Лекция 6

Квантовые алгоритмы

(Конспект: И. Посов)

6.1 Квантовые вычисления

Будем изучать класс сложности **BQP** — языков, принимаемых полиномиальными квантовыми алгоритмами с ограниченной двусторонней ошибкой. На рисунке 6.1 видно его расположение по отношению к другим классам сложности. Скорее всего, $\mathbf{BQP} \neq \mathbf{PP}$. Из обратного ($\mathbf{BQP} =$

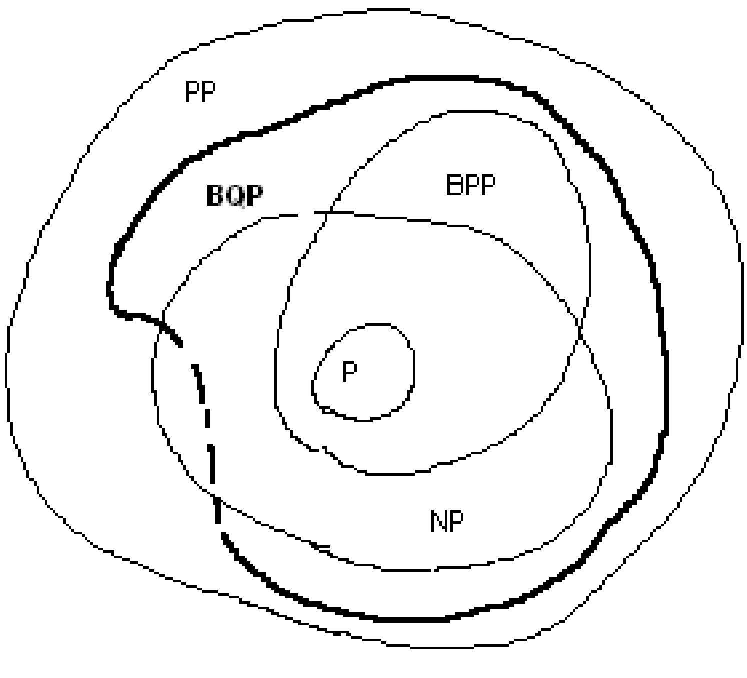

Рис. 6.1: место **BQP** среди других классов

PP) следует, что $\mathbf{PH} \subseteq \mathbf{PP}$. Также неизвестно, содержится ли **NP** в **BQP**.

Вспомним, что такое детерминированные классические вычисления. У нас есть машина Тьюринга и какой-то фиксированный вход (лента с конкретными символами). По внутреннему состоянию машины и текущему символу с ленты однозначно определено следующее внутреннее состояние, выходной символ и изменение положения считывающей/пишущей головки. Но в данном конспекте будем понимать это несколько иначе. Головки не будет, но при этом мы разрешим за один раз считывать и записывать информацию сразу из нескольких мест. Важно, чтобы этих мест было не очень много, т.е. не больше некоторой константы. Все сказанное означает, что у нас есть вычисляющая машина, которая за шаг смотрит на некоторое число ячеек на ленте и в зависимости от своего внутреннего состояния изменяет эти ячейки и свое внутреннее состояние. Можно легко показать, что такая машина «эквивалентна» машине Тьюринга, т.е. они друг друга полиномиально моделируют.

Если машина работает полиномальное время, то она использует лишь полиномиальное число ячеек на ленте. Внутренних состояний — константное число, так что можно считать, что полиномиальным (от входа) числом битов можно записать полное состояние машины. Каждая такая конфигурация (состояние ленты и внутреннее состояние) однозначно определяет следующую конфигурацию машины. Изобразим это в виде матрицы C из 0 и 1 таким образом, что в позиции C_{c_j,c_i} стоит 1 тогда и только тогда, когда из конфигурации c_i можно за один шаг перейти в c_j. Таким образом, в каждом столбце стоит ровно одна 1.

Теперь введем вектор конфигураций. Это столбец из 1 и 0, в котором ровно одна 1, положение которой сответствует той конфигурации, в котором находится машина. Например, в двумерном случае столбец $(1,0,0,0)^t$ означает, что она находится в конфигурации 00, столбец $(0,1,0,0)^t$ означает конфигурацию 01, столбец $(0,0,1,0)^t$ означает конфигурацию 10 и, наконец, столбец $(0,0,0,1)^t$ означает конфигурацию 11. Что будет если матрицу C умножить на вектор конфигураций? Это эквивалентно одному шагу машины, т.е. 1 переместится из одного места в другое, что соответствует переходу машины из состояния в состояние. В результате получается, что машина работает таким образом: изначально есть вектор конфигураций, соответствующий начальной конфигурации машины. За один шаг вектор умножается на матрицу C.

В дальнейшем мы будем путать термины «состояние» и «конфигурация» (при этом всегда будем иметь в виду последнее).

В вероятностных вычислениях все практически также. Только здесь для состояния известно не то, в какое конкретно состояние переходит машина, а определены вероятности перехода из данного состояния во все другие состояния. Зададим матрицу C так, что в позиции c_j, c_i стоит вероятность перехода из состояния c_i в состояние c_j. Сумма элементов в столбцах должна будет оказаться 1. Вектор состояний — это вектор, где в позиции c_i находится вероятность нахождения машины в состоянии c_i. Изначально вектор таков, что в позиции, соответствующей начальному состоянию стоит 1, а во всех остальных местах — 0. Это означает, что в начальный момент времени машина наверняка находится в начальном состоянии. Шаг вычисления — это умножение вектора справа на матрицу C. Очевидно, что после k умножений в позиции c_i вектора будет находится вероятность того, что через k шагов после начала машина окажется в состоянии c_i. Детерминированные вычисления являются частным случаем вероятностных.

И, наконец, квантовые вычисления. Здесь все полностью повторяет предыдущее, но элементы матрицы C теперь уже будут комплексными. Элементы вектора состояний — это не вероятности соответствующих состояний, а их амплитуды. Амплитуда — это как бы корень из вероятности (если мы прервем вычисление и посмотрим, что же получилось, то мы увидим конфигурацию c_i с вероятностью $1/|a_i|^2$, где a_i — амплитуда конфигурации c_i). Таким образом, вектор состояний всегда должен иметь единичную длину. (Сумма квадратов элементов равна 1) Столбцы матрицы C тоже должны иметь единичную длину, чтобы после умножения ее на вектор, он оставался единичной длины. Таким образом, матрица C имеет столбцы единичной длины, сохраняет длины векторов и, тем самым, является унитарным оператором. Заметим также, что матрица C должна быть обратима.

Как получена матрица C? Так же, как и матрица для детерминированного или вероятностного вычисления: именно, за один шаг она должна воздействовать на амплитуды лишь константного числа битов, при этом ее действие также должно зависеть лишь от константного числа битов (на разных шагах алгоритма — от разных!). Например, ее можно построить так (и мы будем этим пользоваться): взять конкретные матрицы C_i, воздействующие на конкретные маленькие множества позиций конфигурации, и построить их произведение. Чтобы это произведение реализовать в вышеприведенной модели, понадобится дополнительная память: счетчик шагов (наша машина должна знать, каков номер шага, чтобы знать, какую из C_i применять). Тем самым, если выделить столбцы и строки матрицы C, соответствующие шагам i и $i+1$, и переупорядочить оставшуюся память так, чтобы задействованные в C_i биты находились в конце конфигурации, получится матрица

$$\begin{array}{ccccc} & & i & i+1 \\ & & \downarrow & \downarrow \\ i & \to & 0 & 0 \\ i+1 & \to & C_i & 0, \end{array}$$

Причем C_i будет выглядеть как блочно-диагональная матрица с одинаковыми блоками константного размера $2^{\text{количество задействованных битов}}$. Даже после таких ухищрений видно, что, строго говоря, на каждом шаге может изменяться много битов: прибавление единицы к счетчику может вызвать перенос разряда и т. д., но это-то мы уже сможем разложить на элементарные операции (формально, вновь увеличив матрицу C).

Итак, квантовое вычисление — это последовательность квантовых воздействий на константное число квантовых битов.

Определение 6.1. Язык принадлежит классу **BQP**, если $\exists$ квантовый полиномиальный по времени алгоритм, который ошибается с ограниченной вероятностью (1/4, например).

Возникают проблемы.

1. Можно ли вложить детерминированные вычисления в квантовые? Как уже говорилось, матрица C для квантового вычисления должна быть обратимой, а для детерминированного — не обязательно. Эта проблема решается. Можно пользоваться дополнительной памятью, т.е. увеличить длину квантовой ленты и за счет этого сделать матрицу обратимой (размер матрицы тоже увеличится).

2. Что будет, если в середине работы посмотреть промежуточный результат, повлияет ли это на окончательный результат? На самом деле, смотреть на конфигурацию «классически» в середине работы — ни к чему.

3. Нужно ли так много амплитуд? Вместо $\mathbb{C}$, можно использовать множество амплитуд $\{-1, -\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}, 1\}$ Можно доказать, что вычислительная сила машины не изменится.

4. Можно ли увеличить вероятность получения правильного ответа, повторив квантовый алгоритм несколько раз? Ответ: да. Но это не столь же очевидно, сколь в вероятностном случае.

5. Существует ли универсальная квантовая машина Тьюринга? Да, но ее конструкция технически сложна.

Квантовый бит (q-бит), в отличие от обычного, находится не в конкретном состоянии 1 или 0, как обычный, а одновременно в обоих с некоторыми амплитудами. Записывается это так: $\frac{1}{\sqrt{2}}\left|0\right\rangle - \frac{1}{\sqrt{2}}\left|1\right\rangle$, т.е. q-бит — это элемент линейного пространства, базисные вектора $(1,0)^t$ (бит находится в состоянии 0) и $(0,1)^t$ (бит находится в состоянии 1) которого обозначаются $\left|0\right\rangle$ и $\left|1\right\rangle$. (По существу, это точно так же для памяти вероятностных алгоритмов: каждый символ на ленте — случайная переменная; просто в терминах линейных пространств о них обычно не рассуждают.)

Несколько q-битов — квантовый вектор. Пример: $\frac{1}{\sqrt{2}}\left|00\right\rangle - \frac{1}{\sqrt{2}}\left|10\right\rangle$. Первый q-бит здесь находится в состоянии $\frac{1}{\sqrt{2}}\left|0\right\rangle - \frac{1}{\sqrt{2}}\left|1\right\rangle$, а второй точно является нулем. Если квантовый вектор имеет вид $\sum_{x=00..0}^{11..1} \lambda_x \left|x\right\rangle$, то, например, первый бит имеет вид: $\sum_{x=00..0}^{01..1} \lambda_x \left|0\right\rangle + \sum_{x=10..0}^{11..1} \lambda_x \left|1\right\rangle$. Квантовая конфигурация нашего квантового компьютера является как раз квантовым вектором.

Состояния вида $\left|x\right\rangle$, (где одна амплитуда равна 1, а остальные 0) называются чистыми (или классическими). Измеряя состояние, в смысле смотря ответ, мы получаем как раз такое состояние. Если проводится измерение, амплитуда состояния определяет только вероятность его появления.

6.2 Алгоритм Гровера

Постановка задачи. Теперь опишем алгоритм поиска в абстрактной базе данных. Пусть есть предикат $\mathcal{U} : \{0,1\}^n \longrightarrow \{0,1\}$. Надо найти вход, на котором ответ равен 1. Для простоты (и только для нее) будем считать, что такой вход ровно один. В классических случаях пришлось бы произвести $\Omega(2^n)$ шагов, квантовом достаточно $O(2^{n/2})$.

Оператор U. Предикат наш должен быть квантовым, он будет воздействовать на нашу систему в целом,

$$U\left|x\right\rangle = \begin{cases} \left|x\right\rangle, & x \neq x_0 \\ -\left|x\right\rangle, & x = x_0 \end{cases}$$

(x_0 — как раз такой, что $\mathcal{U}(x_0) = 1$). Матрица U имеет вид диагональной, где все элементы равны 1, кроме одного, равного -1. Эта -1 как раз находится в строке и столбце x_0. Мы определили U на базисных (чистых) состояниях. По линейности он определен и на всех остальных: например, $U(\frac{1}{\sqrt{2}}\left|x_0\right\rangle - \frac{1}{\sqrt{2}}\left|x\right\rangle) = -\frac{1}{\sqrt{2}}\left|x_0\right\rangle - \frac{1}{\sqrt{2}}\left|x\right\rangle$.

Оператор W. Итак, изначально у нас состояние $|00\dots0\rangle$. Теперь определим оператор W. Он будет делать из нулевого — случайный вектор (квантовых случайных векторов, правда, много (знаки амплитуд!); нам нужен некий конкретный). Определим операцию над одним битом:

$$S' = \begin{array}{ccc} & |0\rangle & |1\rangle \\ |0\rangle & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ |1\rangle & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}. \end{array}$$

Напомним, что для того, чтобы оформить ее в виде матрицы, действующей на все состояние, надо построить блочно-диагональную (с точностью до перестановки битов) матрицу; например, для конфигурации из двух битов подействовать матрицей S' на последний бит можно так:

$$S_1 = \begin{array}{ccccc} & |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ |00\rangle & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ |01\rangle & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ |10\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ |11\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{array}.$$

Итак, после этого преобразования второй q-бит, если он был $|0\rangle$, станет $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, а если он был $|1\rangle$, то он станет $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. (Это как бы значит, что он равен $|0\rangle$ и $|1\rangle$ с верятностями $\frac{1}{2}$.) Чтобы преобразовать первый q-бит, надо «умножить»[1] на матрицу S_2 (S_1 с переставленными блоками, чтобы изменился именно первый q-бит). И так далее. После n умножений наш исходный вектор $|00\dots0\rangle$ превратится в $\sum_x \frac{1}{\sqrt{N}}|x\rangle$ (обозначим его $|\xi\rangle$), где $N = 2^n$. Пусть $W = S_1 \cdot S_2 \cdot \ldots \cdot S_n$.

Оператор Z. Определим оператор Z так, как будто бы это U, но обращающий амплитуду не x_0, а 0. Он делается моделированием детерминированного вычисления внутри квантового. Надо устроить дополнительную память, в один из битов которой после n шагов проверки запишем, является ли текущий вектор полностью нулевым; потом же в зависимости от значения этого бита обратим амплитуду нашего вектора или нет.

Упражнение 6.1. Выписать квантовый оператор, соответствующий этому (не)обращению амплитуды.

[1]См. выше замечание о счетчике шагов, но, абстрагируясь от этого, можно считать, что мы именно перемножаем матрицы: эффект воздействия на биты, отличные от счетчика шагов, будет именно таков, как дало бы произведение матриц.

Итоговая матрица (про дополнительную память забыли) получится такой:

$$Z = \begin{matrix} -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{matrix}.$$

(Сразу так задавать Z было нельзя, т.к. получилось бы, что машина смотрит одновременно сразу на *всю* строку, чтоб решить, писать перед ней минус, или нет. Мы же разложили это на элементарные действия при помощи дополнительной памяти, о которой затем забыли.)

Оператор V. Определим, наконец, $V = W^{-1}ZW = WZW$ (ведь $W = W^{-1}$).

Собственно алгоритм. Теперь алгоритм таков:

$$(VU)^k W |0\rangle .$$

Утверждается, что если применить VU порядка $k \approx \sqrt{N}$ раз, то с большой вероятностью мы увидим конфигурацию $|x_0\rangle$.

Анализ алгоритма. Посчитаем:

$$\begin{aligned} U|\xi\rangle &= |\xi\rangle - \frac{2}{\sqrt{N}}|x_0\rangle, \\ U|x_0\rangle &= -|x_0\rangle, \\ V|\xi\rangle &= WZW|\xi\rangle = WZ|0\rangle = W(-|0\rangle) = -|\xi\rangle, \\ V|x_0\rangle &= \ldots = |x_0\rangle - \frac{2}{\sqrt{N}}|\xi\rangle. \end{aligned}$$

Таким образом, мы не выходим из подпространства, натянутого на вектора $|\xi\rangle$ и $|x_0\rangle$. Пусть дан вектор $q = \alpha|\xi\rangle + \beta|x_0\rangle$.

$$\begin{aligned} U(\alpha|\xi\rangle + \beta|x_0\rangle) &= \alpha|\xi\rangle - \left(\frac{2}{\sqrt{N}}\alpha + \beta\right)|x_0\rangle, \\ V(\alpha|\xi\rangle + \beta|x_0\rangle) &= -\left(\alpha + \frac{2\beta}{\sqrt{N}}\right)|\xi\rangle + \beta|x_0\rangle. \end{aligned}$$

Т.е. U оставляет координату при $|\xi\rangle$, а V — при $|x_0\rangle$. Мы также помним, что все операторы должны сохранять длины векторов. Смотрим на рисунок 6.2. Немного подумав, понятно, что $\varphi + \psi = \frac{\pi}{2}$; $\cos\varphi = |\xi\rangle \cdot |x_0\rangle$ (скалярное произведение). Т.е. $\cos\varphi = \frac{1}{\sqrt{N}}$, т.е. $\varphi \approx \frac{\pi}{2} \Rightarrow \sin\psi \approx \frac{1}{\sqrt{N}} \Rightarrow$

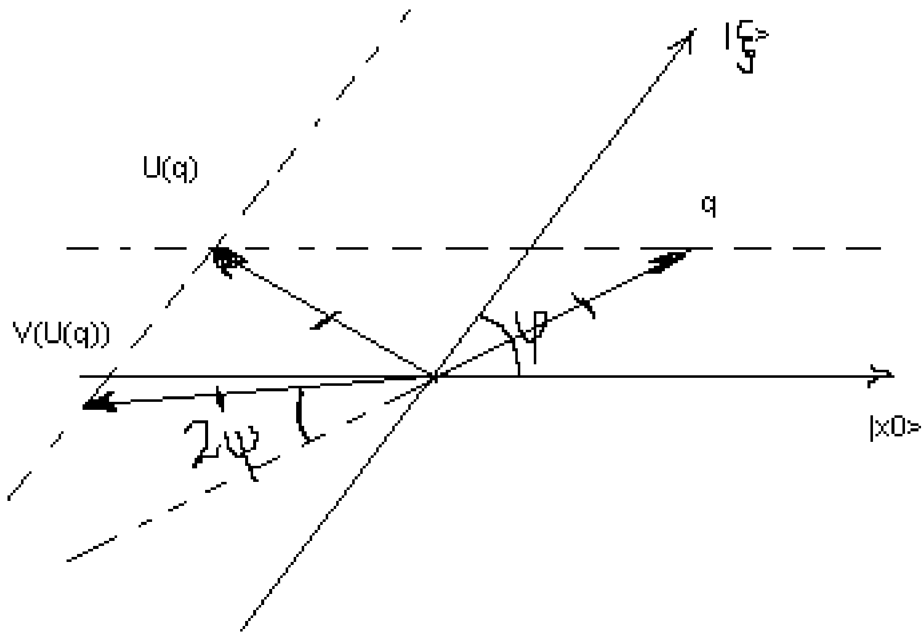

Рис. 6.2: применение операторов U и V

$2\psi \approx \frac{2}{\sqrt{N}}$. Сделав два поворота (т.е. $V(U(V(U(q)))))$, мы повернемся на угол $\approx \frac{4}{\sqrt{N}}$. Начинаем мы на $W\left|0\right\rangle = \left|\xi\right\rangle$. Т.е. надо повернуться на угол $\approx \frac{\pi}{2}$, чтобы получить почти $\left|x_0\right\rangle$. Т.е. надо сделать около $\frac{\pi}{2}/\frac{4}{\sqrt{N}} = \frac{\pi}{8}\sqrt{N}$ поворотов $(VU)^2$.

Ошибка вылезает из разных мест ($\varphi \neq \pi/2 \Rightarrow \sin x \neq x$, а количество поворотов — нецелое), но она порядка $\frac{1}{\sqrt{N}}$, т.е. амплитуда интересующего нас вектора достаточно близка к 1.