Дискретная математика

Григорьева Н.С.[1]

13.09.2023 - ...

[1]"Записал Сергей Киселев, Гараев Тагир"

Оглавление

Глава 1

Комбинаторика

Лекция 1: Введение

13.09.2023

1.1 Основные определения

Определение 1. Перестановкой называется упорядоченный набор неповторяющихся элементов длины n, состоящий из элементов от 1 до n.

Число перестановок: $P_n = n!$

Определение 2. Размещением называется упорядоченный набор неповторяющихся элементов длины k, состоящий из элементов от 1 до n.

Число размещений: $A_n^k = n(n-1)(n-2)\cdot\ldots\cdot(n-k+1) =$
$= \frac{n(n-1)(n-2)\cdot\ldots\cdot(n-k+1)(n-k)!}{(n-k)!} = \frac{n!}{(n-k)!}$

Определение 3. Сочетанием называется набор неповторяющихся элементов длины k, состоящий из элементов от 1 до n.

Число сочетаний: $C_n^k = \frac{A_n^k}{k!} = \frac{n!}{k!(n-k)!}$

Определение 4. Перестановки с повторениями: $\overline{P_n} = \frac{n!}{n_1!\cdot n_2!\cdot\ldots\cdot n_k!}$

Определение 5. Размещения с повторениями: $\overline{A_n^k} = n^k$

Определение 6. Сочетания с повторениями: $\overline{C_n^k} = C_{n+k-1}^k$

Пример. (Толкование к сочетаниям с повторениями) Сколькими способами можно разложить пять одинаковых шаров по трём различным ящикам? На число шаров в ящике ограничений нет.

Решение:

Представим себе, что ящики стоят вплотную друг к другу. Три та-

ких ящика — это фактически две перегородки между ними. Обозначим шар нулём, а перегородку — единицей. Тогда любому способу раскладывания пяти шаров по трём ящикам однозначно соответствует последовательность из пяти нулей и двух единиц; и наоборот, каждая такая последовательность однозначно определяет некоторый способ раскладывания. Например, 0010010 означает, что в первом ящике лежат два шара, во втором — два шара, в третьем — один шар; последовательность 0000011 соответствует случаю, когда все пять шаров лежат в первом ящике.

Теперь ясно, что способов разложить пять шаров по трём ящикам существует ровно столько же, сколько имеется последовательностей из пяти нулей и двух единиц. А число таких последовательностей равно C_7^2

1.2 Множества

Теорема 1. (Формула включений-исключений)
$|A_1 \cup A_2 \cup \ldots \cup A_n| = |\bigcup_{i=1}^n A_i| =$

$$= \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1} |A_1 \cap A_2 \cap \ldots \cap A_n|$$

Доказательство. (докажем по индукции)

1. База индукции: $n = 2 : |A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$
2. Переход индукции: $n \to n + 1$:
   $|A_1 \cup A_2 \cup \ldots \cup A_{n+1}| = |A_1 \cup A_2 \cup \ldots \cup A_n| + |A_{n+1}| - |(A_1 \cup A_2 \cup \ldots \cup A_n) \cap A_{n+1}| =$

$$= \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| -$$

$$- \ldots + (-1)^{n-1} |A_1 \cap A_2 \cap \ldots \cap A_n| + |A_{n+1}| - |(A_1 \cap A_{n+1}) \cup (A_2 \cap A_{n+1}) \ldots \cup (A_n \cap A_{n+1})| =$$

$$= \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| -$$

$$- \ldots + (-1)^{n-1} |A_1 \cap A_2 \cap \ldots \cap A_n| - (\sum_{i=1}^{n} |A_i \cap A_{n+1}| -$$

$$- \sum_{1 \le i < j \le n} |A_i \cap A_j \cap A_{n+1}| + \ldots (-1)^{n-1} |A_1 \cap A_2 \cap \ldots \cap A_{n+1}|) =$$

$$= \sum_{i=1}^{n+1} |A_i| - \sum_{1 \le i < j \le n+1} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n+1} |A_i \cap A_j \cap A_k| -$$

$$\ldots + (-1)^n |A_1 \cap A_2 \cap \ldots \cap A_{n+1}|$$

□

1.3 Разбиения

Определение 7. Пусть A — множество. Имеется $A_1, A_2, \ldots, A_n$.
Совокупность этих множеств — разбиение, если: $A = \bigcup_{i=1}^{n} A_i; A_i \cap A_j = \varnothing$

Определение 8. Пусть у A есть разбиения $\mathcal{A}$ и $\mathcal{B}$
Тогда $\mathcal{B}$ — измельчение $\mathcal{A}$, если $\forall B_i \in \mathcal{B}\ \exists! A_j \in \mathcal{A} : B_i \subset A_j$

Определение 9. Произведение разбиений — разбиение, которое является измельчением $\mathcal{A}$ и $\mathcal{B}$ и является самым крупным измельчением.

Лекция 2: Разбиения, прямое произведение, нумерация

20.09.2023

Теорема 2. Произведение разбиений существует.

Доказательство. $\mathcal{A}, \mathcal{B}$ — разбиения.
Возьмем все множества вида $C_{ij} = A_i \cap B_j$

- C — измельчение $\mathcal{A}$, так как $\forall C_{ij}\ \exists A_i : C_{ij} \subset A_i$
- аналогично C — измельчение $\mathcal{B}$

Предположим, что F — измельчение, большее C, тогда:
$\forall F_k : \begin{cases} \exists A_i : F_k \subset A_i \\ \exists B_i : F_k \subset B_i \end{cases} \Rightarrow F_k \subset A_i \cap B_j \Rightarrow F_k \subset C_{ij}$ — C наибольшее из измельчений. □

Глава 2

Алгоритмы перебора

2.1 Перебор 0-1 векторов

Будем рассматривать множество B^m всех наборов из m битов, каждый из которых может быть нулем и единицей. Элемент множества B^m — вектор $(0, 0, 1, \ldots, 1)$ длиной m. Количество элементов в множестве (мощность): $|B^m| = 2^m$.

Для того, чтобы создать вычислительный процесс, при котором на каждом шаге будет формироваться новый, не встречавшийся ранее, элемент рассматриваемого множества, достаточно заметить, что существует взаимнооднозначное соответствие между числами из $0 \ldots 2^m - 1$ и наборами 0-1 векторов. Т. е. достаточно первым взять число 0 и его двоичное представление $(0, \ldots, 0)$, а затем просто добавлять по единице, имитируя это на текущем наборе, пока мы не дойдем до набора из одних единиц.

Кроме рассмотренного способа перебора наборов, можно предложить другой алгоритм, который на каждом шаге меняет значение только одной компоненты:

Алгоритм. (Перебор и нумерация 0-1 векторов в порядке минимального изменения)

- создаем 2 набора x и y, каждый из m битов. Первоначально $x = y = (0, 0, 0, 0)$
- прибавляем к x единицу и фиксируем позицию j, где произошло изменение.
- изменить j-ую компоненту в наборе $y : y_j = 1 - y_j$
- вернуть y

Пример. (Рассмотрим на примере $m = 4$)

x	y	j
0000	0000	-
0001	0001	4
0010	0011	3
0011	0010	4
0100	0110	2
0101	0111	4

2.2 Перебор прямого произведения

Рассматриваем множество $M(1:k) = M_1 \times M_2 \times \ldots \times M_k$. Число элементов: $\prod_{i \in 1:k} m_i$, где $m_i = |M_i|$.

Будем считать, что каждое M_i состоит из m_i элементов, которые мы будем нумеровать от 0 до $m_i - 1$. Тогда каждый элемент $M(1:k)$ — последовательность неотрицательных чисел $(r_1, \ldots, r_k), r_i < m_i$

Общая формула перехода от элемента $(r_1, \ldots, r_k)$ к номеру этого элемента:

$$num(r_1, \ldots, r_k) = \sum_{i=1}^{k} \times (\prod_{j=1}^{i-1} m_j)$$

2.3 Перебор перестановок

Рассмотрим множество $T_k = M_1 \times M_2 \times \ldots \times M_k, M_i = \{0, 1, \ldots, i-1\}, |T_k| = k!$. Обозначим множество всех перестановок из k элементов через P_k.

Построим взаимнооднозначное соответствие между T_k и P_k. Возьмем перестановку $(r_1, \ldots, r_k)$ и сопоставим ей элемент $(t_1, \ldots, t_k)$ следующим образом: $\forall i \in 1:k$ найдем число значений, меньших r_i среди $r_{i+1}, \ldots, r_k$ — это число перепишем в качестве t_i.

Пример.

i	1	2	3	4	5	6	7	8
r_i	4	8	1	5	7	2	3	6
t_i	3	6	0	2	3	0	0	0

Чтобы получить перестановку по записи $(t_1, \ldots, t_k)$, нужно помнить множество значений S_i, которые могут быть в перестановке на i-ом месте. Так, $S_1 = 1:8, t_1 = 3$ означает, что $r_1 = 4$. Далее $S_2 = 1:3 \cup 5:8, t_2 = 6$ значит, что $r_2 = 8$

Замечание. Если использовать отображение из примера при переборре, то перестановки будут идти в лексикографическом порядке Это значит, что:

$(r_1, \ldots, r_k)$ предшествует $(R_1, \ldots, R_k) \Leftrightarrow$ начала этих перестановок совпадают до i индекса, а далее $r_i < R_i$

Замечание. Очевидно, что если факториальная запись $(t_1, \ldots, t_k)$ лек-

сикографически предшествует другой, то порядок верен и для соответствующих перестановок.

Алгоритм. (Перебор перестановок в лексикографическом порядке)

1. в заданной перестановке $(r_1, \ldots, r_k)$ найдем наибольший суффикс $(r_t, \ldots, r_k)$, в котором элементы расположены по убыванию.
2. выбрать в $(r_t, \ldots, r_k)$ элемент, следующий по велечине после r_{t-1} и поставить его на r_{t-1}. Оставшиеся эелменты, включая r_{t-1} расположить за ним в порядке возрастания.

Пример.

3	4	2	1	7	8	9	5	**6**
3	4	2	1	7	8	**9**	**6**	**5**
3	4	2	1	7	9	5	6	**8**
3	4	2	1	7	9	5	**8**	**6**
3	4	2	1	7	9	6	5	**8**
3	4	2	1	7	9	6	**8**	**5**
3	4	2	1	7	9	8	5	**6**
3	4	2	1	7	**9**	**8**	**6**	**5**
3	4	2	1	8	5	6	7	**9**

Лекция 3: Продолжение

27.09.2023

2.4 Перебор перестановок в лекцикографическом порядке (другой способ)

Алгоритм. Будем брать элементы $(t_1, \ldots, t_k)$ из T_k и сопоставлять им перестановки так, как делали ранее. Переход к следующей перестановке осуществляется путем прибавления единицы к $(t_1, \ldots, t_k)$. (причем последний элемент в $(t_1, \ldots, t_k)$ всегда ноль, т.к. ничего не значит)

Пример. (для P_4)

num	t	p
0	(0, 0, 0, 0)	(1, 2, 3, 4)
1	(0, 0, 1, 0)	(1, 2, 4, 3)
2	(0, 1, 0, 0)	(1, 3, 2, 4)
3	(0, 1, 1, 0)	(1, 3, 4, 2)
4	(0, 2, 0, 0)	(1, 4, 2, 3)
5	(0, 2, 1, 0)	(1, 4, 3, 2)
6	(0, 3, 0, 0)	(2, 1, 3, 4)
⋮	⋮	⋮
23	(3, 2, 1, 0)	(4, 3, 2, 1)

2.5 Перебор с минимальным изменением

На каждой итерации будем менять только два соседних элемента. Для этого необходимо:

- берем последний элемент в перестановке и меняем его с соседом до тех пор, пока элемент не дойдет до начала.
- когда этот элемент оказался в начале, мы меняем у него направление: теперь он будет менятся с соседом справ, а элемент, который оказался на последней позиции, делает 1 шаг (и так каждый раз, когда наш первый элемент меняет направление). Такие действия применяются ко всем элементам в перестановке.

Алгоритм. Кроме самой перестановки p и ее номера t (на этот раз младший разряд в номере — последний), будем хранить массив d, в котором будем хранить направление движения элементов. Если элемент движется вправо, то d[i] = +, если влево, то d[i] = -. Начальное значение d[i] = - для всех i.

Также храним j где будем записывать индекс элемента в t, в котором значение увеличилось.

1. Прибавляем 1 к t
2. Определяем номер разряда в котором значение увеличивается на 1, записываем в j
3. $\forall i \in [1, n] : i > j$, меняем $d_i = -d_i$.
4. j (не номер, именно такой элемент) меняем с соседом слева если $d_j = -$, и с соседом справа, если $d_j = +$.

Пример.

num	t	d	p	j
0	0000	$----$	1234	-
1	0001	$----$	1243	4
2	0002	$----$	1423	4
3	0003	$----$	4123	4
4	0010	$---+$	4132	3
5	0011	$---+$	1432	4
6	0012	$---+$	1342	4
⋮	⋮	⋮	⋮	⋮
19	0113	$--++$	4231	4
20	0120	$--++$	4213	3
21	0121	$--++$	2413	4
22	0122	$--++$	2143	4
23	0123	$--++$	2134	4
24	1000	$-+--$	—	1

2.6 Сочетания и бином Ньютона

$C_n^k = \frac{n!}{k!(n-k)!}$

Свойства. (Свойства сочетаний)

1. $C_n^k = C_n^{n-k}$
2. $C_{n-1}^k + C_{n-1}^{k-1} = C_n^k$

Доказательство. Доказывается путем подстановки непосредственно в формулу, или можно рассматривать пути на целочисленной решетке. □

Теорема 3. (Бином Ньютона)

$$\forall a, b \in \mathbb{R}, n \in \mathbb{N}_0 : (a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$$

Доказательство. (По индкуции)

1. База $n = 1$ очевидна.
2. индкуционый переход $n - 1 \to n$:

$$(a+b)^n = (a+b)(a+b)^{n-1} = a(a+b)^{n-1} + b(a+b)^{n-1} =$$

$$= a \cdot \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{n-1-k} + b \cdot \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{n-1-k} =$$

$$= \sum_{k=0}^{n-1} C_{n-1}^k a^{k+1} b^{n-1-k} + \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{n-k} =$$

$$= \sum_{k=1}^{n} C_{n-1}^{k-1} a^k b^{n-k} + \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{n-k} =$$

$$= a^n + \sum_{k=1}^{n-1} C_{n-1}^{k-1} a^k b^{n-k} + b^n + \sum_{k=1}^{n-1} C_{n-1}^k a^k b^{n-k} =$$

$$= a^n + b^n + \sum_{k=1}^{n-1} (C_{n-1}^k + C_{n-1}^{k-1}) a^k b^{n-k} = (a+b)^n$$

□

2.7 Перебор сочетаний с хорошей нумерацией

Для того чтобы присваивать номер сочетанию, будем рассматривать сочетание как вектор из нулей и единиц: если элемент взяли — единица, иначе — ноль.

Пример. Вектору $b = (1, 0, 1, 1, 1, 0, 0, 0)$ соответствует сочетание 1345.

Алгоритм. определяем номер рекурсивно:

$$num(b[1:n-1], m) = \begin{cases} num(b[1:n-1], m), & \text{если} b[n] = 0, \\ num(b[1:n-1], m-1), & \text{если} b[n] = 1, \end{cases}$$

Где m — кол-во единиц.

Пример. Рассмотрим $b = (1, 0, 1, 1, 1, 0, 0, 0), m = 4$:

$$\begin{aligned} num(b, m) =& C_6^4 + num(b[1:n-1], 3) = \\ =& C_6^4 + C_4^3 + num(b[1:n-2], 3) = \\ =& C_6^4 + C_4^3 + num(b[1:n-3], 2) = \\ =& C_6^4 + C_4^3 + num(b[1:n-4], 2) = \\ =& C_6^4 + C_4^3 + C_2^2 + num(b[1:n-5], 1) = \\ =& C_6^4 + C_4^3 + C_2^2 + 0 = 15 + 4 + 1 = 20 \end{aligned}$$

Лекция 4: Фибоначчи и теория вероятностей

04.10.2023

2.8 Фибоначчи

Определение 10. Последовательность Фибоначчи определяеется как:

$$\begin{cases} F_0 = 0, \\ F_1 = 1, \\ F_n = F_{n-1} + F_{n-2}, n \geq 2. \end{cases}$$

Лемма 1.

$$F_{2k} = F_{2k-1} + F_{2k-3} + \ldots + F_1$$

$$F_{2k-1} = F_{2k} + F_{2k-2} + \ldots + F_0 + 1$$

Доказательство. Докажем по индукции. База: $k = 1$:
$F_2 = F_1 + F_0 = 1 + 0 = 1$.
$F_1 = F_0 + 1 = 0 + 1 = 1$.
Переход: $k \to k + 1$. По предположению индукции:
$F_{2k} = F_{2k-1} + F_{2k-3} + \ldots + F_1$. Тогда $F_{2k+2} = F_{2k+1} + F_{2k-1} + \ldots + F_1 + F_0 = F_{2k+1} + F_{2k}$. Аналогично для нечетных. □

Теорема 4. (Представление натуральных чисел в виде суммы чисел Фибоначчи)
$\forall S \in \mathbb{N} : S = F_{i_0} + F_{i_1} + \ldots + F_{i_s}$
Где $i_0 = 0, i_{k-1} + 1 < i_k : k \in 1 : s$

Доказательство. Докажем, что такое представление существует. Пусть $j(s)$ — номер максимального числа Фибоначчи, не большего чем S. Положим $S' = S - F_{j(s)}$. Предположим, что $S' > F_{j(s)-1}$, тогда: $S' > F_{j(s)-1} \Rightarrow S > F_{j(s)} + F_{j(s)-1} \Rightarrow S > F_{j(s)+1}$, но по лемме $S \leq F_{js+1}$ — противоречие, значит $S' < F_{j(s)-1}$

Далее можно построить представление для S', итого число S прдставляется в виде представление для S', дополненное слагаемым $F_{j(s)}$

Проверим однозначность предсталвения: пусть $S = F_{j_0} + \ldots + F_{j_q}(2)$. Не умоляя общности, $j_q < j(s)$ (больше быть не может, а равные можно отбросить) Заменим F_{j_q} на $F_{j(s)-1}$, тогда правая часть равенства (2) увеличится. Будем заменять F_{j_q-1} на $F_{j(s)-3}$ и т.д. Но при таких заменах сумма не превзойдет $F_{j(s)}$ по лемме, значит, представление для S однозначно. □

2.9 Фибоначчиева система счисления

Вектор набора $(i_0, i_1, \ldots, i_s)$ — запись числа S в фибоначчиевой системе счисления.

Алгоритм. (Прибалвение единицы в фибоначчиевой системе счисления)

- Начальное положение: имеем набор $x[0 : n-1]$ из нулей и единиц, в котором нет двух единиц рядом и $x[0] = 1$.
- Положим $x[1] := 1$.
- Шаг: выбирем наибольшее k: $x[k] = 1 \wedge x[k-1] = 1$. Тогда:

$$\begin{cases} x[k] := 0, \\ x[k-1] := 0, \\ x[k+1] := 1, \\ x[0] := 1. \end{cases}$$

Пример.

0	1	2	3	4	5	6	7	8	9	пояснение
1	0	1	0	1	0	1	0	0	1	$= 46$
1	1	1	0	1	0	1	0	0	1	начало
1	0	0	1	1	0	1	0	0	1	1-й шаг
1	0	0	0	0	1	1	0	0	1	2-й шаг
1	0	0	0	0	0	0	1	0	1	3-й шаг

Глава 3

Элементарная теория вероятностей

3.1 Основные понятия

Определение 11. Ω — множество элементарных исходов. $A \subset \Omega$ — событие.

Определение 12. $\varnothing \subset \Omega$ — невозможное событие, $\Omega \subset \Omega$ — достоверное событие.

Определение 13. Пусть есть события A, B, тогда:

- $A \cup B$ — объединение событий
- $A \cap B$ — совмещение событий, причем, если $A \cap B = \varnothing$, то A, B — несовместные события.
- $\overline{A}$ — противоположное событие.

Определение 14. $p : \Omega \to [0, 1]$ — вероятность события. Причем:

1. $0 \leq p(A) \leq 1$
2. $A \subset B \Rightarrow p(A) \leq p(B)$
3. если $A = A_1 + A_2, A_1 \cap A_2 = \varnothing$, то $p(A) = p(A_1) + p(A_2)$

Определение 15. $p(A) = \frac{|A|}{|\Omega|}$ — классическая вероятность.

Определение 16. (Полная система событий)
Пусть есть события $S_1, \ldots, S_n$, таких, что:

- $S_i \cap S_j = \varnothing, i \neq j$
- $S_1 \cup \ldots \cup S_n = \Omega$

Тогда вероятность события A можно посчитать следующим образом:

$$p(A) = \sum_{i=1}^{n} p(A \cap S_i)$$

Определение 17. События A, B — независимы, если:

$$p(A \cap B) = p(A) \cdot p(B)$$

Определение 18. События $A_1, \ldots, A_n$ — независимы попарно, если:

$$p(A_i \cap A_j) = p(A_i) \cdot p(A_j)$$

И независимы в совокупности, если:

$$p(A_1 \cap \ldots \cap A_n) = \prod_{i=1}^{n} p(A_i)$$

Определение 19. (Условная вероятность)
Событие B при условии, что произошло событие A:

$$p(B|A) = \frac{p(A \cap B)}{p(A)}$$

Пример. Пусть есть тетраэдр с одной красной, одной черной, одной белой и одной, покрашенной во все цвета гранями. Тогда:

- $p(\text{Крас.}) = \frac{1}{2}$
- $p(\text{Крас.} \cap \text{Черн.}) = \frac{1}{4}$ — попарно независимы
- $p(\text{Крас.} \cap \text{Черн.} \cap \text{Бел.}) = \frac{1}{4}$ — не независимы в совокупности.

Определение 20. (Полная вероятность, используя условную)

$$p(A) = \sum_{i=1}^{n} p(A \cap S_i) = \sum_{i=1}^{n} p(A|S_i) \cdot p(S_i)$$

Определение 21. (Формула Байеса) Пусть есть события A, B:

$$p(B) = \sum_{i=1}^{n} p(B|A_i) \cdot p(A_i)$$

$$p(B|A_i) = \frac{p(A_i \cap B)}{p(A_i)} \Rightarrow p(B \cap A_i) = p(A_i) \cdot p(B|A_i) = p(B) \cdot p(A_i|B)$$

Получаем формулу Байеса:

$$p(A_i|B) = \frac{p(B|A_i) \cdot p(A_i)}{\sum_{j=1}^{n} p(B|A_j) \cdot p(A_j)}$$

Лекция 5: Случайные величины, мат.ожидание, дисперсия

11.10.2023

3.2 Случайные величины

Определение 22. Пусть Ω — множество элементарных событий, p_ω — вероятность события ω. Функция $\xi : \Omega \to \mathbb{R}^1$ называется случайной величиной.

Замечание. Способ задать случайную величину (дискретный случай)

$\xi :$

a_1	a_2	$\dots$	a_n
p_1	p_2	$\dots$	p_n

$p(\xi = a_i) = p_i$, где $\xi = a_i \Leftrightarrow \{\omega \in \Omega : \xi(\omega) = a_i\}$

Пример. Стрелок стреляет 3 раза, вероятность попадания — $0,8$:

0	1	2	3
$0,2^3$	$3 \cdot 0,2^2 \cdot 0,8$	$3 \cdot 0,8^2 \cdot 0,2$	$0,8^3$

3.3 Математическое ожидание

Определение 23. математическим ожиданием случайной величины называется:

$$E_\xi = \sum_{i=1}^{n} a_i \cdot p(\xi = a_i)$$

Свойства.

1. Если $p(\xi = a) = 1$, то $E_\xi = a$
2. Если $\eta = c \cdot \xi, c$ — константа, то $E_\eta = c \cdot E_\xi$
3. $E(\xi + \eta) = E_\xi + E_\eta$

4. $E(\xi \cdot \eta) = E_\xi \cdot E_\eta$ — для независимых

Определение 24. Случайные величины ξ и η — независимы, если:

$$p(\xi = a_i \wedge \eta = b_j) = p(\xi = a_i) \cdot p(\eta = b_j)$$

Доказательство. (доказательство свойства 3 для независимых величин)

$$E(\xi + \eta) = \sum_{i=1}^{n}\sum_{j=1}^{k}(a_i + b_j) \cdot p(\xi + \eta = a_i + b_j) =$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{k} a_i p(\xi = a_i)p(\eta = b_j) + \sum_{i=1}^{n}\sum_{j=1}^{k} b_j p(\xi = a_i)p(\eta = b_j) =$$

$$= \sum_{i=1}^{n}\left(a_i p(\xi = a_i)\underbrace{\sum_{j=1}^{k} p(\eta = b_j)}_{=1}\right) + \sum_{j=1}^{k}\left(b_k p(\eta = b_j)\underbrace{\sum_{i=1}^{n} p(\xi = a_i)}_{=1}\right) =$$

$$= E_\xi + E_\eta$$

□

Доказательство. (доказательство свойства 4)

$$\sum_{i=1}^{n}\sum_{j=1}^{k} a_i b_j p(\xi \cdot \eta = a_i b_j) = \sum_{i=1}^{n}\sum_{j=1}^{k} a_i b_j p(\xi = a_i)p(\eta = b_j) =$$

$$= \sum_{i=1}^{n}\left(a_i p(\xi = a_i)\sum_{j=1}^{k} b_j p(\eta = b_j)\right) = E_\xi \cdot E_\eta$$

□

3.4 Дисперсия

Определение 25. Дисперией случайной величины называется:

$$D_\xi = E(\xi - E_\xi)^2 = E(\xi^2 - 2\xi E_\xi + (E_\xi)^2) =$$

$$= E_{\xi^2} - E(2\xi E_\xi) + E((E_\xi)^2) = E_{\xi^2} - 2E_\xi E_\xi + (E_\xi)^2 =$$

$$= E_{\xi^2} = (E_\xi)^2$$

Пример.

$$\xi : \begin{array}{c|c|c} -1 & 0 & 1 \\ \hline \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{array} \quad E_\xi = -\tfrac{1}{4} + \tfrac{1}{4} = 0$$

$$\xi^2 : \begin{array}{c|c} 0 & 1 \\ \hline \frac{1}{2} & \frac{1}{2} \end{array} \quad E_{\xi^2} = \tfrac{1}{2}, D_\xi = \tfrac{1}{2} - 0 = \tfrac{1}{2}$$

Свойства.

1. $p(\xi = a) = 1 \Rightarrow D_\xi = 0$
2. $\eta = c \cdot \xi \Rightarrow D_\eta = c^2 D_\xi$
3. $D(\xi + \eta) = D_\xi + D_\eta$ — независимы

Доказательство. (свойство 3)

$$D(\xi+\eta) = E(\xi+\eta)^2 - (E(\xi+\eta))^2 = E(\xi^2 + 2\xi\eta + \eta^2) - (E_\xi + E_\eta)^2 =$$
$$= E_{\xi^2} + 2E_\xi E_\eta - (E_\xi)^2 - 2E_\xi E_\eta - (E_\eta)^2 = D_\xi + D_\eta$$

□

Лекция 6: Схема Бернулли, схема Уолкера, неравенство Крафта.

18.10.2023

3.5 Схема Бернулли

Определение 26. Пусть $\delta_1, ..., \delta_n$ — последовательность независимых одинаково распределенных случайных величин, каждая из которых принимает значение 1 с вероятностью p и значение 0 с вероятностью $q = 1 - p$. Такая вероятностная схема называется схемой Бернулли.

Определение 27. Случайная величина $\xi_n = \delta_1 + \ldots, \delta_n$ имеет биномиальное распределение:

$$p(\xi_n = k) = C_n^k p^k q^{n-k}$$

Замечание. Чтобы найти $\xi_n = k$ нужно чтобы k из случайных величин $\delta_1, ..., \delta_n$ принимали значение 1, остальные - 0. Вероятность этого события при фиксированных местах единиц и нулей равна $p^k q^{n-k}$, и если учесть все возможные C_n^k расположений этих мест, то получим $k-$ый член биномиального разложения $(p+q)^n$

Свойства. Т.к. $E_{\xi_1} = p$ и $D_{\xi_1} = p - p^2 = pq$:

1. $E_\xi = np$

2. $D_\xi = npq$

3.6 Случайные числа и схема Уолкера

Дискретная случайная величина — это такая случайная величина, значения которой могут быть не более чем счетными, то есть либо конечными, либо счетными. Под счётностью имеется ввиду, что значения случайной величины можно занумеровать.

В вычислительных машинах можно имитировать случайные эксперименты. В качестве источника случайности используются специальные программы - *датчики случайных чисел*. Датчик при каждом обращении к нему вырабатывает некоторое число (обычно целое или вещественное число из фиксированного диапазона), и последовательность этих чисел по своему поведению очень похожа на последовательность независимых *случайных величин, имеющих одинаковое равномерное распределение.*

Есть стандартные функции — генераторы (псевдо-)случайных чисел, которые выдают случайные натуральные числа в диапазоне $[0 : n]$, либо вещественные из $[0 : 1)$. Равномерное распределение обладает таким свойством, что вероятность того, что случайная величина принимает значения из некоторого множества, пропорциональна количеству элементов в нем. Или же что вероятность попадания в промежуток $[a, b] \subset [0 : 1)$ равна длине этого промежутка.

Определение 28. Равномерное (дискретное) распределение — распределение на конечном множестве, в котором все исходы равновероятны.

Пример. Равномерное распределение на множестве целых чисел от k до l :

$$p(\xi = S) = \frac{1}{l - k + 1}, S \in [k : l]$$

Алгоритм (Схема Уолкера). Пусть мы умеем получать случайное число из диапазона $[0 : 1)$. Требуется смоделировать вероятностную схему из m исходов с заданными вероятностями $p_1, p_2, ..., p_m$.

- Назовём “донорами” те исходы, которые получили больше, чем им нужно, т.е. $p_i < \frac{1}{m}$.
- Назовём “реципиентами” те исходы, которые получили меньше, чем им нужно, т.е. $p_i > \frac{1}{m}$.

Алгоритм перераспределения:

1. Берём произвольного донора и реципиента.
2. Донор отдаёт реципиенту излишек из своего отрезка.
3. Отмечаем точку, где донор отдал излишек реципиенту – барьер.

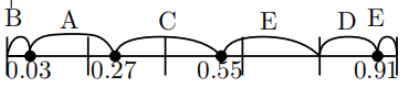
4. После этого у донора остаётся ровно столько, сколько ему нужно, а реципиент мог как остаться реципиентом (ему дали слишком мало), так и перейти в доноры (дали с избытком).

И так до тех пор, пока доноры и реципиенты не закончатся. После проведения такой предварительной работы можно за $O(1)$ генерировать случайные числа с нужным нам распределением следующим образом:

1. Генерируем случайное число x из диапазона $[0:1)$.
2. берем $\lfloor xm \rfloor$ – номер интервала на отрезке.
3. Сравниваем x с барьером на этом отрезке: если x больше, возвращаем реципиент, если меньше – донора.

Пример:
$m = 5, p(A) = 0.24, p(B) = 0.03, p(C) = 0.28, p(D) = 0.11, p(E) = 0.34$

i	Донор	Реципиент	Барьер
1	B	A	0.03
2	A	C	0.27
3	C	E	0.55
4	D	E	0.91



3.7 Двоичный поиск и неравенство Крафта.

Замечание. Данная глава писалась с опорой на другие источники. Думаю, вы понимаете почему. ⊙ ⌣ ⊙

Определение 29. Алфавит A – конечное непустое множество символов, $\alpha \in A^k$ – строка длины k над алфавитом A.

Определение 30. $A^* = \bigcup_{k=0}^{\infty} A^k$ – множество всех строк над алфавитом A (слова). ε – пустая строка.

Определение 31. Кодом называется функция $\varphi : A^* \to B^*$. B – алфавит кода.

Определение 32. Код называется декодируемым, если $\forall \alpha, \beta \in A^* : \alpha \neq \beta \Rightarrow \varphi(\alpha) \neq \varphi(\beta)$, т.е. φ – инъекция.

Пример. Операция сжатия каким либо архиватором – декодируемый код. *jpeg* – сжатие с потерей информации, недекодируемый код.

Определение 33. Разделяемый код – код, в котором каждый символ алфавита A кодируется отдельно, т.е. код – функция $\varphi : A \to B^*$.

Если хотим закодировать строку $\alpha = a_1a_2...a_n$, то $\varphi(\alpha) = \varphi(a_1)\varphi(a_2)...\varphi(a_n)$ (конкатенируем коды символов)

Определение 34. Префиксный код – функция $\varphi : A \to B^*$, такая, что $\forall a, b \in A : \varphi(a)$ – не префикс $\varphi(b)$, т.е. ни одно кодовое слово не является префиксом другого кодового слова.

Пример. Пример: $A = \{a, b, c\}, B = \{0, 1\}$
$\varphi(a) = 0, \varphi(b) = 10, \varphi(c) = 11 \Rightarrow \varphi$ – не префиксный код.

Теорема 5. Если код префиксный, то он однозначно декодируемый.

Доказательство. Приведем алгоритм декодирования.

Пусть дана $t = \varphi(s)$, нужно найти s. Тогда будем из t выделять префикс (он будет один, т.к. код префиксный), который является кодом некоторого символа a_1. Отрежем этот префикс от t. Так делаем, пока t не кончится. Получим $a_1a_2...a_n = s$. □

Теорема 6 (неравенство Крафта). Для алфавита $A = \{c_1, ..., c_k\}$ можно построить однозначно декодируемый двоичный код с длинами кодовых слов $s_1, \ldots, s_k$ если:

$$\sum_{i=1}^{k} 2^{-s_i} \leq 1 \qquad (6)$$

И наоборот, если для чисел $s_1, \ldots, s_k$ выполняется неравенство 6, то существует однозначно декодируемый двоичный код в алфавите A.

Доказательство. (Необходимость) Рассмотрим бинарное дерево T. Каждой вершине r сопоставим число $a_r = 2^{-l}$, где l длина пути от корня до вершины r. Для корня имеем $a_{root} = 1$.
Заметим, что для любого узла, который не лист, справедливо неравенство:

$$2^{-l} \geq \underbrace{2^{-(l+1)}}_{left} + \underbrace{2^{-(l+1)}}_{right}$$

Просуммируем все узлы, которые не являются листьями и все узлы не корни, тогда справедливо неравенство:

$$\sum_{nodes \setminus leaves} a_r \geq \sum_{nodes \setminus \{a_{root}\}} a_r$$

После сокращения общих слагаемых в левой и правой частях неравенства получаем корень слева и все листья справа:

$$a_{root} = 1 \geq \sum_{leaves} a_r = \sum_{i=1}^{k} 2^{-s_i}$$

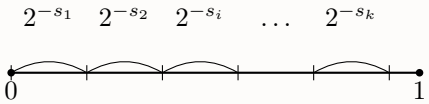
□

Лекция 7: Достаточность неравенства Крафта, задача о наилучших длинах кодов, энтропия.

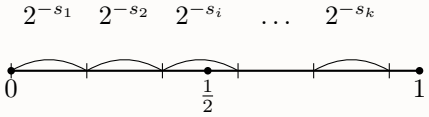
25.10.2023

Доказательство. (Достаточность)
Не умоляя общности, будем считать, что $s_1 \leq s_2 \leq \ldots \leq s_k$. Тогда $2^{-s_1} \geq 2^{-s_2} \geq \ldots \geq 2^{-s_k}$

Теперь распложим эти значения на отрезке $[0, 1]$:



Утверждение: $\frac{1}{2}$ является границей отрезков. (не может лежать внутри какого либо 2^{-s_i})
Предположим, что это не так, тогда:



Имеем неравенства:

$$\begin{cases} 2^{-s_1} + \ldots + 2^{-s_{i-1}} \leq \frac{1}{2} \\ 2^{-s_1} + \ldots + 2^{-s_{i-1}} + 2^{-s_i} > \frac{1}{2} \end{cases} \quad \text{домножим на } 2^{s_i} \Rightarrow$$

$$\Rightarrow \begin{cases} \underbrace{2^{s_i - s_1} + \ldots + 2^{s_i - s_{i-1}}}_{c} < 2^{s_i - 1} \\ \underbrace{2^{s_i - s_1} + \ldots + 2^{s_i - s_{i-1}}}_{c} + 1 > 2^{s_i - 1} \end{cases} \Rightarrow \begin{cases} c < 2^{s_i - 1} \\ c + 1 > 2^{s_i - 1} \end{cases} \quad \text{— противоречие.}$$

Таким образом какой-то отрезок из 2^{-s_i} упрется в $\frac{1}{2}$ или хотя бы не дойдет до нее. Тогда Разделим $2^{-s_1}, 2^{-s_2}, \ldots, 2^{-s_k}$ на 2 части: те, которые меньше $\frac{1}{2}$ и те, которые больше. Тем кодам, которые меньше $\frac{1}{2}$ поставим 0 в начало кода, а тем, которые больше $\frac{1}{2}$ поставим 1, тогда

их длина уменьшилась на единицу. Тогда сумма тех, что слева и тех, что справа:

$$\sum 2^{-(s_i-1)} = \sum 2 \cdot 2^{-s_i} = 2 \cdot \sum 2^{-s_i} \leq 1$$

Тогда можем рекурсивно выполнять деление отрезков, т.к. если есть всего 2 символа, можем дать им коды 0 и 1 соответственно. □

3.8 Задача о наилучших длинах кодов

Задан набор вероятностей $p_1, p_2, ..., p_m$, $p_i > 0$, $\sum\limits_{i=1}^{m} p_i = 1$. Требуется найти набор чисел $s_1, s_2, .., s_m$, таких что:

$$\sum_{i=1}^{m} p_i s_i \longrightarrow \min$$

$$\sum_{i=1}^{m} 2^{-s_i} \leq 1 s_i \in \mathbb{N}$$

Теорема 7. Минимум достигается функцией:

$$H(p) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

Доказательство. □

3.9 Энтропия

Определение 35. Энтропией вероятностной схемы называется мера содержащейся в ней неопределенности. Она задается как конкретная функция $H : RS \to \mathbb{R}^+$, где RS – множество всех возможных вероятностных схем.

Функция, задающая энтропию обладает рядом свойств, и этим свойствам удовлетворяет функция 7. Это докажем позже, а сейчас рассмотрим свойства энтропии:

Свойства.

1. Мера неопределенности непрерывно зависит от вероятностей. (функция 7 этим свойством, очевидно, обладает)
2. При перестановке вероятностей мера неопределенности не меняется.

3. Необходимо ввести единицу измерения неопределенности. За единицу будем брать энтропию честной монеты: $H(\{\frac{1}{2}, \frac{1}{2}\}) = 1$ – бит.
4. Обозначим за $h(m) = H(\{\frac{1}{m}, \frac{1}{m}, \dots, \frac{1}{m}\})$. Тогда $h(m)$ растет с ростом m. (функция 7 этим свойством тоже обладает)
5. При фиксированном m максимум энтропии достигается в случае равновероятных исходов, т.е. $h(m)$.
6. Пусть есть схемы $P_m = p_1, \dots, p_m$ и $Q_k = q_1, \dots, q_k$. Образуем комбинированную схему с $m-k+1$ исходами следующим образом: выбирается m-й исход в P_m и для него выбираются исходы из Q_k. Получим схему PQ с ихсодами:

$$1, 2, \dots, m-1, (m,1), (m,2), \dots, (m,k)$$

Вероятность этих исходов:

$$p_1, \dots, p_{m-1}, p_m q_1, \dots, p_m q_k$$

Тогда энтропия схемы PQ : $H(PQ) = H(P_k) + p_m H(Q_k)$

Доказательство. (Какие-то свойства либо уже доказаны, либо были даны без доказательства)

6

$$H(PQ) = \sum_{i=1}^{m-1} p_i \log_2 \frac{1}{p_i} + \sum_{j=1}^{k} p_m q_j \log_2 \frac{1}{p_m q_j} =$$

$$= \sum_{i=1}^{m-1} p_i \log_2 \frac{1}{p_i} + p_m \sum_{j=1}^{k} q_j \log_2 \frac{1}{p_m} + p_m \sum_{j=1}^{k} q_j \log_2 \frac{1}{q_j} =$$

$$= 1 \cdot p_m \log_2 \frac{1}{p_m} + \sum_{i=1}^{m-1} p_i \log_2 \frac{1}{p_i} + p_m \sum_{j=1}^{k} q_j \log_2 \frac{1}{q_j} = H(P_m) + p_m H(Q_k)$$

□

Лекция 8: Энтропия. Сортировки.

1.11.2023

Теорема 8. Если функция $G(p_1, \dots, p_m)$ обладает свойствами (1)-(6), то $G(p_1, \dots, p_m) = H(P_m)$

Лемма 2. Пусть $g(m) = G(\frac{1}{m}, \dots, \frac{1}{m})$ и $g(m)$ обладает свойствами (1)-(6), тогда:

$$g(m) = \log_2 m$$

Доказательство. Возьмем 2 схемы: Q_k и Q_l равновероятных исходов:

$$g(kl) = g(k) + g(l) \text{ – из (6)} \Rightarrow g(m^k) = k \cdot g(m)$$

$$g(2) = 1 \text{ – из (3)} \Rightarrow g(2^s) = s$$

Возьмем $s : 2^s \leq m^k \leq 2^{s+1}$, тогда, в силу монотонности g:

$$g(2^s) \leq g(m^k) \leq g(2^{s+1}) \Rightarrow s \leq kg(m) \leq s + 1$$

$$\frac{s}{k} \leq g(m) \leq \frac{s}{k}\frac{1}{k}$$

$$\frac{\lfloor \log_2 m^k \rfloor}{k} \leq g(m) \leq \frac{\lfloor \log_2 m^k \rfloor + 1}{k}$$

$$0 \leq g(m) - \log_2 m + \frac{\{k \log_2 m\}}{k} \leq \frac{1}{k}$$

$$\lim_{k\to\infty} \frac{\{k \log_2 m\}}{k} = 0 \Rightarrow g(m) = \log_2 m$$

□

Лемма 3. Если $\forall p_i \in \mathbb{Q} : p_i \in P_m$, то:

$$G(P_m) = H(P_m), \text{ где } G(P_m) \text{ удовлетворяет (1)-(6)}$$

Доказательство. Пусть $p_i = \frac{r_1}{n}$. Пусть есть равновероятные схемы $Q_{r_1}, \ldots, Q_{r_n}$, тогда, комбинируя их с P_m, получим равновероятную схему Q_n С n равновероятными исходами.

$$G(Q_n) = G(P_m) + \sum_{i=1}^{n} p_i G(Q_{r_i})$$

По лемме 1:

$$\log_2 n = G(P_m) + \sum_{i=1}^{m} p_i \log_2 r_i$$

$$G(P_m) = \log_2 n - \sum_{i=1}^{m} p_i \log_2 r_i = \log_2 n - \sum_{i=1}^{m} p_i(\log_2 p_i + \log_2 n) =$$

$$= \log_2 n - \sum_{i=1}^{m} p_i \log_2 p_i - \sum_{i=1}^{m} p_i \log_2 n =$$

$$= \sum_{i=1}^{m} p_i \log_2 \frac{1}{p_i}$$

□

Доказательство. (теоремы 8) Для любого набора вероятностей $\{p_1, ..., p_m\}$ рассмотрим сходящуюся к нему последовательность рациональных наборов $\{p_1^{(k)}, ..., p_m^{(k)}\}$. По лемме 2 для каждого из этих наборов $G = H$. Так как обе функции непрерывны, то это равенство выполняется и в предельной точке. □

Глава 4

Сортировки

4.1 Сортировка слиянием (Фон Неймана)

Первоначально сортируемый массив представляется в виде n “отсортированных массивов” длины 1. Далее массивы сливаются попарно и сортируются. Затем еще раз и т. д.

Пример. для массива $a = [38, 27, 43, 3, 9, 82, 10]$:

|38| |27| |43| |3| |9| |82| |10|
|27, 38| |3, 43| |9, 82| |10|
|3, 27, 38, 43| |9, 10, 82|
|3, 9, 10, 27, 38, 43, 82|

4.2 Сортировка вставками и сортировка Шелла

Идея для сортировки вставками: наращивать отсортированную часть последовательности. Сначала берем уже “отсортированную” последовательность из одного элемента. Далее берем очередной элемент, сравниваем его с предыдущим и переставляем до тех пор, пока он не займет свое место.

Пример. для массива $a = [38, 27, 43, 3, 9, 82, 10]$:

38	27	43	3	9	82	10
27	38	43	3	9	82	10
27	38	43	3	9	82	10
3	27	38	43	9	82	10
3	9	27	38	43	82	10
3	9	27	38	43	82	10
3	9	10	27	38	43	82
3	9	10	27	38	43	82

Идея для сортировки Шелла: давайте попробуем сократить количество перемещений элементов за счет того, что будем сдвигать их не на одну

позицию, а сразу на несколько.

Пример. Наш массив: 13 44 7 21 78 3 25 9 28 35 10 66 33 16 Выберем шаг d = 6, отдельно сортируем (13 25 33), (44 9 16), (7 28), (21 35), (78 10) и (3 66). Затем уменьшим шаг d вдвое и повторим процедуру. На последнем шаге d = 1, что соответствует «обычной» сортировке вставками.

Лекция 9: Сортировки. Сжатие и защита информации.

8.11.2023

4.3 Сортировка Хоара (быстрая сортировка)

Идея: выберем элемент-разделитель. Разместим все меньшие разделителя элементы слева от него, большие — справа и запустим процесс для каждой из частей.

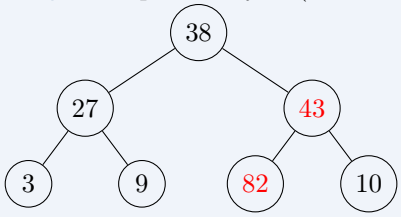
Пример.

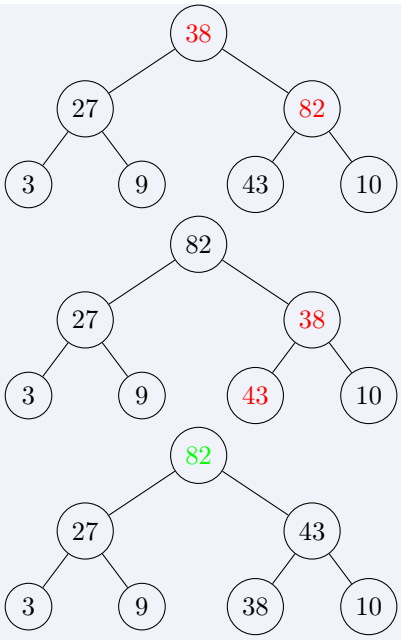
```
38 27 43 3 9 82 10 (pivot is 3)
3 27 38 43 9 82 10 (pivot is 38)
3 27 10 9 38 43 82 (pivot is 27)
3 10 9 27 38 43 82 (pivot is 10)
3 9 10 27 38 43 82 (pivot is 9)
3 9 10 27 38 43 82
```

4.4 Пирамидальная сортировка (иерархическая, heapsort)

1. **Построение кучи:** Преобразуем исходный массив в максимальную кучу. В максимальной куче значение каждого узла больше или равно значению его потомков.
2. **Сортировка:** Удаляем максимальный элемент (корень кучи), заменяем его последним элементом кучи, уменьшаем размер кучи на один и восстанавливаем свойство кучи. Повторяем, пока в куче не останется элементов.

Пример. Построение кучи (массив $a = [38, 27, 43, 3, 9, 82, 10]$):





Далее меняем местами 82 и 10, удаляем 82 из кучи и восстанавливаем свойство кучи.

4.5 Порозрядная сортировка (radix sort)

Если известно, что диапазон ключей ограничен, то сортировать можно значительно быстрее: сортируем последовательно по разрядам (например, по десятичным или двоичным), начиная с младшего. (Пример придумайте сами)

Глава 5

Сжатие и защита информации

5.1 Задача о префиксном коде.

Пусть $A = \{c_1, \ldots, c_n\}$ — алфавит, $p_1, \ldots, p_n$ — вероятности появления символов. Пусть задан код $\varphi : A \to \{0,1\}^*$
Фиксируем текст (большую строку) длины $N : a_1 a_2 \ldots a_N$. В этом тексте количество символов c_i – $N \cdot a_i$ (закон больших чисел). Тогда длина кодовой последовательности текста:

$$\sum_{i=1}^{n} N p_i \varphi(c_i)$$

Задача заключается в том, чтобы минимализировать длину кодовой последовательности такого текста, а т.к. N – фиксированная величина, не влияющая на коды, задача сводиться к тому, чтобы минимализировать сумму:

$$\sum_{i=1}^{n} p_i \varphi(c_i)$$

При этом для $p_i, \varphi(c_i)$ выполняются:

1. $\sum p_i = 1$
2. $0 < p_i < 1$
3. $\varphi(c_i) \in \mathbb{N}$
4. $\varphi(c_1), \ldots, \varphi(c_n)$ – длины кодов символов в префиксом коде, т.е. для этих длин выполняется неравенство Крафта.

Алгоритм решения такой задачи основывается на нескольких свойств длин кодовых последовательностей:

Лемма 4. Пусть p_i – вероятности символов и s_i – длины кодов символов для оптимального кода.

Если $p_1 \geqslant p_2 \geqslant ... \geqslant p_m$, то $s_1 \leqslant s_2 \leqslant ... \leqslant s_m$.

Доказательство. Мы просто покажем, что если найдутся две пары чисел (p_i, s_i) и (p_j, s_j), такие что $p_i < p_j$ и $s_i < s_j$, то значение суммы попарных произведений можно уменьшить, заменив эти две пары парами (p_i, s_j) и (p_j, s_i). Действительно, т.к. $(p_j - p_i)(s_j - s_i) > 0$, то, раскрывая скобки, получим после элементарных преобразований $p_i s_i + p_j s_j > p_i s_j + p_j s_i$.

Так как число возможных расположений конечно, поскольку равно $m!$, то начиная с любого расположения за конечное число шагов мы закончим процесс улучшений на расположении, которое дальше улучшить невозможно. На нём и достигается минимум. □

Лемма 5. Две самые длинные кодовые последовательности имеют одинаковую длину.

Доказательство. Предположим, что $s_{m-1} < s_m$. Тогда в силу префиксности кода, можно отбросить лишние символы из кода s_m. Мы получим снова префиксный код, но с меньшим значением целевой функции. Противоречие. □

Лемма 6. Рассмотрим задачу P', которая получается объединением двух самых редких символов в один символ. Минимальное значение целевой функции в задаче P' будет отличаться от значения в задаче P на $p'_{m-1} = p_{m-1} + p_m$, а оптимальный кодовый набор для P получается из решения задачи P' удлинением на 1 бит кодовых последовательностей для символов, которые были объединены.

Доказательство. Пересчитываем новую функцию:

$$\sigma(P) = \sum_{i=1}^{m-2} p_i s_i + (s'_{m-1} + 1)p_{m-1} + (s'_{m-1} + 1)p_m =$$

$$= \sum_{i=1}^{m-2} p_i s_i + s'_{m-1}(p_{m-1} + p_m) + p_{m-1} + p_m = \sigma(P') + p'_{m-1}.$$

Осталось доказать, что построенный код для P действительно оптимальный.

Предположим, что существует решение для P, такое что $\sigma^*(P) < \sigma(P)$. По лемме 2 длины кодов самых редких символов равны. Объединим эти символы в один, а код для него построим откидыванием последнего бита. В силу префиксности кода для P это можно сделать.

Таким образом, получили префиксный код для задачи P', и при этом $\sigma^*(P') < \sigma(P')$. Противоречие. □

5.2 Код Шеннона-Фано и алгоритм Хаффмана

Кодирование Шеннона — Фано — алгоритм префиксного неоднородного кодирования. Относится к вероятностным методам (т.е. мера на множестве событий, принимающая значения от 0 до 1) сжатия.

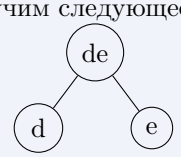
Часто встречающийся символ кодируется кодом меньшей длины (короткой двоичной последовательностью), редко встречающийся — кодом большей длины (длинной двоичной последовательностью).

Алгоритм (Хаффмана). Лемма 3 фактически даёт нам алгоритм для построения оптимального кода:

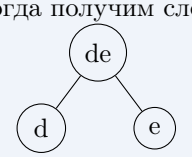
1. Составим список кодируемых символов вместе с их частотами. Каждый символ представим в виде одноэлементного дерева с соответствующим весом.
2. Из списка выберем два дерева с наименьшими весами.
3. Создадим новый корень и назначим ему в качестве детей эти два дерева. Вес сформированного дерева положим равным сумме весов дочерних. Таким образом, два дерева “склеились” в одно.
4. Добавим новое дерево в список.
5. Если в списке осталось ровно одно дерево, закончить процесс, иначе - перейти к п.2.

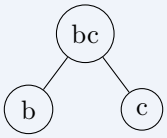
Пример. Пусть $A = \{a, b, c, d, e\}$, $p_a = 0.37, p_b = 0.22, p_c = 0.16, p_d = 0.14, p_e = 0.11$.

Объединим d и e в один символ de с вероятностью 0.25. Тогда получим следующее дерево:

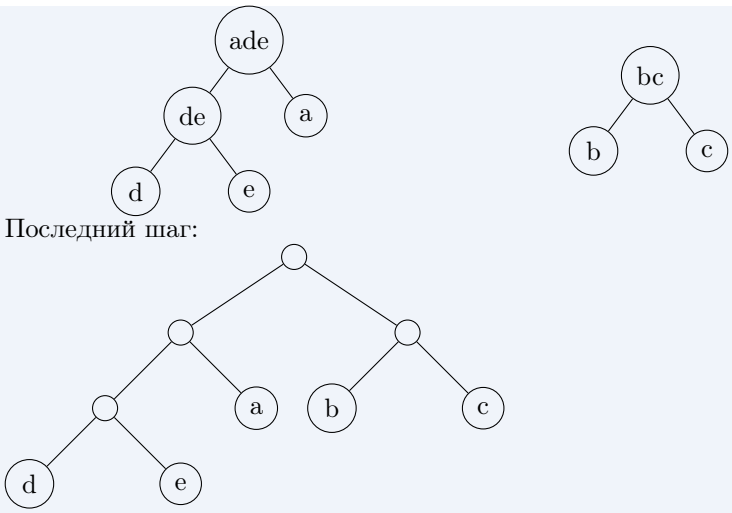


На следующем шаге объединим c и b в один символ bc с вероятностью 0.38. Тогда получим следующее дерево:





Следующий шаг:



Последний шаг:

Коды для символов: $a = 01, b = 10, c = 11, d = 000, e = 001$.

Лекция 10: Сортировки. Сжатие и защита информации. 15.11.2023

5.3 Сжатие по Хаффману

Пример. (из Романовского)
Рассмотрим текст из 223 знаков:

```
ехали_медведи_на_велосипеде_а_за_ними_кот_задом
_наперед_а_за_ним_комарики_на_воздушном_шарике_а
_за_ними_раки_на_хромой_собаке_волки_на_кобыле_
львы_в_автомобиле_зайчики_в_трамвайчике_жаба_
на_метле_едут_и_смеются_пряники_жуют
```

Применение сжатия по Хаффману дает следующий результат:

```
яь 3 жю 4 ыч 4 хш 4 йу 6 пяь 6 сб 8 жюыч 8 зхш 10
йупяь 12 тр 14 лд 14 сбжюыч 16 взхш 20 нк 22 мйупяь 24
отр 27 лдсбжюыч 30 евзхш 36 инк 42 амйупяь 48
отрлдсбжюыч 57 евзхш_ 76 инкамйупяь 90
отрлдсбжюычевзхш_ 133
```

Если взять кодовые последовательности для этих символов и закодировать ими текст, получим сжатие до 119 байтов, что почти вдвое меньше исходного текста.

Но код Хаффмана никак не учитывет закономерность распределения символов в тексте, поэтому он не является оптимальным.

5.4 Сжатие по Лемпелю-Зиву (LZ77, LZ78, LZO, LZMA, LZW etc.)

Алгоритм. Основная идея: Кодируемый текст разбивается на небольшие строки, каждая из которых составлена из одной из предыдущих строк (предопределена пустая строка, имеющая номер 0) и еще одного символа. Сначала записаны номера строк, затем сами строки, затем представление каждой из этих строк в виде пар (номер предыдущей строки, дополняющий символ).

Пример. На примере текста "aababaacbbbcccaaaaa": (сначала строки, затем представление строк в виде пар (номер предыдущей строки и дополняющий символ), затем номер строки)

```
 a ab aba ac  b bb  c cc aa aaa
0a 1b  2a 1c 0b 5b 0c 7c 1a  9a
 1  2   3  4  5  6  7  8  9  10
```

5.5 Метод Барроуза-Уилера (BWT)

Рассмотрим две строки: *aababbbbaa* и *aaaaabbbbb*. Вторая лучше с точки зрения сжатия, потому что она устроена проще.

Обычные тексты устроены сложнее, в них не просто буквы повторяются, а целые буквосочетания (например, артикли).

Идея преобразования Барроуза-Уилера (BWT — Barrous-Wheeler transformation) в том, чтобы так переставить буквы, чтобы:

1. одинаковые буквы по возможности шли подряд;
2. по преобразованной строке можно восстановить исходную.

Но это только преобразование, а не сжатие, ведь перестановки букв длину текста не уменьшают.

Алгоритм.
1. В конец строки добавляем специальный символ, минимальный лексикографически (для обратного преобразования)
2. Выписываем всем возможные циклические сдвиги строки.
3. Сортируем их лексикографически.
4. В качестве результата берем последний столбец полученной матрицы.

Обратное преобразование:

1. Берем исходную строку и записываем в первый столбец матрицы.

2. Сортируем строки матрицы лексикографически и записываем в следующий столбец.
3. Записываем в следующий столбец предыдущий + исходная строка слева.
4. Продолжаем, пока не получим столбец, в котором длина строк равна длине исходной строки, берем из этого столбца ту строку, в которой послений символ – специальный.

Пример. Имеется строка каркаркар. Добавляем в конец специальный символ, например, $.

каркаркар$	$каркаркар
аркаркар$к	ар$каркарк
ркаркар$ка	аркар$карк
каркар$кар	аркаркар$к
аркар$карк	кар$каркар
ркар$карка	каркар$кар
кар$каркар	каркаркар$
ар$каркарк	р$каркарка
р$каркарка	ркар$карка
$каркаркар	ркаркар$ка

Получили строку ркккрр$ааа, которая и будет результатом преобразования.

Обратное преобразование:

0	1	2	3	4	...	n
р	$	р$	$к	р$к	...	$каркаркар
к	а	ка	ар	кар	...	ар$каркарк
к	а	ка	ар	кар	...	аркар$карк
к	а	ка	ар	кар	...	аркаркар$к
р	к	рк	ка	рка	...	кар$каркар
р	к	рк	ка	рка	...	каркар$кар
$	к	$к	ка	$ка	...	каркаркар$
а	р	ар	р$	ар$	...	р$каркарка
а	р	ар	рк	арк	...	ркар$карка
а	р	ар	рк	арк	...	ркаркар$ка

Получили строку каркаркар$

Замечание. (на лекции этого не было, но посчитал это важным.) Преобразование и в ту, и в другую сторону можно реализовать за линейное время. (докажите сами...)

Для того, чтобы преобразование Барроуза-Уилера имело смысл, нужно

применить к полученной строке алгоритм сжатия.

Алгоритм (MTF (Move To Front)). Заведем вспомогательный массив, в котром будем хранить уникальные символы на текущей итерации. При встрече символа будет вставлять его в начало массива. Также заведем массив индексов, в котором будем хранить позиции символов в массиве уникальных символов.

1. Начальное положение: в массив уникальных символов записываем текущий элемент, в массив индексов записываем 0.
2. На i-ой итерации: если символа нет в массиве уникальных символов, то записываем его в начало массива, иначе находим его позицию в массиве и записываем эту позицию в массив индексов.

По итогу получим на выходе алфавит и набор индексов, по которым можно легко декодировать исходную строку.

Пример.

```
   i
 0: р 0 р
 1: к 0 кр
 2: к 1 кр
 3: к 1 кр
 4: р 2 рк
 5: р 1 рк
 6: $ 0 $рк
 7: а 0 а$рк
 8: а 1 а$рк
 9: а 1 а$рк
10: а 1 а$рк
```

Далее получившийся набор индексов можно закодировать с помощью кода Хаффмана.

Лекция 11: Избыточное кодирование. Криптография.

22.11.2023

5.6 Код Хэмминга

При передаче или хранении данных возможны искажения. Аппаратное обеспечение не идеально. Хочется научиться находить и исправлять такие ошибки. Для этого нужно передавать какую-то избыточную информацию.

Избыточное кодирование — это вид кодирования, использующий избыточное количество информации с целью последующего контроля целостности данных при записи/воспроизведении информации или при её передаче по линиям связи.

Код Хэмминг - это алгоритм, который позволяет закодировать какое-

либо информационное сообщение определённым образом, после передачи определить, появилась ли какая-то ошибка в этом сообщении во время его передачи, и, при возможности, восстановить это сообщение.

Рассмотрим самый простой код Хэмминга (может исправлять только одну ошибку). Также существуют более совершенные модификации данного алгоритма, которые позволяют обнаруживать большее количество ошибок.

Алгоритм (Код Хэмминга). Пусть требуется передать какое-либо сообщение a, состоящее из n битов (например, $a = [0, 1, \ldots, 1]$). Тогда, для построения кода Хэмминга, потребуется передать всего $N + 1 (N \geq n)$ битов, среди которых будут контрольные (необходимо распознать $N+1$ положение ошибки, значит справедливо $2^{N-n} \geq N + 1$)

1. Перенумеруем биты от 1 до N, номера, являющиеся степенями двойки отводятся под контрольные биты.
2. Контрольному биту 2^i сопостовляется множество $p_i = \{j : j \& i = i\}$, где j – номер в коде, & – побитовое И. Другими словами, 2^i-ому контрольному биту сопоставляются номера кода j, такие, что в двоичном представлении j на $\log_2 i$ позиции стоит единица. (порядок нумерации для двоичных чисел)
3. 2^i-ый контрольный бит принимает такое значение, чтобы:

$$\bigoplus_{j \in P_i} a[j] = 0 \text{ – контрольное соотношение}$$

Декодирование: Если все контрольные соотношения сходятся, тогда сообщение передано без ошибок. Отбрасываем контрольные биты и выводим сообщение.

1. если не сошлись соотношения, соответствующие множествам $P_1, P_2, \ldots, P_k$, то берем номер $err = \min\{P_1 \cap P_2 \cap \ldots \cap P_k\}$ — номер с ошибкой.
2. Берем $a[err] = \neg a[err]$
3. если не сошлось всего одно соотношение, соответствующее P_i, тогда контрольный бит передался с ошибкой. Просто отбрасываем его и все контрольные биты и передаем сообщение.

Пример. $a = [1, 0, 0, 1, 1, 1, 1]$

Добавляем контрольные биты, меняем нумерацию и состовляем множества для контрольных битов:

i	1	2	3	4	5	6	7	8	9	10	11
a			1		0	0	1		1	1	1
P_0	1		3		5		7		9		11
P_1		2	3			6	7			10	11
P_2				4	5	6	7				
P_3								8	9	10	11

Вычисляем XOR для $P_0, \ldots, P_3$. соответствующие им контрольные

биты с номерами 1, 2, 4, 8 принимают значения: 0, 0, 1, 1. Тогда код Хэмминга для собощения:

$$a = [0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1]$$

Пусть передали сообщение с ошибкой: 00111**1**011111. Вычисляем контрольные соотношения, если не сошлись с исходыми, то ищем номер, на котором ошибка и заменяем ее: $err = \min\{\{1, 3, \ldots, 11\} \cap \{4, 5, 6, 7\}\} = 5$.

$a[err] = \neg 1 = 0$. Передаваемое сообщение исправлено.

5.7 Шифрование

Определение 36. Сообщение — то, что защищается от несанкционированного доступа.
Криптографический алгоритм — алгоритм, который используется для шифрования или дешифрования исходного сообщения.
Ключ — вспомогательная информация, используемая алгоритмом.

Пример. (Шифрование с закрытым ключом) $\{a_i\}$ – псевдослучайная последовательность битов – ключ
$\{b_i\}$ – исходное сообщение
$\{c_i\}$ – зашифрованное сообщение, полученное следующим образом:

$$\forall i : c_i = a_i \oplus b_i$$

Для шифрования и дешифрования используется один и тот же ключ.

Алгоритм. (RSA)

1. Выбираются два больших простых числа p и q, $n = pq$.
2. Вычисляется функция Эйлера $\varphi(n) = (p-1)(q-1)$.
3. Выбирается целое число $e \in (1, \varphi(n))$, такое, что $\gcd(e, \varphi(n)) = 1$.
4. Вычисляется число d, обратное к e по модулю $\varphi(n)$, то есть такое, что $ed \equiv 1 \pmod{\varphi(n)}$.

Пара (n, e) – открытый ключ, пара (n, d) – закрытый ключ.

Замечание. Потребуется теорема Эйлера и единственность d (следует из линейного представления НОД).

На лекции это было вынесено в 2 отедльные леммы (не знаю зачем)

Алгоритм. (Применение RSA)

- **Шифрование:** Пусть m — шифруемое сообщение. Предположим, что m представим в виде числа от 0 до $n-1$ (иначе разобьем на блоки). Отправитель вычисляет зашифрованное сообщение: $c \equiv m^e \mod n$
- **Дешифрование:** Получатель получает c и вычисляет $c^d \equiv m^{ed} \equiv m \mod n$

Замечание. Асимметричное шифрование — довольно ресурсоемкая процедура, в отличие от симметричного. Как упростить жизнь?

1. Отправитель генерирует ключ сессии b.
2. Исходное сообщение a шифруется симметричным алгоритмом при помощи
3. ключ сессии b шифруется открытым ключом e — получим цифровой конверт x.
4. c и x передаются получателю.
5. Получатель из x восстанавливает b при помощи своего закрытого ключа, а затем восстанавливает a из c при помощи b.

Лекция 12: АВЛ-дерево, хэширование, B-деревья

29.11.2023

Глава 6

Информационный поиск и организация информации

6.1 АВЛ-дерево

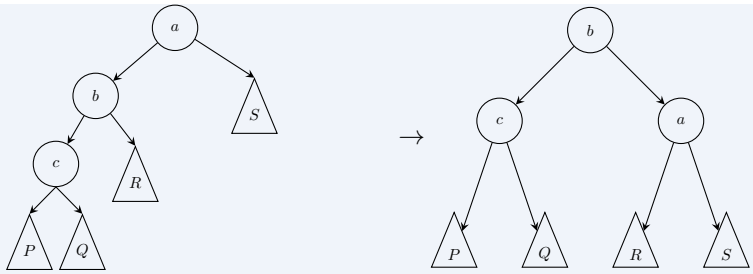
Определение 37. Высота дерева — длина пути (количество ребер) от корня до листьев.
Вершина дерева называется сбалансированной, если высоты ее правого и левого поддеревьев различаются не более чем на 1.
Двоичное дерево называется сбалансированным, если каждая его вершина сбалансирована.
Чтобы при добавлении элементов в дерево время поиска росло не слишком быстро, его нужно балансировать.
АВЛ-дерево (самобалансирующееся дерево) позволяет поддерживать приблизительное равенство двух ветвей двоичного дерева во всех его узлах, затрагивая за раз не более трех из них.

Алгоритм. (Балансировка)

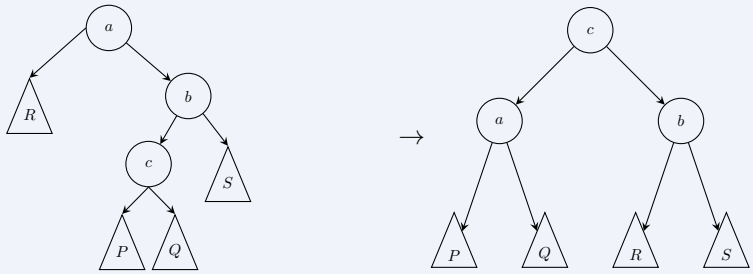
1. добавляем вершину d
2. проверяем вершины на пути от d к корню
3. обозначим через a, b, c (b, c – потомки a) первые несблансированные вершины на пути от d к корню (баланс может нарушится только в трех или в двух вершинах)
4. выполняем перебалансировку (поворот):

2 вида перебалансировки:

1. узел b "вытягиваем"вверх на место a, так, что a становится потомком b (b соотвественно "прикрепляется"к узлу выше, если он был). Поддерево на b становится поддеревом a.



2. узел c "вытягиваем"вверх, a – левый ребенок, b – правый ребенок. Левое поддерево c прикрепляется к a, правое – к b.



Теорема 9. После операции поворота полученное дерево окажется сбалансированным.

Доказательство. (да, я просто списал с Романовского :c) Обозначим за $H(i)$ максимальную длину пути от добавленной вершины до вершины i, за h_k - максимальную высоту поддерева K.

По предположению:

$$H(a) = H(b) + 1 = H(c) + 2,$$

$$h_s = H(b) - 2 = H(c) - 1$$

Рассмотрим отдельно 2 случая:

1. $h_R < H(c)$. Тогда после первой балансировки дерева $H(a) = h_s + 1 = H(c)$ и высота верхней вершины стала меньше
2. в этом случае высота так же уменьшается на 1, так как определяющие максимальный путь поддеревья R и Q поднялись на одну ступень выше.

□

6.2 Хэширование

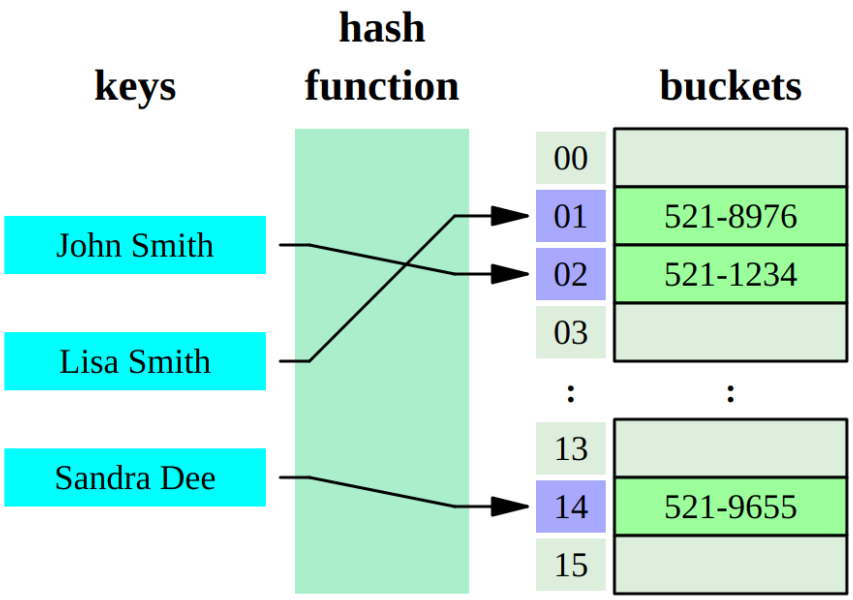
Определение 38. Хеш — это функция, сопоставляющая объектам какого-то множества числовые значения из ограниченного промежутка.

«Хорошая» хеш-функция:

- Быстро считается — за линейное от размера объекта время;
- Имеет не очень большие значения — влезающие в 64 бита;
- «Детерминировано-случайная» — если хеш может принимать n различных значений, то вероятность того, что хеши от двух случайных объектов совпадут, равна примерно $\frac{1}{n}$.

Обычно хеш-функция не является взаимно однозначной: одному хешу может соответствовать много объектов, т.е. она сюръективная.

Пример:



6.3 B-деревья

Определение 39. B-дерево — сильноветвящееся сбалансированное дерево поиска, в котором каждый узел содержит множество ключей и имеет более двух потомков.

Элементы находятся в листьях, остальные уровни представляют собой иерархию индексов, они указывают путь, по какой ветке двигаться, чтобы прийти в тот лист, где находится нужная запись.

Количество ключей в узле и количество потомков зависит от порядка B-дерева. Для каждого дерева фиксируется какое-то число t.

B-дерево обладает следующими свойствами:

1. Все листья находятся на одном и том же уровне, т.е. имеют оди-

наковую глубину (B-дерево идеально сбалансировано)

2. Каждый узел, кроме корневого, должен иметь, как минимум $t-1$, и не более $2t-1$ ключей
3. Если узел не является листом, то он имеет детей (кол-во ключей в узле + 1) штук
4. Все ключи в узле должны располагаться в порядке возрастания их значений

Замечание. Примеры для всех операций в B-дереве очень долго писать, можете потыкать тут (открывайте в pdf)

Алгоритм. (Поиск в B-дереве)

1. Считать элемент для поиска
2. Сравнить искомый элемент с первым значением ключа в корневом узле дерева.
3. Если они совпадают, вернуть значение.
4. Если они не совпадают, проверить больше или меньше значение элемента, чем текущее значение ключа.
5. Если искомый элемент меньше, продолжить поиск по левому поддереву.
6. Если искомый элемент больше, сравнить элемент со следующим значением ключа в узле и повторять Шаги 3, 4, 5 и 6 пока не будет найдено совпадение или пока искомый элемент не будет сравнен с последним значением ключа в узле-листе.
7. Если последнее значение ключа в узле-листе не совпало с искомым, вернуть *null*.

Алгоритм. (Добавление элемента) В B-дереве новый элемент может быть добавлен только в узел-лист. Вставка происходит следующим образом:

1. Проверить пустое ли дерево.
2. Если дерево пустое, создать новый узел с новым значением ключа и его принять за корневой узел.
3. Если дерево не пустое, найти подходящий узел-лист, к которому будет добавлено новое значение, используя логику дерева двоичного поиска.
4. Если в текущем узле-листе есть незанятая ячейка, добавить но-

вый ключ-значение к текущему узлу-листу, следуя возрастающему порядку значений ключей внутри узла.

5. Если текущий узел полон и не имеет свободных ячеек, разделите узел-лист, отправив среднее значение родительскому узлу. Повторяйте шаг, пока отправляемое значение не будет зафиксировано в узле.

6. Если разделение происходит с корнем дерева, тогда среднее значение становится новым корнем дерева и высота дерева увеличивается на единицу.

Алгоритм. (Удаление элемента)

1. Если корень является листом (в дереве только один узел), просто удаляем ключ из этого узла.

2. Находим узел x, содержащий ключ, запоминая путь к нему.

3. Если x — лист, удаляем ключ из x. Если в x осталось не меньше $t-1$ ключей, процедура завершается.

4. Если соседний правый узел имеет не менее t ключей, переносим ключ-разделитель в x и первый ключ соседа на место разделителя. Если соседний правый узел не подходит, но соседний левый узел имеет не менее t ключей, аналогично переносим ключ-разделитель и последний ключ соседа. Если ни один из соседей не подходит, объединяем x с одним из соседей и перемещаем разделяющий ключ в объединенный узел.

5. Если после объединения в родительском узле остается $t-2$ ключей и это не корень, повторяем процедуру для родителя.

6. Если в результате в корне осталось от 1 до $t-1$ ключей, дополнительных действий не требуется. Если в корне не осталось ключей, исключаем корневой узел и делаем его единственного потомка новым корнем дерева.

7. Если x не является листом, удаляем самый правый ключ из поддерева i-го потомка x или самый левый ключ из поддерева $(i+1)$-го потомка x. Заменяем удаленный ключ на место ключа K.