим $X_i = \begin{cases} 0 & \text{if } x_i < y_k \\ 1 & \text{if } x_i \geq y_k \end{cases}$, запустим сеть на входе $X = \langle X_1, \dots, X_n \rangle$

Для всех i, j имеем $x_i \leq x_j \Rightarrow X_i \leq X_j$, следовательно путь X_i по сети тот же, что у x_i

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц: Схема сравнения является сортирующей т. и т.т., когда она сортирует все последовательности нулей и единиц

Доказательство. “Только тогда”: тривиально. “Тогда”: от противного.

Предположим, что заданная сеть не сортирует вход $x = \langle x_1, \dots, x_n \rangle$

$$\langle x_1, \dots, x_n \rangle \mapsto \langle y_1, \dots, y_n \rangle \quad \exists k, l : k < l : y_k > y_l$$

Положим $X_i = \begin{cases} 0 & \text{if } x_i < y_k \\ 1 & \text{if } x_i \geq y_k \end{cases}$, запустим сеть на входе $X = \langle X_1, \dots, X_n \rangle$

Для всех i, j имеем $x_i \leq x_j \Rightarrow X_i \leq X_j$, следовательно путь X_i по сети тот же, что у x_i

$$\langle X_1, \dots, X_n \rangle \mapsto \langle Y_1, \dots, Y_n \rangle \quad Y_k = 1 > 0 = Y_l$$

Имеем $k < l$, но $Y_k > Y_l$ — сеть не сортирует нули и единицы. □

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц применим к сортировке, слиянию и другим задачам сравнения (например, выбору порядковой статистики)

Параллельные алгоритмы

Принцип нулей-единиц

Принцип нулей-единиц применим к сортировке, слиянию и другим задачам сравнения (например, выбору порядковой статистики)

Позволяет протестировать:

- сеть сортировки путем проверки только 2^n входных последовательностей, вместо $n! = (1 + o(1))(2\pi n)^{1/2} \cdot (n/e)^n \gg 2^n$
- сеть слияния путем проверки только $(n' + 1) \cdot (n'' + 1)$ пар входных последовательностей вместо $\binom{n}{n'} = \binom{n}{n''}$, например для $n = 2n' = 2n''$: $\binom{n}{n'} = (1 + o(1))(\pi n/2)^{-1/2} \cdot 2^n \ggg (n' + 1) \cdot (n'' + 1)$

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Общее (адаптивное) слияние: $O(n)$ сравнений

Насколько эффективно можно выполнить слияние неадаптивно?

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Общее (адаптивное) слияние: $O(n)$ сравнений

Насколько эффективно можно выполнить слияние неадаптивно?

$$\langle x_1 \leq \dots \leq x_{n'} \rangle, \langle y_1 \leq \dots \leq y_{n''} \rangle \mapsto \langle z_1 \leq \dots \leq z_n \rangle$$

Нечетно-четное слияние

Если $n' = n'' = 1$ сравниваем (x_1, y_1) , иначе рекурсивно

- слияние $\langle x_1, x_3, \dots \rangle, \langle y_1, y_3, \dots \rangle \mapsto \langle u_1 \leq u_2 \leq \dots \leq u_{\lceil n'/2 \rceil + \lceil n''/2 \rceil} \rangle$
- слияние $\langle x_2, x_4, \dots \rangle, \langle y_2, y_4, \dots \rangle \mapsto \langle v_1 \leq v_2 \leq \dots \leq v_{\lfloor n'/2 \rfloor + \lfloor n''/2 \rfloor} \rangle$
- попарное сравнение: $(u_2, v_1), (u_3, v_2), \dots$

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Общее (адаптивное) слияние: $O(n)$ сравнений

Насколько эффективно можно выполнить слияние неадаптивно?

$$\langle x_1 \leq \dots \leq x_{n'} \rangle, \langle y_1 \leq \dots \leq y_{n''} \rangle \mapsto \langle z_1 \leq \dots \leq z_n \rangle$$

Нечетно-четное слияние

Если $n' = n'' = 1$ сравниваем (x_1, y_1) , иначе рекурсивно

- слияние $\langle x_1, x_3, \dots \rangle, \langle y_1, y_3, \dots \rangle \mapsto \langle u_1 \leq u_2 \leq \dots \leq u_{\lceil n'/2 \rceil + \lceil n''/2 \rceil} \rangle$
- слияние $\langle x_2, x_4, \dots \rangle, \langle y_2, y_4, \dots \rangle \mapsto \langle v_1 \leq v_2 \leq \dots \leq v_{\lfloor n'/2 \rfloor + \lfloor n''/2 \rfloor} \rangle$
- попарное сравнение: $(u_2, v_1), (u_3, v_2), \dots$

$$\text{size}(\text{OEM}(n', n'')) \leq 2 \cdot \text{size}(\text{OEM}(n'/2, n''/2)) + O(n) = O(n \log n)$$

$$\text{depth}(\text{OEM}(n', n'')) \leq \text{depth}(\text{OEM}(n'/2, n''/2)) + 1 = O(\log n)$$

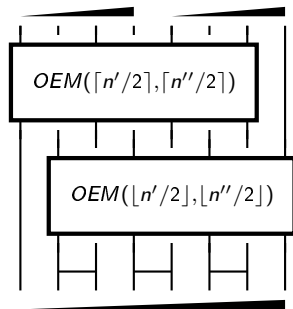
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$OEM(n', n'')$

размер $O(n \log n)$

глубина $O(\log n)$



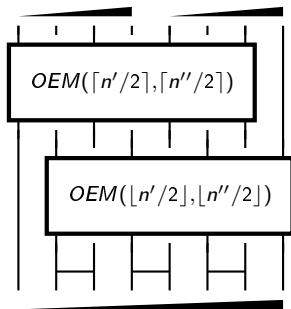
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$OEM(n', n'')$

размер $O(n \log n)$

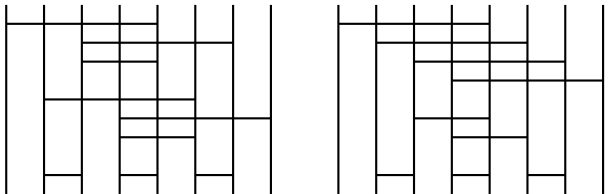
глубина $O(\log n)$



$OEM(4, 4)$

размер 9

глубина 3



Параллельные алгоритмы

Эффективные сети сортировки и слияния

Доказательство корректности нечетно-четного слияния:

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Доказательство корректности нечетно-четного слияния: индукция, принцип нулей-единиц

База: тривиальна (2 входа, 1 элемент)

Переход. Индукционное предположение: нечетное и четное слияния корректны

Рассмотрим вход из нулей и единиц. Имеем для всех k, l :

$\langle 0^{\lceil k/2 \rceil} 11 \dots \rangle, \langle 0^{\lceil l/2 \rceil} 11 \dots \rangle \mapsto \langle 0^{\lceil k/2 \rceil + \lceil l/2 \rceil} 11 \dots \rangle$ нечетным слиянием

$\langle 0^{\lfloor k/2 \rfloor} 11 \dots \rangle, \langle 0^{\lfloor l/2 \rfloor} 11 \dots \rangle \mapsto \langle 0^{\lfloor k/2 \rfloor + \lfloor l/2 \rfloor} 11 \dots \rangle$ четным слиянием

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Доказательство корректности нечетно-четного слияния: индукция, принцип нулей-единиц

База: тривиальна (2 входа, 1 элемент)

Переход. Индукционное предположение: нечетное и четное слияния корректны

Рассмотрим вход из нулей и единиц. Имеем для всех k, l :

$\langle 0^{\lceil k/2 \rceil} 11 \dots \rangle, \langle 0^{\lceil l/2 \rceil} 11 \dots \rangle \mapsto \langle 0^{\lceil k/2 \rceil + \lceil l/2 \rceil} 11 \dots \rangle$ нечетным слиянием

$\langle 0^{\lfloor k/2 \rfloor} 11 \dots \rangle, \langle 0^{\lfloor l/2 \rfloor} 11 \dots \rangle \mapsto \langle 0^{\lfloor k/2 \rfloor + \lfloor l/2 \rfloor} 11 \dots \rangle$ четным слиянием

$$(\lceil k/2 \rceil + \lceil l/2 \rceil) - (\lfloor k/2 \rfloor + \lfloor l/2 \rfloor) = \begin{cases} 0, 1 & \text{все выходы упорядочены: } \langle 0^{k+l} 11 \dots \rangle \\ 2 & \text{одна пара неупорядочена: } \langle 0^{k+l-1} 1011 \dots \rangle \end{cases}$$

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Доказательство корректности нечетно-четного слияния: индукция, принцип нулей-единиц

База: тривиальна (2 входа, 1 элемент)

Переход. Индукционное предположение: нечетное и четное слияния корректны

Рассмотрим вход из нулей и единиц. Имеем для всех k, l :

$\langle 0^{\lceil k/2 \rceil} 11 \dots \rangle, \langle 0^{\lceil l/2 \rceil} 11 \dots \rangle \mapsto \langle 0^{\lceil k/2 \rceil + \lceil l/2 \rceil} 11 \dots \rangle$ нечетным слиянием

$\langle 0^{\lfloor k/2 \rfloor} 11 \dots \rangle, \langle 0^{\lfloor l/2 \rfloor} 11 \dots \rangle \mapsto \langle 0^{\lfloor k/2 \rfloor + \lfloor l/2 \rfloor} 11 \dots \rangle$ четным слиянием

$$(\lceil k/2 \rceil + \lceil l/2 \rceil) - (\lfloor k/2 \rfloor + \lfloor l/2 \rfloor) = \begin{cases} 0, 1 & \text{все выходы упорядочены: } \langle 0^{k+l} 11 \dots \rangle \\ 2 & \text{одна пара неупорядочена: } \langle 0^{k+l-1} 1011 \dots \rangle \end{cases}$$

Заключительный слой элементов сравнения корректирует единственную неупорядоченную пару

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Сортировка произвольного входа $\langle x_1, \dots, x_n \rangle$

Сортировка нечетно-четным слиянием

[Batcher: 1968]

Если $n = 1$ останавливаемся, иначе рекурсивно:

- сортировка $\langle x_1, \dots, x_{\lceil n/2 \rceil} \rangle$
- сортировка $\langle x_{\lceil n/2 \rceil + 1}, \dots, x_n \rangle$
- слияние результатов при помощи $OEM(\lceil n/2 \rceil, \lfloor n/2 \rfloor)$

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Сортировка произвольного входа $\langle x_1, \dots, x_n \rangle$

Сортировка нечетно-четным слиянием

[Batcher: 1968]

Если $n = 1$ останавливаемся, иначе рекурсивно:

- сортировка $\langle x_1, \dots, x_{\lceil n/2 \rceil} \rangle$
- сортировка $\langle x_{\lceil n/2 \rceil + 1}, \dots, x_n \rangle$
- слияние результатов при помощи $OEM(\lceil n/2 \rceil, \lfloor n/2 \rfloor)$

$$\begin{aligned} \text{size}(OEM\text{-}SORT)(n) &\leq \\ 2 \cdot \text{size}(OEM\text{-}SORT(n/2)) + \text{size}(OEM(n/2, n/2)) &= \\ 2 \cdot \text{size}(OEM\text{-}SORT(n/2)) + O(n \log n) &= O(n(\log n)^2) \end{aligned}$$

$$\begin{aligned} \text{depth}(OEM\text{-}SORT(n)) &\leq \\ \text{depth}(OEM\text{-}SORT(n/2)) + \text{depth}(OEM(n/2, n/2)) &= \\ \text{depth}(OEM\text{-}SORT(n/2)) + O(\log n) &= O((\log n)^2) \end{aligned}$$

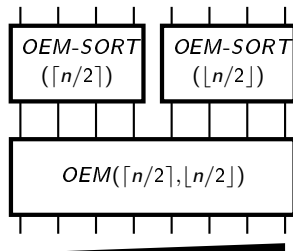
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$OEM-SORT(n)$

размер $O(n(\log n)^2)$

глубина $O((\log n)^2)$



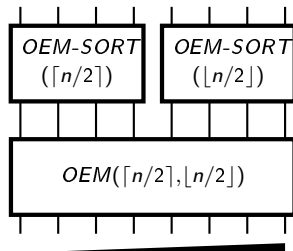
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$OEM-SORT(n)$

размер $O(n(\log n)^2)$

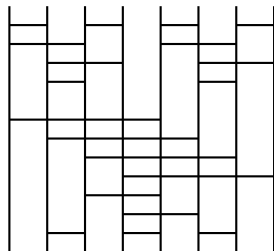
глубина $O((\log n)^2)$



$OEM-SORT(8)$

размер 19

глубина 6



Параллельные алгоритмы

Эффективные сети сортировки и слияния

Битонная последовательность: $\langle x_1 \geq \dots \geq x_m \leq \dots \leq x_n \rangle \quad 1 \leq m \leq n$

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Битонная последовательность: $\langle x_1 \geq \dots \geq x_m \leq \dots \leq x_n \rangle \quad 1 \leq m \leq n$

Битонное слияние: сортировка битонной последовательности

Если $n = 1$ останавливаемся, иначе рекурсивно:

- битонное слияние $\langle x_1, x_3, \dots \rangle \mapsto \langle u_1 \leq u_2 \leq \dots \leq u_{\lceil n/2 \rceil} \rangle$
- битонное слияние $\langle x_2, x_4, \dots \rangle \mapsto \langle v_1 \leq v_2 \leq \dots \leq v_{\lfloor n/2 \rfloor} \rangle$
- попарное сравнение: $(u_1, v_1), (u_2, v_2), \dots$

Доказательство корректности — упражнение

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Битонная последовательность: $\langle x_1 \geq \dots \geq x_m \leq \dots \leq x_n \rangle \quad 1 \leq m \leq n$

Битонное слияние: сортировка битонной последовательности

Если $n = 1$ останавливаемся, иначе рекурсивно:

- битонное слияние $\langle x_1, x_3, \dots \rangle \mapsto \langle u_1 \leq u_2 \leq \dots \leq u_{\lceil n/2 \rceil} \rangle$
- битонное слияние $\langle x_2, x_4, \dots \rangle \mapsto \langle v_1 \leq v_2 \leq \dots \leq v_{\lfloor n/2 \rfloor} \rangle$
- попарное сравнение: $(u_1, v_1), (u_2, v_2), \dots$

Доказательство корректности — упражнение

Битонное слияние более гибко, чем нечетно-четное слияние, поскольку одна и та же схема может осуществлять слияние для любого m при фиксированном n

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Битонная последовательность: $\langle x_1 \geq \dots \geq x_m \leq \dots \leq x_n \rangle \quad 1 \leq m \leq n$

Битонное слияние: сортировка битонной последовательности

Если $n = 1$ останавливаемся, иначе рекурсивно:

- битонное слияние $\langle x_1, x_3, \dots \rangle \mapsto \langle u_1 \leq u_2 \leq \dots \leq u_{\lceil n/2 \rceil} \rangle$
- битонное слияние $\langle x_2, x_4, \dots \rangle \mapsto \langle v_1 \leq v_2 \leq \dots \leq v_{\lfloor n/2 \rfloor} \rangle$
- попарное сравнение: $(u_1, v_1), (u_2, v_2), \dots$

Доказательство корректности — упражнение

Битонное слияние более гибко, чем нечетно-четное слияние, поскольку одна и та же схема может осуществлять слияние для любого m при фиксированном n

$$\text{size}(BM(n)) = O(n \log n) \quad \text{depth}(BM(n)) = O(\log n)$$

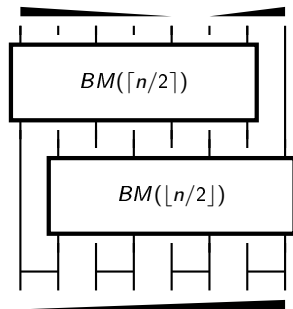
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$BM(n)$

размер $O(n \log n)$

глубина $O(\log n)$



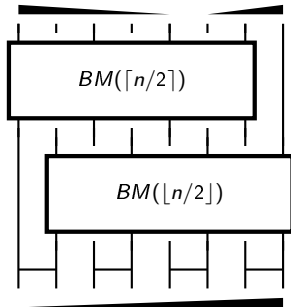
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$BM(n)$

размер $O(n \log n)$

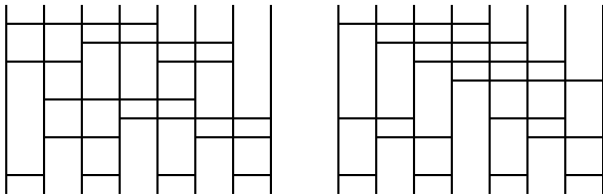
глубина $O(\log n)$



$BM(8)$

размер 12

глубина 3



Сортировка битонным слиянием

[Batcher: 1968]

Если $n = 1$ останавливаемся, иначе рекурсивно:

- сортируем $\langle x_1, \dots, x_{\lceil n/2 \rceil} \rangle \mapsto \langle y_1 \geq \dots \geq y_{\lceil n/2 \rceil} \rangle$ в обратном порядке
- сортируем $\langle x_{\lceil n/2 \rceil + 1}, \dots, x_n \rangle \mapsto \langle y_{\lceil n/2 \rceil + 1} \leq \dots \leq y_n \rangle$
- битонное слияние $\langle y_1 \geq \dots \geq y_m \leq \dots \leq y_n \rangle$,
 $m \in \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1\}$

Казалось бы, для сортировки в обратном порядке требуются “обратные элементы сравнения”

Сортировка битонным слиянием

[Batcher: 1968]

Если $n = 1$ останавливаемся, иначе рекурсивно:

- сортируем $\langle x_1, \dots, x_{\lceil n/2 \rceil} \rangle \mapsto \langle y_1 \geq \dots \geq y_{\lceil n/2 \rceil} \rangle$ в обратном порядке
- сортируем $\langle x_{\lceil n/2 \rceil + 1}, \dots, x_n \rangle \mapsto \langle y_{\lceil n/2 \rceil + 1} \leq \dots \leq y_n \rangle$
- битонное слияние $\langle y_1 \geq \dots \geq y_m \leq \dots \leq y_n \rangle$,
 $m \in \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1\}$

Казалось бы, для сортировки в обратном порядке требуются “обратные элементы сравнения”, однако элементы сравнения — это узлы схемы, а прямое/обратное направление сортировки — артефакт рисунка схемы на плоскости

$$\begin{aligned} \text{size}(BM\text{-}SORT(n)) &= O(n(\log n)^2) \\ \text{depth}(BM\text{-}SORT(n)) &= O((\log n)^2) \end{aligned}$$

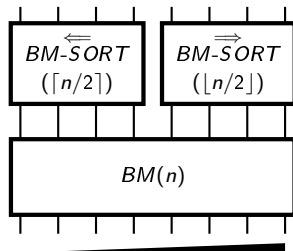
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$BM-SORT(n)$

размер $O(n(\log n)^2)$

глубина $O((\log n)^2)$



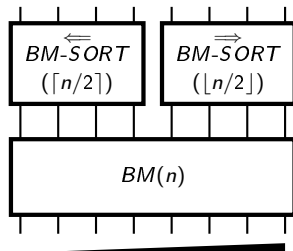
Параллельные алгоритмы

Эффективные сети сортировки и слияния

$BM-SORT(n)$

размер $O(n(\log n)^2)$

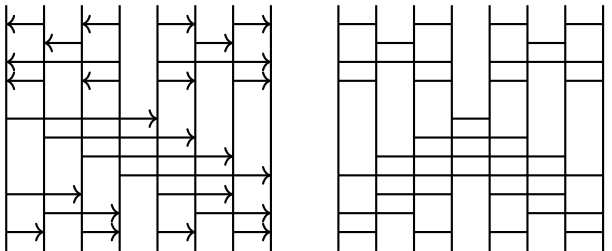
глубина $O((\log n)^2)$



$BM-SORT(8)$

размер 24

глубина 6



Параллельные алгоритмы

Эффективные сети сортировки и слияния

Как *OEM-SORT*, так и *BM-SORT* имеют размер $\Theta(n(\log n)^2)$

Возможна ли неадаптивная сортировка схемой размера $o(n(\log n)^2)$?
 $O(n \log n)$?

Параллельные алгоритмы

Эффективные сети сортировки и слияния

Как *OEM-SORT*, так и *BM-SORT* имеют размер $\Theta(n(\log n)^2)$

Возможна ли неадаптивная сортировка схемой размера $o(n(\log n)^2)$?
 $O(n \log n)$?

Схема AKS

[Ajtai, Komlós, Szemerédi: 1983]

[Paterson: 1990]; [Seiferas: 2009]

размер $O(n \log n)$, глубина $O(\log n)$

Использует глубокие понятия теории графов (**экспандеры**)

Асимптотически оптимальна, но имеет огромный константный множитель

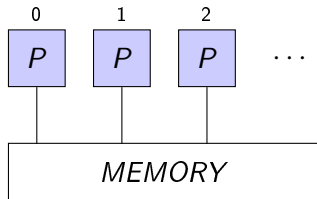
Параллельные алгоритмы

Модели параллельных вычислений

Parallel Random Access Machine (PRAM)

Простая, идеализированная модель общих параллельных вычислений

[Fortune, Wyllie: 1978]



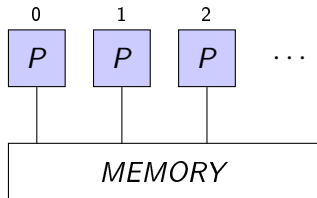
Параллельные алгоритмы

Модели параллельных вычислений

Parallel Random Access Machine (PRAM)

[Fortune, Wyllie: 1978]

Простая, идеализированная модель общих параллельных вычислений



Включает

- неограниченное количество **процессоров** (1 операция за единицу времени)
- глобальная общая память (1 доступ за единицу времени)

Вычисления полностью синхронны

Параллельные алгоритмы

Модели параллельных вычислений

Вычисление PRAM: последовательность параллельных **шагов**

Коммуникация и синхронизация считаются “бесплатными”

Труднореализуемо на практике!

Параллельные алгоритмы

Модели параллельных вычислений

Вычисление PRAM: последовательность параллельных **шагов**

Коммуникация и синхронизация считаются “бесплатными”

Труднореализуемо на практике!

Варианты PRAM:

- concurrent/exclusive read
- concurrent/exclusive write

Параллельные алгоритмы

Модели параллельных вычислений

Вычисление PRAM: последовательность параллельных **шагов**

Коммуникация и синхронизация считаются “бесплатными”

Труднореализуемо на практике!

Варианты PRAM:

- concurrent/exclusive read
- concurrent/exclusive write

Класс NC: задачи, решаемые на PRAM с $O(n^c)$ процессорами за $O((\log n)^d)$ шагов, $c, d = O(1)$

Например, сложность одного конкретного алгоритма PRAM для решения линейной системы: $O((\log n)^2)$ шагов на n^4 процессорах : –0

Разработка алгоритмов PRAM: минимизация количества шагов, также иногда количества процессоров

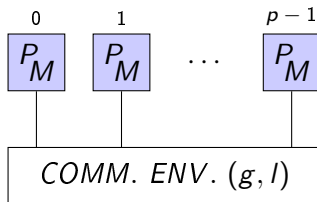
Параллельные алгоритмы

Модели параллельных вычислений

Bulk-Synchronous Parallel (BSP) computer

[Valiant: 1990]

Простая, реалистичная модель общих
параллельных вычислений —
масштабируемая, переносимая,
предсказуемая



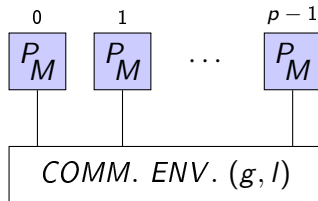
Параллельные алгоритмы

Модели параллельных вычислений

Bulk-Synchronous Parallel (BSP) computer

[Valiant: 1990]

Простая, реалистичная модель общих параллельных вычислений — масштабируемая, переносимая, предсказуемая



Включает

- p процессоров, каждый с локальной памятью (1 операция за единицу времени)
- коммуникационная среда, состоящая из сети и (возможно) внешней памяти (1 единица данных за g единиц времени)
- механизм барьерной синхронизации (не чаще 1 раза за l единиц времени)

Некоторые компоненты BSP могут быть реализованы при помощи других механизмов, например

- внешняя память при помощи локальной памяти и коммуникации
- барьерная синхронизация при помощи попарной синхронизации

Некоторые компоненты BSP могут быть реализованы при помощи других механизмов, например

- внешняя память при помощи локальной памяти и коммуникации
- барьерная синхронизация при помощи попарной синхронизации

Параметры коммуникационной среды:

- g — **communication gap** (inverse bandwidth), время (в худшем случае) для единицы данных войти в сеть или покинуть сеть
- l — **latency**, время (в худшем случае) для единицы данных быть переданной внутри сети

Параллельные алгоритмы

Модели параллельных вычислений

Некоторые компоненты BSP могут быть реализованы при помощи других механизмов, например

- внешняя память при помощи локальной памяти и коммуникации
- барьерная синхронизация при помощи попарной синхронизации

Параметры коммуникационной среды:

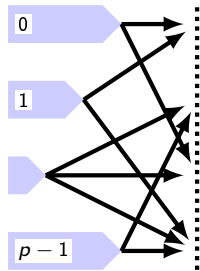
- g — **communication gap** (inverse bandwidth), время (в худшем случае) для единицы данных войти в сеть или покинуть сеть
- l — **latency**, время (в худшем случае) для единицы данных быть переданной внутри сети

Параллельную систему можно (приблизительно) описать параметрами p , g , l . Эффективность сетей исторически растет медленнее, чем процессоров, и требует больше энергии: $g, l \gg 1$. Например, для компьютера Cray T3E (1995): $p = 64$, $g \approx 78$, $l \approx 1825$, с тех пор принципиально не изменились.

Параллельные алгоритмы

Модели параллельных вычислений

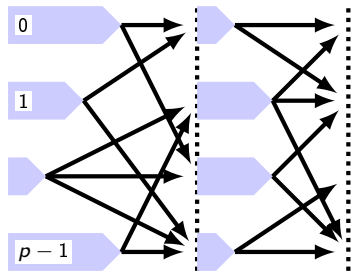
Вычисление BSP: последовательность параллельных **супершагов**



Параллельные алгоритмы

Модели параллельных вычислений

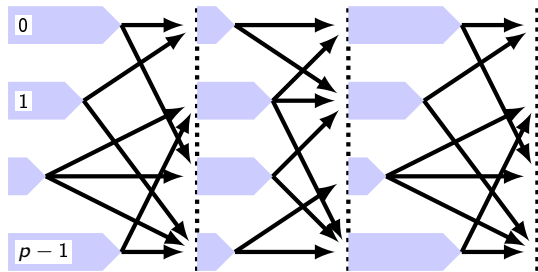
Вычисление BSP: последовательность параллельных **супершагов**



Параллельные алгоритмы

Модели параллельных вычислений

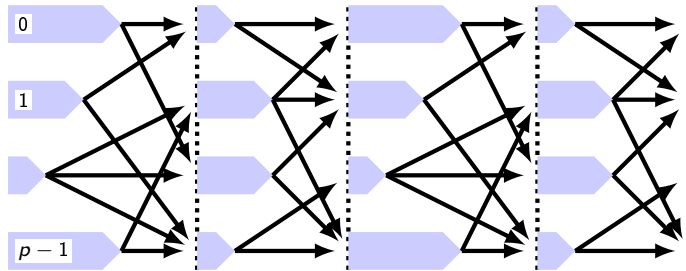
Вычисление BSP: последовательность параллельных **супершагов**



Параллельные алгоритмы

Модели параллельных вычислений

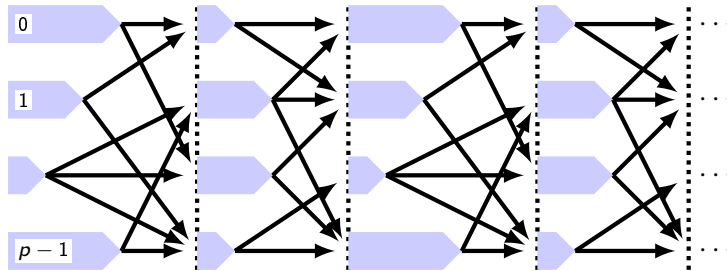
Вычисление BSP: последовательность параллельных супершагов



Параллельные алгоритмы

Модели параллельных вычислений

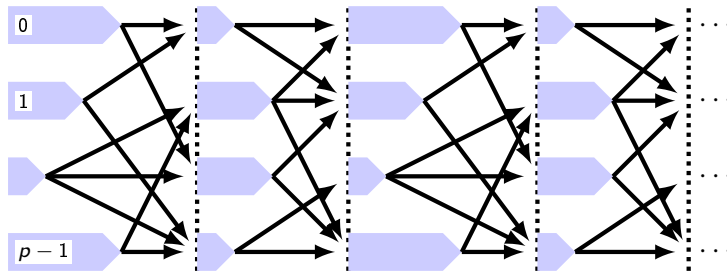
Вычисление BSP: последовательность параллельных супершагов



Параллельные алгоритмы

Модели параллельных вычислений

Вычисление BSP: последовательность параллельных **супершагов**



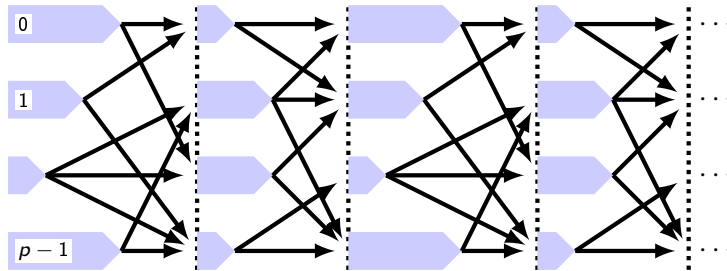
Асинхронные вычисления/коммуникация внутри супершага
(коммуникация включает обмен данными с внешней памятью)

Синхронизация между супершагами

Параллельные алгоритмы

Модели параллельных вычислений

Вычисление BSP: последовательность параллельных **супершагов**



Асинхронные вычисления/коммуникация внутри супершага
(коммуникация включает обмен данными с внешней памятью)

Синхронизация между супершагами

(Альтернативная модель — CSP: взаимодействующие последовательные процессы)

Композиционная модель стоимости вычислений

Для конкретного процессора $proc$ на супершаге $sstep$:

- $comp(sstep, proc)$: объем локальных вычислений и операций над локальной памятью процессором $proc$ на супершаге $sstep$
- $comm(sstep, proc)$: объем данных, отправленных и полученных процессором $proc$ на супершаге $sstep$

Композиционная модель стоимости вычислений

Для конкретного процессора $proc$ на супершаге $sstep$:

- $comp(sstep, proc)$: объем локальных вычислений и операций над локальной памятью процессором $proc$ на супершаге $sstep$
- $comm(sstep, proc)$: объем данных, отправленных и полученных процессором $proc$ на супершаге $sstep$

Для компьютера BSP в целом на одном супершаге $sstep$:

- $comp(sstep) = \max_{0 \leq proc < p} comp(sstep, proc)$
- $comm(sstep) = \max_{0 \leq proc < p} comm(sstep, proc)$
- $cost(sstep) = comp(sstep) + comm(sstep) \cdot g + l$

Для вычисления BSP, состоящего из *sync* супершагов:

- $comp = \sum_{0 \leq sstep < sync} comp(sstep)$
- $comm = \sum_{0 \leq sstep < sync} comm(sstep)$
- $cost = \sum_{0 \leq sstep < sync} cost(sstep) = comp + comm \cdot g + sync \cdot l$

Параллельные алгоритмы

Модели параллельных вычислений

Для вычисления BSP, состоящего из *sync* супершагов:

- $comp = \sum_{0 \leq sstep < sync} comp(sstep)$
- $comm = \sum_{0 \leq sstep < sync} comm(sstep)$
- $cost = \sum_{0 \leq sstep < sync} cost(sstep) = comp + comm \cdot g + sync \cdot l$

Входные/выходные данные хранятся во внешней памяти; стоимость ввода/вывода включена в *comm*

Параллельные алгоритмы

Модели параллельных вычислений

Для вычисления BSP, состоящего из *sync* супершагов:

- $comp = \sum_{0 \leq sstep < sync} comp(sstep)$
- $comm = \sum_{0 \leq sstep < sync} comm(sstep)$
- $cost = \sum_{0 \leq sstep < sync} cost(sstep) = comp + comm \cdot g + sync \cdot l$

Входные/выходные данные хранятся во внешней памяти; стоимость ввода/вывода включена в *comm*

Например, сложность одного конкретного алгоритма PRAM для решения линейной системы:

$$comp = O(n^3/p) \quad comm = O(n^2/p^{1/2}) \quad sync = O(p^{1/2})$$

Параллельные алгоритмы

Модели параллельных вычислений

Разработка алгоритмов для BSP

Минимизация com_r , com_t , $sync$ как функций от n , p

Соглашения:

- размер задачи $n \gg p$ (допуск)
- входные/выходные во внешней памяти, ввод/вывод — односторонняя коммуникация

Разработка алгоритмов для BSP (продолжение)

Баланс вычислений

- **work-optimal** $comp = O\left(\frac{seq\ work}{p}\right)$

Баланс коммуникации:

- цель **scalable** $comm = O\left(\frac{input+output}{p^c}\right)$, $0 < c \leq 1$
- в идеале **fully-scalable** $comm = O\left(\frac{input+output}{p}\right)$

Крупноблочность:

- цель — **sync** не зависящая от n (может зависеть от p)
- еще лучше **quasi-flat** $sync = O((\log p)^{O(1)})$
- в идеале **flat** $sync = O(1)$

BSP software: индустриальные проекты

- Google's Pregel [2010]
- Apache Hama, Spark, Giraph (apache.org) [2010–16]

BSP software: исследовательские проекты

- Oxford BSP (www.bsp-worldwide.org/implmnts/oxtool) [1998]
- Paderborn PUB (www2.cs.uni-paderborn.de/~pub) [1998]
- BSMML (traclifo.univ-orleans.fr/BSMML) [1998]
- BSPonMPI (bsponmpi.sourceforge.net) [2006]
- Multicore BSP (www.multicorebsp.com) [2011]
- Epiphany BSP (www.codu.in/ebsp) [2015]
- Petuum (petuum.org) [2015]

Параллельные алгоритмы

Сбалансированное дерево

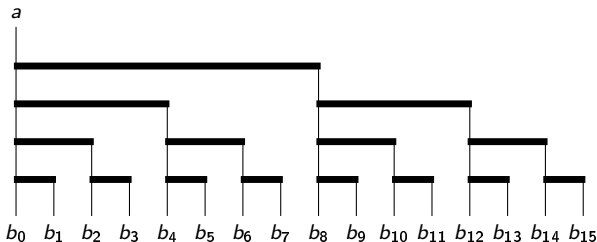
Схема на основе **сбалансированного двоичного дерева**:

$tree(n)$

1 вход, n выходов
(или наоборот)

размер $n - 1$

глубина $\log n$



Параллельные алгоритмы

Сбалансированное дерево

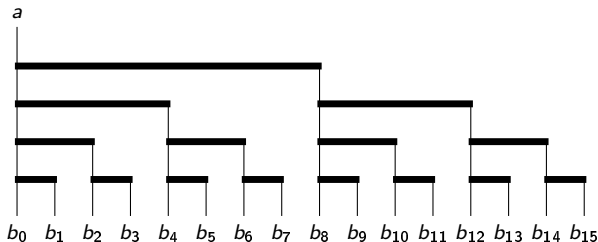
Схема на основе **сбалансированного двоичного дерева**:

$tree(n)$

1 вход, n выходов
(или наоборот)

размер $n - 1$

глубина $\log n$



Каждый узел вычисляет произвольную заданную операцию за $O(1)$

Может быть ориентирован

- сверху вниз (один вход в корне, n выходов в листьях)
- снизу вверх (n входов в листьях, один выход в корне)

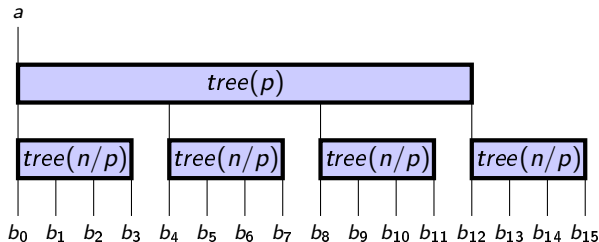
Последовательная сложность $O(n)$; сложность на PRAM $O(\log n)$

Параллельные алгоритмы

Сбалансированное дерево

Вычисление сбалансированного двоичного дерева на BSP, $p = 4$

$tree(n)$



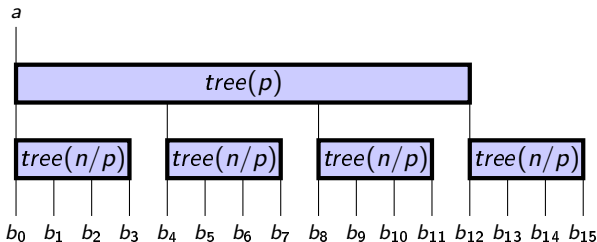
Предполагаем, что вход читается во внешней памяти, а выход туда записывается

Параллельные алгоритмы

Сбалансированное дерево

Вычисление сбалансированного двоичного дерева на BSP, $p = 4$

$tree(n)$



Предполагаем, что вход читается во внешней памяти, а выход туда записывается

Поедим $tree(n)$ на

- один верхний блок, изоморфный $tree(p)$
- нижний слой из p blocks, каждый из них изоморфный $tree(n/p)$

Параллельные алгоритмы

Сбалансированное дерево

Вычисление сбалансированного двоичного дерева на BSP
(продолжение)

При вычислении сверху вниз, произвольно назначенный процессор отвечает за верхний блок:

- читает вход блока, вычисляет блок, записывает p выходов блока

Затем каждый процессор отвечает за один из нижних блоков:

- читает вход блока, вычисляет блок, записывает n/p выходов блока

При вычислении снизу вверх, последовательность действий обратная

Параллельные алгоритмы

Сбалансированное дерево

Вычисление сбалансированного двоичного дерева на BSP
(продолжение)

При вычислении сверху вниз, произвольно назначенный процессор отвечает за верхний блок:

- читает вход блока, вычисляет блок, записывает p выходов блока

Затем каждый процессор отвечает за один из нижних блоков:

- читает вход блока, вычисляет блок, записывает n/p выходов блока

При вычислении снизу вверх, последовательность действий обратная

$$comp = O(n/p)$$

$$comm = O(n/p)$$

$$sync = O(1)$$

Предполагаем допуск $n \geq p^2$

Параллельные алгоритмы

Сбалансированное дерево

Описанный BSP-алгоритм вычисления сбалансированного дерева
полностью оптимален:

- оптимальное $comp = O(n/p) = O\left(\frac{\text{sequential work}}{p}\right)$
- оптимальное $comm = O(n/p) = O\left(\frac{\text{input/output size}}{p}\right)$
- оптимальное $sync = O(1)$

Параллельные алгоритмы

Сбалансированное дерево

Описанный BSP-алгоритм вычисления сбалансированного дерева **полностью оптимален**:

- оптимальное $comp = O(n/p) = O\left(\frac{\text{sequential work}}{p}\right)$
- оптимальное $comm = O(n/p) = O\left(\frac{\text{input/output size}}{p}\right)$
- оптимальное $sync = O(1)$

С другими задачами нам может и не повезти с полностью оптимальным алгоритмом. Однако обычно нас интересуют алгоритмы, оптимальные хотя бы по $comp$ (при разумных предположениях).

Оптимальность по $comm$ и $sync$ ставится как цель при условии оптимальности по $comp$. Например, нельзя “схитрить” и выполнить все вычисление в единственном процессоре, принося в жертву $comp$ и $comm$ ради оптимального $sync = O(1)$

Параллельные алгоритмы

Префиксное накопление

Задача **префиксного накопления**

Дан массив $a = [a_0, \dots, a_{n-1}]$

Вычислить $b_{-1} = 0 \quad b_i = a_i + b_{i-1} \quad 0 \leq i < n$

В более общем виде: ассоциативный оператор \bullet (предполагается наличие единицы ϵ , может быть добавлена формально)

Вычислить $b_{-1} = \epsilon \quad b_i = a_i \bullet b_{i-1} \quad 0 \leq i < n$

$$b_0 = a_0$$

$$b_1 = a_0 \bullet a_1$$

$$b_2 = a_0 \bullet a_1 \bullet a_2$$

...

$$b_{n-1} = a_0 \bullet a_1 \bullet \dots \bullet a_{n-1}$$

Параллельные алгоритмы

Префиксное накопление

Задача **префиксного накопления**

Дан массив $a = [a_0, \dots, a_{n-1}]$

Вычислить $b_{-1} = 0 \quad b_i = a_i + b_{i-1} \quad 0 \leq i < n$

В более общем виде: ассоциативный оператор \bullet (предполагается наличие единицы ϵ , может быть добавлена формально)

Вычислить $b_{-1} = \epsilon \quad b_i = a_i \bullet b_{i-1} \quad 0 \leq i < n$

$$b_0 = a_0$$

$$b_1 = a_0 \bullet a_1$$

$$b_2 = a_0 \bullet a_1 \bullet a_2$$

...

$$b_{n-1} = a_0 \bullet a_1 \bullet \dots \bullet a_{n-1}$$

Последовательная сложность $O(n)$ при помощи тривиальной схемы размера $n - 1$, глубины $n - 1$

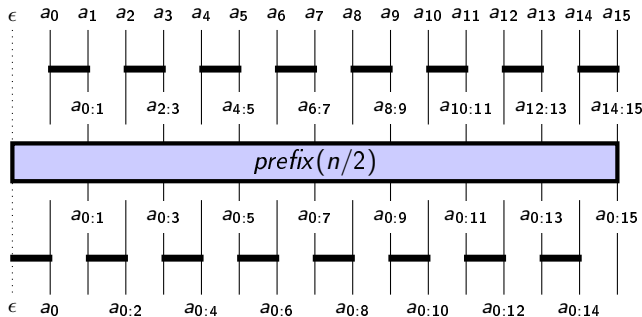
Параллельные алгоритмы

Префиксное накопление

Схема префиксного накопления

[Ladner, Fischer: 1980]

$prefix(n)$



где $a_{k:l} = a_k \bullet a_{k+1} \bullet \dots \bullet a_l$

Параллельные алгоритмы

Префиксное накопление

Схема **префиксного накопления** (продолжение)

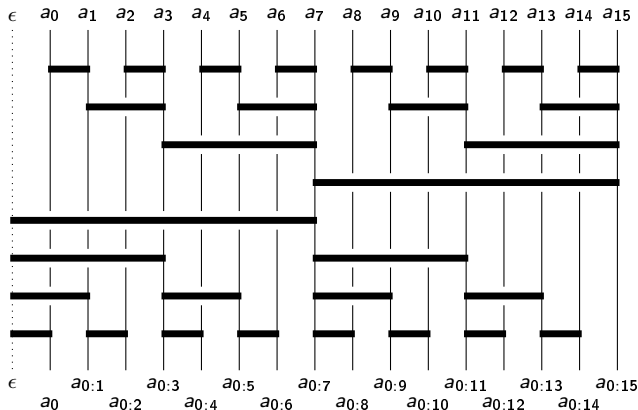
$prefix(n)$

n входов

n выходов

размер $2n - 2$

глубина $2 \log n$



Сложность на PRAM $O(\log n)$

Префиксное накопление на BSP

Граф $prefix(n)$ состоит из

- верхнего поддереза, вычисляемого от листьев к корню $tree(n)$
- переноса значений от узлов верхнего поддереза к узлам нижнего
- нижнего поддереза, вычисляемого от корня к листьям $tree(n)$

Параллельные алгоритмы

Префиксное накопление

Префиксное накопление на BSP

Граф $prefix(n)$ состоит из

- верхнего поддереза, вычисляемого от листьев к корню $tree(n)$
- переноса значений от узлов верхнего поддереза к узлам нижнего
- нижнего поддереза, вычисляемого от корня к листьям $tree(n)$

Оба поддереза можно вычислить предыдущим алгоритмом

Перенос значений: $comm = O(n/p)$ — на самом деле не обязательно; промежуточные значения можно запомнить; каждое нижнее поддерево вычислять в том же процессоре, что и соответствующее верхнее

Параллельные алгоритмы

Префиксное накопление

Префиксное накопление на BSP

Граф $prefix(n)$ состоит из

- верхнего поддереза, вычисляемого от листьев к корню $tree(n)$
- переноса значений от узлов верхнего поддереза к узлам нижнего
- нижнего поддереза, вычисляемого от корня к листьям $tree(n)$

Оба поддереза можно вычислить предыдущим алгоритмом

Перенос значений: $comm = O(n/p)$ — на самом деле не обязательно; промежуточные значения можно запомнить; каждое нижнее поддерево вычислять в том же процессоре, что и соответствующее верхнее

$$comp = O(n/p)$$

$$comm = O(n/p)$$

$$sync = O(1)$$

Предполагаем допуск $n \geq p^2$

Параллельные алгоритмы

Линейные рекуррентные соотношения

Обобщенное линейное рекуррентное соотношение (первого порядка)

Даны массивы $a = [a_0, \dots, a_{n-1}]$, $b = [b_0, \dots, b_{n-1}]$

Вычислить $c_{-1} = 0$ $c_i = a_i + b_i \cdot c_{i-1}$ $0 \leq i < n$

$$c_0 = a_0$$

$$c_1 = a_1 + b_1 \cdot c_0$$

$$c_2 = a_2 + b_2 \cdot c_1$$

...

$$c_{n-1} = a_{n-1} + b_{n-1} \cdot c_{n-2}$$

Параллельные алгоритмы

Линейные рекуррентные соотношения

$$c_{-1} = 0 \quad c_i = a_i + b_i \cdot c_{i-1} \quad 0 \leq i < n$$

$$\text{Let } A_i = \begin{bmatrix} 1 & 0 \\ a_i & b_i \end{bmatrix} \quad C_i = \begin{bmatrix} 1 \\ c_i \end{bmatrix} \quad A_i C_{i-1} = \begin{bmatrix} 1 & 0 \\ a_i & b_i \end{bmatrix} \begin{bmatrix} 1 \\ c_{i-1} \end{bmatrix} = \begin{bmatrix} 1 \\ c_i \end{bmatrix} = C_i$$

$$C_0 = A_0 \cdot C_{-1}$$

$$C_1 = A_1 A_0 \cdot C_{-1}$$

$$C_2 = A_2 A_1 A_0 \cdot C_{-1}$$

...

$$C_{n-1} = A_{n-1} \dots A_1 A_0 \cdot C_{-1}$$

Параллельные алгоритмы

Линейные рекуррентные соотношения

Вычисление обобщенного линейного рекуррентного соотношения

- суффиксное накопление (= префиксное накопление в обратном направлении) на $[A_{n-1}, \dots, A_0]$, с оператором умножения 2×2 -матриц
- умножаем каждый выход накопления на C_{-1}
- берем в качестве выхода нижнюю компоненту получившихся 2-векторов

У полученной схемы размер $O(n)$, глубина $O(\log n)$

Параллельные алгоритмы

Линейные рекуррентные соотношения

Операторы $+$, \cdot можно заменить на любые \oplus , \odot , где

- операторы \oplus , \odot вычисляются за время $O(1)$
- оператор \oplus ассоциативен: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
- операторы \odot ассоциативен: $a \odot (b \odot c) = (a \odot b) \odot c$
- оператор \odot дистрибутивен (слева) над \oplus :
$$a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$$

Примеры возможных \oplus , \odot :

- численные $+$, \cdot
- численные \min , $+$; численные \max , $+$
- булевы \wedge , \vee ; булевы \vee , \wedge ; булевы \wedge , xor

Параллельные алгоритмы

Двоичное сложение

Параллельное **двоичное сложение** при помощи булевой логики

$x + y = z$ x, y, z представлены булевыми массивами

$$x = [x_{n-1}, \dots, x_0] \quad y = [y_{n-1}, \dots, y_0] \quad z = [z_n, z_{n-1}, \dots, z_0]$$

По заданным x, y , вычислить z

Примитивные операторы — булевы \wedge (“and”), \vee (“or”), \oplus (“xor”)

Пусть $c = [c_{n-1}, \dots, c_0]$, где c_i — это i -й бит переноса

$$\text{Имеем: } x_i + y_i + c_{i-1} = z_i + 2c_i \quad 0 \leq i < n$$

Параллельные алгоритмы

Двоичное сложение

Определим битовые массивы $u = [u_{n-1}, \dots, u_0]$, $v = [v_{n-1}, \dots, v_0]$

$$u_i = x_i \wedge y_i \quad v_i = x_i \oplus y_i \quad 0 \leq i < n$$

$$z_0 = v_0 \qquad c_0 = u_0$$

$$z_1 = v_1 \oplus c_0 \qquad c_1 = u_1 \vee (v_1 \wedge c_0)$$

$$\dots \qquad \dots$$

$$z_{n-1} = v_{n-1} \oplus c_{n-2} \qquad c_{n-1} = u_{n-1} \vee (v_{n-1} \wedge c_{n-2})$$

$$z_n = c_{n-1}$$

Параллельные алгоритмы

Двоичное сложение

Определим битовые массивы $u = [u_{n-1}, \dots, u_0]$, $v = [v_{n-1}, \dots, v_0]$

$$u_i = x_i \wedge y_i \quad v_i = x_i \oplus y_i \quad 0 \leq i < n$$

$$z_0 = v_0 \qquad c_0 = u_0$$

$$z_1 = v_1 \oplus c_0 \qquad c_1 = u_1 \vee (v_1 \wedge c_0)$$

$$\dots \qquad \dots$$

$$z_{n-1} = v_{n-1} \oplus c_{n-2} \qquad c_{n-1} = u_{n-1} \vee (v_{n-1} \wedge c_{n-2})$$

$$z_n = c_{n-1}$$

Получилась схема **сумматора со сквозным переносом** размера и глубины $O(n)$

Можно ли ее распараллелить?

Параллельные алгоритмы

Двоичное сложение

$$c_{-1} = 0 \quad c_i = u_i \vee (v_i \wedge c_{i-1})$$

Вычисляем

- массив c при помощи обобщенного линейного рекуррентного соотношения со входами u , v и операторами \vee , \wedge : размер $O(n)$, глубина $O(\log n)$
- массив z : дополнительный размер $O(n)$, глубина $O(1)$

Получилась схема **сумматора с ускоренным переносом** размера $O(n)$, глубины $O(\log n)$

- 1 Дискретное преобразование Фурье и его применения
- 2 Параллельные алгоритмы
- 3 Оптимизационные задачи и приближенные алгоритмы

Оптимизационные задачи и приближенные алгоритмы

Задача линейного программирования

Задача **линейного программирования** (LP): оптимизировать (максимизировать или минимизировать) многомерную вещественную линейную **целевую функцию** при линейных **ограничениях** (неравенствах или равенствах)

$$\max(\min) \quad c_1x_1 + c_2x_2 + \dots + c_nx_n$$

при условиях

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \geq b_2$$

$$a_{31}x_1 + a_{32}x_2 + \dots + a_{3n}x_n = b_3$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

Оптимизационные задачи и приближенные алгоритмы

Задача линейного программирования

Вектор $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ — **допустимое решение** задачи LP, если он удовлетворяет всем ограничениям

Задача LP **разрешима**, если у нее есть допустимое решение

Допустимое решение **оптимально**, если целевая функция достигает на нем (нестромого) максимума (соответственно, минимума)

Многие теоретические и практические задачи имеют вид LP или приводятся к нему. Например, имеется m материалов, в наличии b_i единиц материала i . Возможны n продуктов, для производства единицы продукта j нужно a_{ij} единиц материала i . Единица продукта j дает прибыль c_j . Как максимизировать прибыль?

Стандартная форма задачи LP

$$\max \quad c_1x_1 + c_2x_2 + \dots + c_nx_n$$

при условиях

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_1, x_2, \dots, x_n \geq 0$$

n переменных, $m + n$ ограничений, включая n ограничений неотрицательности (будем их опускать, предполагая по умолчанию)

$$\max \quad c^T x$$

при условиях

$$Ax \leq b$$

$$x \geq 0$$

Оптимизационные задачи и приближенные алгоритмы

Стандартная форма

Приведение задачи LP к стандартной форме

$$\min_{1 \leq j \leq n} c_j x_j \quad \rightsquigarrow \quad \max_{1 \leq j \leq n} -c_j x_j$$

при условиях

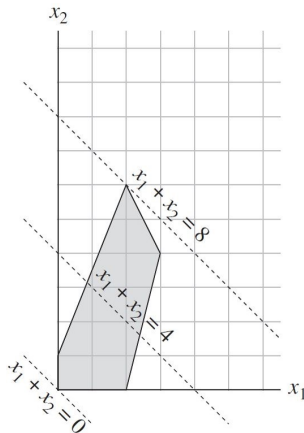
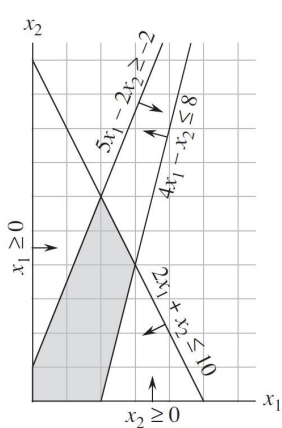
$$\sum_{1 \leq j \leq n} a_{ij} x_j \geq b_i \quad \rightsquigarrow \quad \sum_{1 \leq j \leq n} -a_{ij} x_j \leq -b_i$$

$$\sum_{1 \leq j \leq n} a_{ij} x_j = b_i \quad \rightsquigarrow \quad \sum_{1 \leq j \leq n} a_{ij} x_j \geq b_i \wedge \sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i$$

$$x_j \text{ неограничено} \quad \rightsquigarrow \quad x_j = x_j^+ - x_j^- \wedge x_j^+, x_j^- \geq 0$$

Оптимизационные задачи и приближенные алгоритмы

Стандартная форма



Оптимизационные задачи и приближенные алгоритмы

Стандартная форма

Пусть $P \subseteq \mathbb{R}^n$

P — **выпуклое множество**, если для любых $x, y \in P$, $0 \leq \alpha \leq 1$, имеем $\alpha x + (1 - \alpha)y \in P$

$u \in P$ — **экстремальная точка**, если не существует $x, y \in P$, $0 < \alpha < 1$, для которых $u = \alpha x + (1 - \alpha)y$

Линейное неравенство с n переменными задает **полупространство**.
Пересечение конечного множества полупространств — **многогранник**.

Многогранник — выпуклое множество; его экстремальные точки — **вершины**. Многогранник, заданный ограничениями задачи LP — **допустимый многогранник**.

Задача LP **ограничена**, если $\sum_{1 \leq i \leq n} c_i x_i \leq B$ для некоторого $B \in \mathbb{R}$ и любого допустимого x (в случае максимизации, иначе \geq)

Каноническая форма задачи LP

$$\max \quad c_1 x_1 + c_2 x_2 + \dots + c_n x_n$$

при условиях

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

$$x_1, x_2, \dots, x_n \geq 0$$

$$\max \quad c^T x$$

при условиях

$$Ax = b$$

$$x \geq 0$$

Как правило, $m \geq n$, строки A линейно независимы

Оптимизационные задачи и приближенные алгоритмы

Каноническая форма

Приведение задачи LP от стандартной формы к канонической

Вводятся **вспомогательные переменные** $s_i = x_{n+i}$

$$\max \quad \sum_{1 \leq j \leq n} c_j x_j$$

при условиях

$$\sum_{1 \leq j \leq n} a_{ij} x_j \geq b_i \quad \rightsquigarrow s_i = b_i - \sum_{1 \leq j \leq n} a_{ij} x_j$$

$$x_j \geq 0 \quad \rightsquigarrow x_j \geq 0 \quad s_i \geq 0$$

$$\max \quad c^T x$$

при условиях

$$s = b - Ax$$

$$x, s \geq 0$$

Основная теорема линейного программирования Задача LP в стандартной форме имеет оптимальное решение, если она разрешима и ограничена. Если задача LP имеет оптимальные решения, то хотя бы одно из них — вершина допустимого многогранника.

(Без доказательства; одно из конструктивных доказательств — симплекс-метод)

Теоретически, для решения задачи LP достаточно перебрать все вершины допустимого многогранника — это конечное множество. Однако количество вершин экспоненциально относительно количества переменных и ограничений (например, n -мерный гиперкуб).

Симплекс-метод: последовательные перемещение по вершинам допустимого многогранника, улучшающие целевую функцию

Основная идея симплекс-метода:

- берем одну из вершин в качестве начального решения
- если решение не оптимально, но оптимальное решение существует, то в какой-то из соседних вершин значение целевой функции лучше, чем в текущей; перемещаемся в эту вершину и повторяем

В теории, симплекс-метод также экспоненциален в худшем случае. На практике, работает вполне эффективно.

Полиномиальные алгоритмы для решения задачи LP: метод эллипсоидов, метод внутренней точки. Применяются на практике, но не так широко, как симплекс-метод.

Оптимизационные задачи и приближенные алгоритмы

Двойственность

$$\max \quad c_1x_1 + c_2x_2 + \dots + c_nx_n$$

при условиях

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m$$

$$x_1, x_2, \dots, x_n \geq 0$$

Сокращенная форма:

$$\max \quad c^T x$$

$$\text{при условиях } Ax \leq b \quad x \geq 0$$

$$\min \quad b_1y_1 + b_2y_2 + \dots + b_my_m$$

при условиях

$$a_{11}y_1 + a_{21}y_2 + \dots + a_{m1}y_m \geq c_1$$

$$a_{12}y_1 + a_{22}y_2 + \dots + a_{m2}y_m \geq c_2$$

...

$$a_{1n}y_1 + a_{2n}y_2 + \dots + a_{mn}y_m \geq c_n$$

$$y_1, y_2, \dots, y_m \geq 0$$

Сокращенная форма:

$$\min \quad b^T y$$

$$\text{при условиях } A^T y \geq c \quad y \geq 0$$

Теорема о двойственности Если первичная задача разрешима, то двойственная также разрешима, и оптимальные решения равны
(Без доказательства)

Экономический смысл двойственности: материал i расходуется для изготовления единицы продукта j в количестве a_{ij}

Первичная задача:

- имеются материалы в заданных количествах b_i
- продукты изготавливаются в количествах x_j и продаются по ценам c_j
- цель — max прибыль (сумму по продуктам с учетом цен)

Двойственная задача:

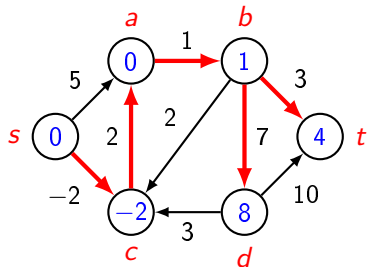
- материалы покупаются в количествах y_i по ценам b_i
- нужно изготовить заданное количество каждого продукта c_j ,
- цель — min стоимость (сумму по материалам с учетом цен)

Описывают одну и ту же экономическую ситуацию с разных сторон

Оптимизационные задачи и приближенные алгоритмы

Задача о кратчайших путях как задача LP

$G = (V, E, w)$ — ориентированный граф с весами на ребрах, без отрицательных циклов



$s \in V$: источник

$w : E \rightarrow \mathbb{R}$: стоимость ребра

$\delta(u) = \delta(s, u)$: расстояние от источника s до $u \in V$

Пусть каждая вершина достижима из s : $\delta(u) < \infty$ для всех $u \in V$

Оптимизационные задачи и приближенные алгоритмы

Задача о кратчайших путях как задача LP

Лемма Если $y_s = 0$ и $y_v \leq y_u + w(u, v)$ для всех $(u, v) \in E$, тогда $y_u \leq \delta(u)$ для всех $u \in V$.

Если вдобавок для каждой $v \in V \setminus \{s\}$ существует $(u, v) \in E$ такое, что $y_v = y_u + w(u, v)$, тогда $y_u = \delta(u)$ для всех $u \in V$.

Доказательство (Упражнение) □

Задача о кратчайших путях с одним источником в форме задачи LP

$$\max \sum_{u \in V} y_u$$

при условиях

$$y_v - y_u \leq w(u, v) \quad (u, v) \in E$$

$$y_s = 0$$

Теорема Эта задача LP разрешима. Пусть \bar{y} — оптимальное решение. Тогда $\bar{y}_u = \delta(u)$ для всех $u \in V$.

Доказательство $\delta(u)$ — допустимое решение:

- $\delta(s) = 0$
- для всех $u \in V$, $\delta(u) \geq 0$
- для любого $(u, v) \in E$, $\delta(v) \leq \delta(u) + w(u, v)$ по неравенству треугольника

Пусть y — произвольное допустимое решение

По первой части леммы, $y_u \leq \delta(u)$ для всех $u \in V$

Следовательно, $\bar{y}_u = \delta(u)$ — единственное оптимальное решение.

Оптимизационные задачи и приближенные алгоритмы

Задача о кратчайших путях как задача LP

Проверим также вторую часть леммы: докажем, что для каждой $v \in V \setminus \{s\}$ существует $(u, v) \in E$ такое, что $\bar{y}_v = \bar{y}_u + w(u, v)$.

Предположим противное: $\bar{y}_v < \bar{y}_u + w(u, v)$ для некоторой $v \in V$ и всех $(u, v) \in E$

Тогда можем увеличить u , не нарушая ограничения и увеличивая целевую функцию. Противоречие с оптимальностью \bar{u} .

По второй части леммы, снова имеем $\bar{y}_u \leq \delta(u)$ для всех $u \in V$



Оптимизационные задачи и приближенные алгоритмы

Задача о кратчайших путях как задача LP

Если у G есть отрицательный цикл, задача неразрешима

Если у G есть вершина, недостижимая из S , задача неограничена

Задачу можно рассматривать, как физическую модель G : вершины u , v соединены нерастяжимой (но сжимаемой) веревкой длины $w(u, v)$

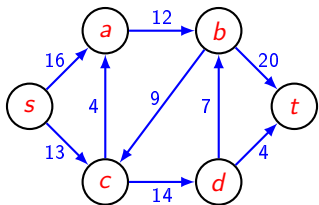
Задача LP не дает эффективного алгоритма для поиска кратчайших путей — есть более эффективные алгоритмы

Однако формулировка в виде задачи LP устойчива к изменению условий задачи: например, **стохастические кратчайшие пути**, **марковские процессы принятия решений**

Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP

$G = (V, E, c)$ — ориентированный граф с пропускными способностями на ребрах



$s, t \in V$: источник, сток

$c : E \rightarrow \mathbb{R}^+$: **пропускная способность** ребра

$f : E \rightarrow \mathbb{R}^+$: **поток** вдоль ребра,
 $0 \leq f(u) \leq c(u)$

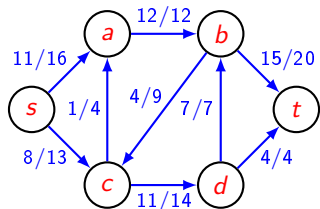
Сохранение потока: для любого $u \in V \setminus \{s, t\}$ суммарный входящий поток в u должен равняться суммарному исходящему потоку из u

Моделирует потоки жидкостей по трубам, грузов по дорогам и т.п.

Поток считается стационарным, не меняется со временем

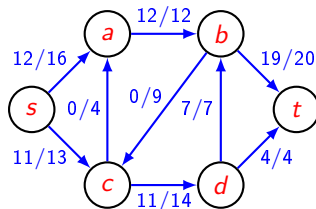
Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP



Суммарный поток

$$19 = 11 + 8 = 15 + 4$$



Суммарный поток

$$23 = 12 + 11 = 19 + 4$$

Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP

Задача о максимальном потоке в форме задачи LP

$$\max |f| = \sum_{u:(u,t) \in E} f_{ut} - \sum_{w:(t,w) \in E} f_{tw}$$

при условиях

$$\sum_{u:(u,v) \in E} f_{uv} - \sum_{w:(v,w) \in E} f_{vw} = 0 \quad v \in V \setminus \{s, t\}$$

$$f_{uv} \leq c_{uv} \quad (u, v) \in E$$

$$f_{uv} \geq 0 \quad (u, v) \in E$$

Задача разрешима и ограничена, следовательно, имеет оптимальное решение

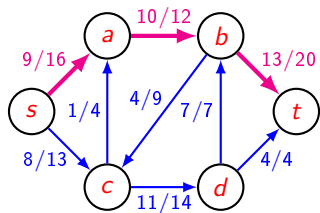
Ребро (u, v) **насыщено**, если $f_{uv} = c_{uv}$

(s, t) -разрез: Разбиение $V = S \cup T$, где $s \in S$, $t \in T$

Значение (s, t) -разреза: $c(S, T) = \sum_{(u,v) \in E \cap (S \times T)} c_{uv}$

Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP

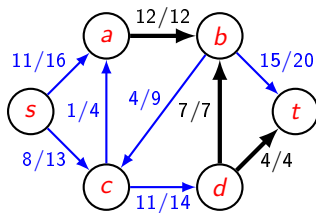


Суммарный поток = 17

Прямой увеличивающий путь: путь из s в t , где на всех ребрах поток ненасыщен

Блокирующий поток: имеется (s, t) разрез, где все ребра из компоненты s в компоненту t насыщены

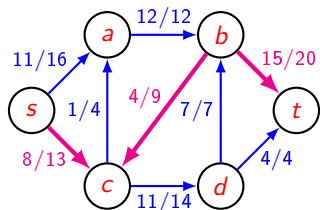
При блокирующем потоке прямой увеличивающий путь отсутствует, однако поток может не быть максимальным



Блокирующий поток = 19

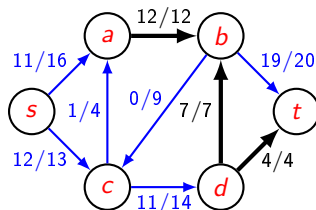
Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP



Суммарный поток = 19

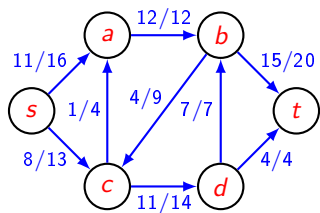
Увеличивающий путь: путь из s в t по прямым и обратным ребрам, где на всех прямых ребрах поток ненасыщен, а на всех обратных ребрах поток ненулевой



Максимальный поток = 23

Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP

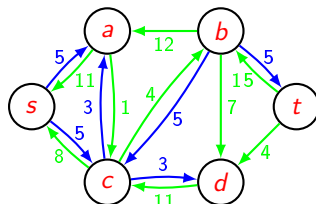


Произвольный поток

Для простоты предполагаем, что в исходном графе нет параллельных противонаправленных ребер

Остаточная сеть содержит для каждого $(u, v) \in E$:

- прямое ребро веса $c_{uv} - f_{uv}$
- обратное ребро веса f_{uv}

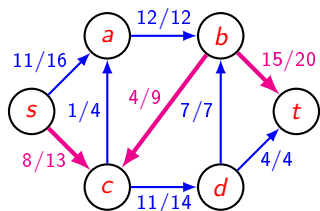


Остаточная сеть

Ребра нулевого веса в остаточную сеть не включаются

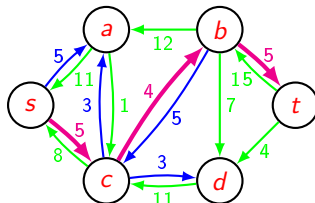
Оптимизационные задачи и приближенные алгоритмы

Задача о максимальном потоке как задача LP



Увеличивающий путь

Значение увеличивающего пути — минимальный вес по всем ребрам пути в остаточной сети



Увеличивающий путь в остаточной сети

Теорема Следующие утверждения равносильны:

- ❶ f — максимальный поток
- ❷ f не имеет увеличивающих путей
- ❸ существует (s, t) -разрез со значением $c(S, T) = |f|$

Доказательство $(1) \Rightarrow (2)$, $(3) \Rightarrow (1)$ — очевидно

$(2) \Rightarrow (3)$: Пусть S — множество вершин остаточной сети, достижимых из s ; $T = V \setminus S$

Доказательство (окончание):

Поскольку f не имеет увеличивающих путей, имеем

- $s \in S, t \in T$
- $f_{uv} = c_{uv}$ при $(u, v) \in E \cap (S \times T)$
- $f_{uv} = 0$ при $(u, v) \in E \cap (T \times S)$

$$\begin{aligned} c(S, T) &= \sum_{(u,v) \in E \cap (S \times T)} c_{uv} = \\ &= \sum_{(u,v) \in E \cap (S \times T)} c_{uv} + \sum_{(u,v) \in E \cap (T \times S)} 0 = \\ &= \sum_{(u,v) \in E \cap (S \times T)} f_{uv} + \sum_{(u,v) \in E \cap (T \times S)} f_{uv} = |f| \end{aligned}$$



Замечание Максимальный поток и минимальный разрез — оптимальные решения взаимно двойственных задач LP.

Альтернативное доказательство теоремы следует из общей теории двойственности.

Алгоритм Форда–Фалкерсона для нахождения максимального потока

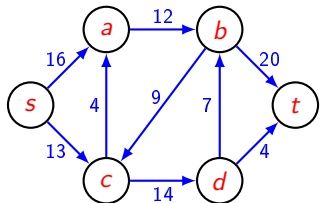
Начинаем с произвольного допустимого потока (например, нулевого), затем

- Строим для текущего потока остаточную сеть
- Ищем увеличивающий путь в остаточной сети. Если его нет, поток максимальный.
- Включаем увеличивающий путь в поток, добавляя его значение к прямым ребрам и вычитая из обратных
- Повторяем для нового потока

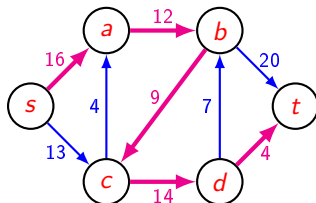
Поиск увеличивающего пути: поиск в графе в ширину (находит кратчайший такой путь) или в глубину

Оптимизационные задачи и приближенные алгоритмы

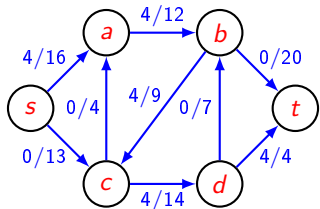
Алгоритм Форда–Фалкерсона



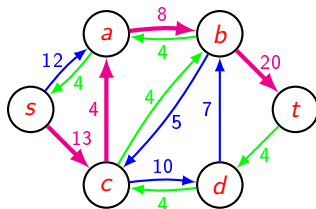
Суммарный поток = 0



Увеличивающий путь: +4



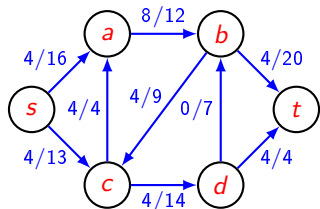
Суммарный поток = 4



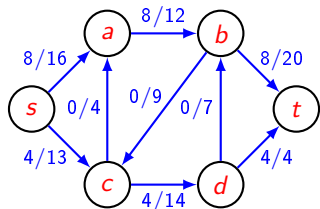
Увеличивающий путь: +4

Оптимизационные задачи и приближенные алгоритмы

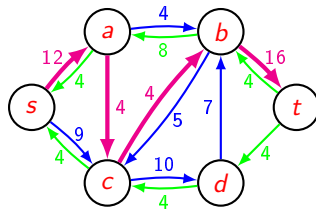
Алгоритм Форда–Фалкерсона



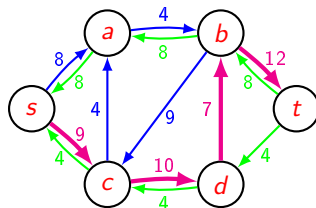
Суммарный поток = 8



Суммарный поток = 12



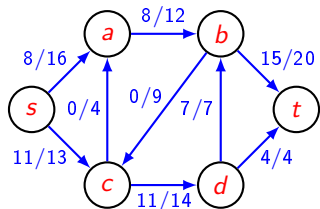
Увеличивающий путь: +4



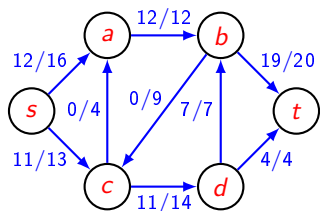
Увеличивающий путь: +7

Оптимизационные задачи и приближенные алгоритмы

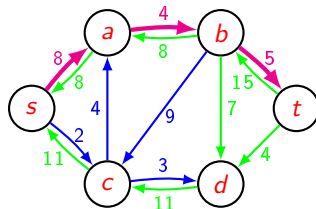
Алгоритм Форда–Фалкерсона



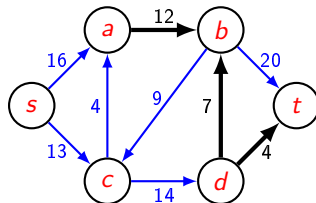
Суммарный поток = 19



Максимальный поток = 23



Увеличивающий путь: +4



Минимальный разрез = 23

Оптимизационные задачи и приближенные алгоритмы

Алгоритм Форда–Фалкерсона

Целочисленная задача о максимальном потоке: все пропускные способности и значения начального потока — натуральные числа

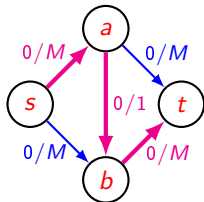
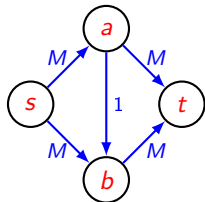
Теорема В целочисленной задаче о максимальном потоке значение оптимального потока $|f^*|$ — натуральное число. Алгоритм Форда–Фалкерсона находит его за $|f^*|$ итераций. Общая трудоемкость алгоритма $|E||f^*|$.

Доказательство На каждой итерации, все веса в остаточном графе — натуральные; значение увеличивающего пути — натуральное > 0 . \square

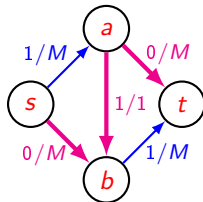
Замечание Количество итераций $|f^*|$ может быть экспоненциальным от размера входа!

Оптимизационные задачи и приближенные алгоритмы

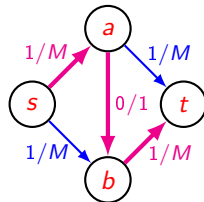
Алгоритм Форда–Фалкерсона



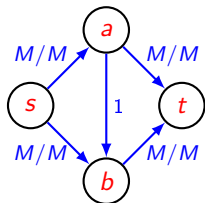
Поток = 0
Ув. путь: +1



Поток = 1
Ув. путь: +1



Поток = 2
Ув. путь: +1



Макс. поток = $2M$

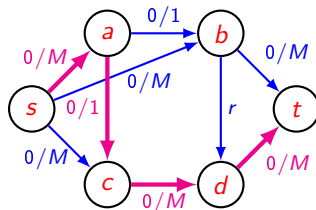
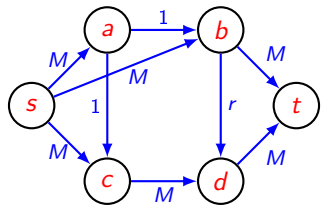
Увеличивающие пути далее повторяются с периодом 2; вследствие неудачного выбора увеличивающих путей, алгоритм выполняет $2M$ итераций

Оптимизационные задачи и приближенные алгоритмы

Алгоритм Форда–Фалкерсона

Если хотя бы одна пропускная способность — не натуральное число, алгоритм может не завершиться; предельное значение потока может быть не оптимальным (сколь угодно далеким от оптимального)

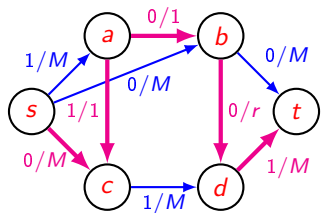
Пусть $r = 1 - r^2 = \frac{\sqrt{5}-1}{2}$ (золотое сечение); $M \geq 4$



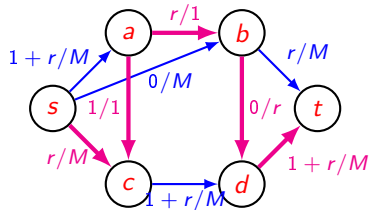
Поток = 0; ув. путь: +1

Оптимизационные задачи и приближенные алгоритмы

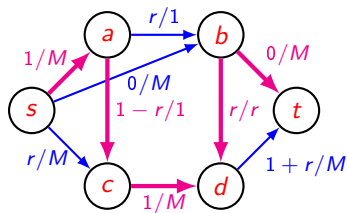
Алгоритм Форда–Фалкерсона



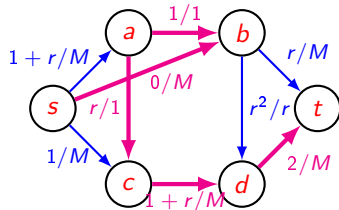
Поток = 1; ув. путь: $+r$



Поток = $1 + 2r$; у.п.: $+1 - r = +r^2$



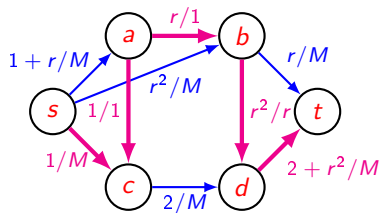
Поток = $1 + r$; ув. путь: $+r$



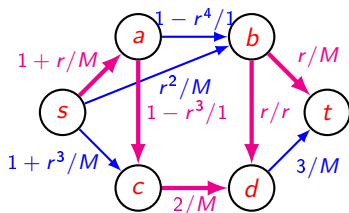
Поток = $2 + r$; у.п.: $+1 - r = +r^2$

Оптимизационные задачи и приближенные алгоритмы

Алгоритм Форда–Фалкерсона



Поток = 3; у.п.: $+r - r^2 = +r^3$



Поток = $3 + r^3$; у.п.: $+1 - 1 + r^3 = +r^3$

Увеличивающие пути далее повторяются с периодом 4; вследствие неудачного выбора увеличивающих путей, алгоритм не завершается

Предельный поток $1 + 2 \sum_{k \geq 1} r^k = 3 + 2r < 5$

Оптимальный поток $2M$

Оптимизационные задачи и приближенные алгоритмы

Задача о минимальном вершинном покрытии как задача LP

$G = (V, E)$ — неориентированный граф

$S \subseteq V$ — **вершинное покрытие**, если для любого $(u, v) \in E$, имеем $u \in S$ или $v \in S$

Задача о минимальном вершинном покрытии — не имеет известного полиномиального алгоритма; NP-полна (будет доказано в курсе сложности вычислений)

Формулировка в виде задачи LP позволяет получить

2-аппроксимацию, т.е. решение, которое не более, чем в 2 раза превосходит оптимальное, за полиномиальное время

Оптимизационные задачи и приближенные алгоритмы

Задача о минимальном вершинном покрытии как задача LP

Задача о минимальном вершинном покрытии в форме задачи LP

$$\min \sum_{v \in V} x_v$$

при условиях

$$x_u + x_v \geq 1 \quad (u, v) \in E$$

$$0 \leq x_v \leq 1 \quad v \in V$$

Для решения исходной задаче необходимо условие целочисленности: $x_v \in \{0, 1\}$. Без условия целочисленности задача LP является **релаксацией** исходной задачи.

Задача разрешима и ограничена, следовательно, имеет оптимальное решение. Очевидно, оно не превосходит оптимального решения исходной задачи (и равно ему, если соответствующие значения переменных натуральные).

Оптимизационные задачи и приближенные алгоритмы

Задача о минимальном вершинном покрытии как задача LP

Пусть x^* — значения переменных в некотором оптимальном решении LP-релаксации

Определим $S_{LP} = \{v \in V : x_v^* \geq \frac{1}{2}\}$

Пусть S_{OPT} — некоторое минимальное вершинное покрытие, \bar{x} — значения переменных в соответствующем оптимальном решении с дополнительным ограничением $x_v \in \{0, 1\}$, т.е. $\bar{x}_v = 1$, когда $v \in S_{OPT}$

Оптимизационные задачи и приближенные алгоритмы

Задача о минимальном вершинном покрытии как задача LP

Утверждение S_{LP} — вершинное покрытие, $|S_{LP}| \leq 2|S_{OPT}|$

Доказательство

Для любого $(u, v) \in E$, имеем $x_u^*, x_v^* \geq 0$, $x_u^* + x_v^* \geq 1$. Следовательно $x_u^* \geq \frac{1}{2}$ или $x_v^* \geq \frac{1}{2}$, таким образом $u \in S_{LP}$ или $v \in S_{LP}$, то есть S_{LP} — вершинное покрытие.

$$|S_{LP}| = \sum_{v \in S_{LP}} 1 \leq$$

$$(\text{поскольку } x_v^* \geq \frac{1}{2} \text{ для } v \in S_{LP}) \sum_{v \in S_{LP}} 2x_v^* \leq$$

$$(\text{поскольку } S_{LP} \subseteq V) \sum_{v \in V} 2x_v^* \leq$$

$$(\text{поскольку допустимые решения для исходной задачи допустимы и для LP-релаксации}) \sum_{v \in V} 2\bar{x}_v = 2|S_{OPT}|$$



Таким образом, решение LP-релаксации дает 2-аппроксимацию для задачи минимального вершинного покрытия