Лабораторная работа № 5

“Исследование организации данных методом расстановки”

1. Программа работы

При подготовке к лабораторной работе

1. Изучить теоретический материал по теме лабораторной работы в размере конспекта лекций и данного пособия и ответить на контрольные вопросы.
2. Заполнить разделы отчета по лабораторной работе.
3. Из таблицы 1 выбрать n=8 чисел в соответствии с формулой (1)

$$N1=S+K(S \bmod 2+1), \qquad (1)$$

где $N1$ – номер первого ключа последовательности, S – номер по списку, К – порядковий номер метода открытой адресации для разрешения коллизий в соответствии с таблицей 2.

4. Создать таблицу расстановки последовательности ключей из пункта 3 с помощью методов открытой адресации в соответствии с таблицей 2 и хеш-функцией $H(k)=k \bmod N$, N=13.
5. Написать программу, которая создаёт таблицу расстановки последовательности ключей с помощью хеш-функции $H(k)=k \bmod N$ и разрешает коллизии с помощью заданных в таблице 2 методов открытой адресации.

При выполнении лабораторной работы

1. Отладить программу исследования данных методом расстановки.
2. Определить для исследуемых методов открытой адресации из таблицы 2 среднее число обращений к таблице расстановки при вставке и среднее число обращений при результативном поиске как функцию от коэффициента заполнения таблицы расстановки $\alpha=n/N$, (α={0.25, 0.50, 0.75, 0.90, 0.95}).
3. В качестве последовательности ключей использовать целые числа, равномерно распределенные в интервале $[A, B]$. Для формирования последовательности ключей использовать датчик равномерно распределенных чисел *rand*().
4. На основе проведенных исследований обобщить полученные результаты, сделать выводы по работе, оформить отчет по лабораторной работе и представить его преподавателю на защиту.

2. Оформление отчета

Письменный отчет по лабораторной работе должен содержать:

1. Название, цель и программу выполнения лабораторной работы.
2. Фамилию, имя, отчество, номер группы исполнителя, дату сдачи.
3. Математическую постановку задачи (основные расчетные формулы и соотношения).
4. Выполняемый вариант и его содержание.
5. Диаграмму классов проекта.
6. *Распечатку подпрограмм* методов открытой адресации и кода интерфейса проекта для исследования методов открытой адресации (*обязательны комментарии* к программе).

7. Расставляемые *последовательности ключей* по пункту 1.3.
8. *Таблицы* расстановки последовательности ключей по пункту 1.4.
9. *Таблицу* зависимости среднего количества обращений S_{rv} при вставке как функцию от коэффициента заполнения таблицы α для исследуемых методов открытой адресации из таблицы 2.
10. *Таблицу* зависимости среднего количества обращений S_{rp} при результативном поиске как функцию от коэффициента заполнения таблицы α для исследуемых методов открытой адресации из таблицы 2.
11. *График* зависимости среднего количества обращений S_{rv} при вставке как функцию от коэффициента заполнения таблицы α для исследуемых методов открытой адресации из таблицы 2.
12. *График* зависимости среднего количества обращений S_{rp} при результативном поиске как функцию от коэффициента заполнения таблицы α для исследуемых методов открытой адресации из таблицы 2.
13. Привести результаты тестирования проекта.
14. *Выводы* по лабораторной работе в соответствии с программой работы.

3. Контрольные вопросы

1. Каково определение и основные особенности прямой адресации?
2. Каково определение и основные особенности хеширования?
3. Каковы требования к хеш-функции?
4. В чем сущность метода середины квадрата для выбора хеш-функции?
5. В чем сущность метода деления с остатком для выбора хеш-функции?
6. В чем сущность метода свертки для выбора хеш-функции?
7. В чем сущность мультипликативного метода для выбора хеш-функции?
8. Какие группы методов существуют для разрешения коллизий?
9. В чем сущность метода линейных проб для разрешения коллизий?
10. В чем сущность метода квадратичных проб для разрешения коллизий?
11. В чем сущность метода двойного хеширования для разрешения коллизий?
12. В чем сущность метода двух аргументов для разрешения коллизий?
13. В чем сущность метода цепочек для разрешения коллизий?
14. Какова сложность операций поиска, вставки и удаления при организации данных методом расстановки?

4. **Литература**

1. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.- М.: Мир, 1979.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы..- М.: ‘Вильямс’, 2000.-384с.
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ.-М.:МЦНМО, 2000.-960с.
4. Вирт Н. Алгоритмы и структуры данных.- М.: Мир, 1989.
5. Кнут Д. Искусство программирования для ЭВМ. Т.3.Сортировка и поиск. -М.: Мир, 1979.
6. Лэнгсам И., Огенстайн М., Тененбаум А. Структуры данных для персональных ЭВМ. -М.: Мир, 1989.
7. Рейнгольд Э., Нивергельд Ю., Део Н. Комбинаторные алгоритмы.-М.: Мир, 1980.
8. Проценко В.С. та ін. Техніка програмування мовою Сі: Навч.Посібник,-К.:Либідь, 1993.-224с.
9. Мейер Б., Бодуэн К. Методы программирования. Т2.-М.:Мир, 1982.-368с.

Таблица 1 130 равномерно распределенных чисел

	1	2	3	4	5	6	7	8	9	10
1	10	09	73	25	33	76	52	01	35	86
2	34	67	35	48	76	37	54	20	48	05
3	64	89	47	42	96	24	80	52	40	37
4	08	42	26	89	53	19	64	50	93	03
5	23	20	90	25	60	99	01	90	25	29
6	89	37	67	07	15	38	31	13	11	65
7	12	80	79	99	70	80	15	73	61	47
8	04	03	23	66	53	80	95	90	91	17
9	39	29	27	49	45	66	06	57	47	17
10	20	63	61	04	02	00	88	29	16	65
11	31	06	01	08	05	15	95	33	47	64
12	35	08	03	36	06	85	26	97	76	02
13	88	67	67	43	97	04	43	62	76	59

Таблица 2 Варианты исследуемых методов открытой адресации

Вариант	*Интервал распределения чисел [A, B]*	*Размерность таблицы N*	*Варианты методов открытой адресации*
1.	[0,3000]	41	1, 3, 6
2.	[0,4000]	43	2, 3, 4
3.	[0,5000]	53	4, 5, 6
4.	[1000,3000]	79	1, 3, 5
5.	[0,1000]	83	2, 3, 4
6.	[1000,2000]	89	1, 4, 5
7.	[0,100]	97	3, 5, 6
8.	[0,200]	101	2, 3, 4
9.	[10,300]	103	1, 4, 5
10.	[0,400]	107	1, 3, 5
11.	[200,500]	109	2, 3, 4
12.	[0,600]	113	1, 4, 5
13.	[0,700]	139	1, 3, 5
14.	[0,800]	149	3, 4, 6
15.	[100,900]	151	1, 4, 5
16.	[0,999]	157	2, 3, 6
17.	[0,500]	161	2, 3, 4
18.	[0,700]	163	4, 5, 6
19.	[0,900]	167	1, 3, 5
20.	[0,1100]	171	2, 3, 6

Варианты методов открытой адресации:

1) метод линейных проб ($H_i = (H_0+i) \bmod N$);
2) метод линейных проб ($H_i = (H_{i-1}+c) \bmod N$);
3) метод квадратичных проб;
4) метод двойного хеширования;
5) метод двух аргументов;
6) случайный метод.

Методика определения среднего числа обращений к таблице при вставке

1. Сформировать случайным образом n ключей из диапазона [A, B].
2. Заполнить таблицу расстановки случайным образом на α процентов.
3. Произвести P=50 вставок случайно выбранных из диапазона [A, B] ключей (ключ вставить, а затем удалить), каждый раз фиксируя число обращений OV_i к таблице.
4. Определить среднее число обращений при вставке S_{rv} по формуле $$Srv = \frac{\sum_{i=1}^{P} OV_i}{P}.$$

Методика определения среднего числа обращений к таблице при результативном поиске

1. Сформировать случайным образом n ключей из диапазона [A, B].
2. Заполнить таблицу расстановки на α процентов ключами из п. 1.
3. Случайным образом выбрать один из n ключей из п.1.
4. Определить число обращений к таблице при поиске ключа OP_i из п. 3.
5. Пункты 1, 2, 3, 4 проделать P=50 раз.
6. Определить среднее число обращений при результативном поиске S_{rp} по формуле
$$Srp = \frac{\sum_{i=1}^{P} OP_i}{P}$$

Лабораторная работа № 5

“Исследование организации данных методом расстановки”

Задание (*более сложное*):

1. Провести сравнительную оценку всех 6-ти методов открытой адресации (метода линейных проб (два варианта), метода квадратичных проб, метода двойного хеширования, метода двух аргументов, случайного метода) по методике Лр № 5.

Краткий теоретический материал по теме **“Организация данных методом расстановки”**

Прямая адресация - применима, если количество возможных ключей невелико. Пусть возможными ключами являются числа из множества $S=\{0,1,\dots,N-1\}$. Пусть также, ключи всех элементов различны. Для хранения множества S используют массив $T[0,\dots, N-1]$, называемый таблицей с прямой адресацией. Каждая ячейка таблицы T соответствует определенному ключу из множества S: $T[k]$- место, предназначенное для записи указателя на элемент с ключом k; если элемента с ключом k в таблице T нет, то $T[k]=NIL$. В таблице T выполняются операции: *Поиск*, *Включение*, *Удаление*, каждая из этих операций требует времени $O(1)$.

Недостатки прямой адресации: 1) если мощность множества ключей S велика, то хранить в оперативной памяти массив T размером $\|S\|$ непрактично, а то и невозможно; 2) если число реально используемых ключей невелико по сравнению с $\|S\|$, то много памяти тратится неэффективно; 3) невозможно динамически расширять такую таблицу.

Если количество записей в таблице существенно меньше, чем количество всевозможных ключей, то говорят о хешировании, причем хеш-таблица занимает гораздо меньше места, чем таблица с прямой адресацией.

Хеш-таблица требует памяти порядка $\Theta(\|k\|)$, а среднее время поиска записи - $O(1)$.

При прямой адресации элементу с ключом k отводится позиция с номером k, при хешировании же этот элемент записывается в позицию номер $H(k)$ в хеш-таблице $T[0,\dots,N-1]$.

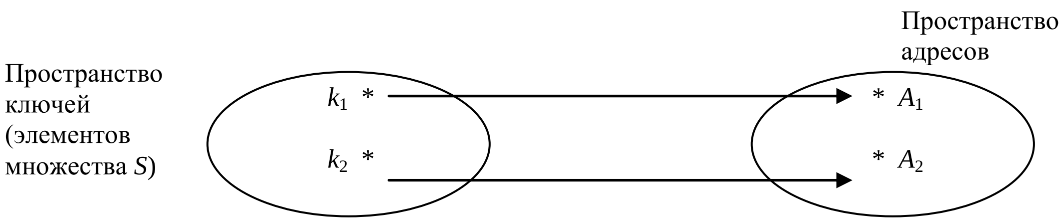
При $\|S\| > N$ коллизии неизбежны, необходимо быть готовым обработать коллизии.

Достоинство метода расстановки. При реализации словаря с помощью массива выполнение операций П, В, У имеет сложность $O(N)$, где N − размерность словаря, т.е. массива. При реализации словаря методом хеширования (таблица расстановки) операции П, В, У имеют фиксированную сложность $O(1)$ (в среднем).

1. Организация данных методом расстановки

Метод расстановки в идеальном случае предполагает, что для представления множества S используется таблица расстановки T, задаваемая с помощью массива указателей A из N элементов, элементы массива A удобно нумеровать от 0 до (N-1). Причем N − общее число возможных ключей, т.е. $N = n = \|S\|$ − мощность множества S.

Предполагается, что существует функция расстановки (хеш-функция) $H=H(k)$, ставящая в соответствие всякому ключу $k=a\in S$ некоторое целое i, различное для разных k и такое, что $0 \le i \le N$-1. Тогда достаточно разместить элемент $a = k$, обладающий этим ключом в $A[i]$, и время поиска становится постоянным − $O(1)$.



Однако, данная идеальная схема практически нереализуема, т.к. число возможных ключей велико, а пространство адресов, обычно, ограничено несколькими тысячами, если таблица должна оставаться в оперативной памяти.

Значит, функция $H(k)$ должна давать самое большее N различных значений, где N много меньше числа теоретически возможных ключей. Такая функция, следовательно, будет не инъективной, т.е. два различных ключа могут иметь одинаковые значения. Будем говорить, что два ключа k_1 и k_2, таких, что

$k_1 \neq k_2$ и $H(k_1)=H(k_2)$ вызывают <u>коллизию</u> (конфликт). Наличие коллизий повышает время выполнения операций В(a, S), У(a, S), П(a, S). Наличие коллизий обязывает сдержанно относиться к использованию имитации "прямого доступа".

<u>Выбор функции расстановки *H* (хеш-функции)</u>

Требования к функции расстановки:
1. аргументами функции $H(k)$ являются ключи множества S;
2. простота вычисления функции $H(k)$;
3. равномерное распределение элементов $a \in S$ по таблице;
4. минимизация коллизий;
5. отображение элементов множества S в множество целых чисел от 0 до (N−1).

Для выбора хеш-функции используют следующие <u>методы</u>:

1. <u>Метод середины квадрата</u> − ключ умножается сам на себя и в качестве индекса массива используется несколько средних цифр этого квадрата. Причиной возведения числа в квадрат до извлечения средних цифр является то, что все цифры первоначального числа дают свой вклад в значение средних цифр квадрата.

<u>Пример</u>: пусть k =15, тогда k*k =15*15=$0\boxed{22}5$, то есть k =15 → $H(k)$=22.

Однако метод середины квадрата дает не всегда вполне равномерное распределение значений между 000 и 999.

<u>Пример</u>: 2*2=$0\boxed{00}4$ и 3*3=$0\boxed{00}9$ ← коллизия.

Пусть ключи являются целыми числами из интервала 0, 1,…, p. Введем целую константу C такую, что $NC^2 \cong p^2$, тогда функция

$H(k) = [k^2/C] \bmod N$,

где [*] обозначает целую часть *, эффективно извлекает из середины числа k^2 цифры, составляющие число, не превышающее N.

<u>Пример</u>: если количество ключей p = 1000, размерность массива N = 8, то можно так выбрать константу C = 354 (C=$(p^2/N)^{-2}$=$(1000^2/8)^{-2} \cong 354$).Тогда при k=456

$$H(456) = \left[\frac{207\ 936}{354}\right] \bmod 8 = 587 \bmod 8 = 3.$$

Или при k=223 − $H(223) = \left[\frac{49729}{354}\right] \bmod 8 = 140 \bmod 8 = 4.$

2. Метод деления с остатком − целый ключ k или функция от него $\varphi(k)$ делится на размер таблицы N и остаток от деления берется в качестве значения хеш-функции.

$H(k) = \varphi(k) \bmod N$,

или $H(k) = mod\ (\varphi(k),\ N)$, где $\varphi(k)$ - функция, принимающая целые значения. В частности $\varphi(k) = k$.

Существует два вида коллизий:

- непосредственные коллизии, соответствующие ключам k_1 и k_2, $k_1 \neq k_2$, таким, что $\varphi(k_1) = \varphi(k_2)$;
- коллизии по модулю, порождаемые ключами k_1 и k_2, такими, что $\varphi(k_1) \neq \varphi(k_2)$, но $\varphi(k_1) = \varphi(k_2) \bmod N$.

Доля коллизий "по модулю" уменьшается, если размером таблицы N выбрать целое N, не имеющее делителей меньших 20.

Плохой выбор числа возможных хеш-значений N таков:

1. $N=2^p$, тогда $H(k)$ - это p младших битов числа k. Если нет уверенности, что все комбинации младших битов распределены с одинаковой частотой, то N не должно быть $= 2^p$.
2. $N=10^p$, тогда часть цифр ключа k полностью определяют хеш-значение $N=10^1$ $k=1\underline{5} \rightarrow H(k)=5$ $k=11\underline{5} \rightarrow H(k)=5$
3. Если ключами являются числа в системе счисления с основанием 2^p, то нехорошо брать $N=2^p-1$, т.к. при этом одинаковые хеш-значения имеют ключи, отличающиеся лишь перестановкой "2^p- ичных цифр".

Хороший результат дает следующий выбор → N − простое число, далеко отстоящее от степени двойки, то есть от 2^p.

Пример: Пусть множество ключей – $\{k\}=\{5, 4, 3, 8, 10, 25\}$, $\varphi(k) = k$, функция расстановки - $H(k) = k \bmod 20$ (это N=20 для практики нереально).

3. Метод свертки − ключ разбивается на несколько сегментов, над которыми выполняется, например, операция сложения по mod 2 (при возможности двоичного представления ключа) (в общем случае операция не тождественности) для формирования хеш-функции.

Пример: пусть внутреннее представление ключа имеет вид k=01011|10010|10110, для индекса отводится 5 разрядов. Над тремя последовательностями разрядов 01011,10010 и 10110 выполняется операция не тождественности, что дает $(01111)_2=(15)_{10}$ $H(k)=(01111)_2=(15)_{10}$

Сложение по *mod* 2 выполняется следующим образом

01011

10010

10110

$(01111)_2=(15)_{10}$

4. Мультипликативный метод (метод умножения)

Хеш-функция имеет вид $H(k) = \lfloor N((k \cdot A) \bmod 1) \rfloor$,

где N − количество хеш-значений, k – значение ключа, A − константа, $0 < A < 1$,

$(k\,A) \bmod 1$ − дробная часть $k\,A$, $\lfloor \cdot \rfloor$ − выделение меньшего целого.

Значения константы A зависят от рода данных, подвергаемых хешированию, некоторые значения A могут быть лучше других.

Значение $A \approx \left(\sqrt{5}-1\right)/2 = 0{,}6180339887.....$ является довольно удачным [Кнут].

Достоинство метода умножения в том, что качество хеш-функции мало зависит от выбора N. Обычно в качестве N выбирают степень 2, так как в этом случае в ЭВМ умножение на такое N реализуется как сдвиг слова. Пусть длина слова ЭВМ w битов и ключ k помещается в одно слово. Тогда, если $N = 2^p$, то вычисление $H(k)$ проводится так: ключ k, представленный в виде ω-битового числа, умножается на ω-битовое целое число $A\,2^{\omega}$, получится 2ω-битовое число. У произведения берут младшие ω битов (это число r_0), а из этих ω битов выделяют p старших битов. Это и есть хеш-значение $H(k)$.

Пример: $k = 123456$, $N = 10000$, $A = 0{,}61803\ldots$

$H(k) = \lfloor 10000 \cdot (123456 \cdot 0{,}61803.. \bmod 1) \rfloor =$

$= \lfloor 10000 \cdot (76300{,}0041151.. \bmod 1) \rfloor =$

$= \lfloor 10000 \cdot 0{,}0041151.. \rfloor =$

$= \lfloor 41{,}151.. \rfloor = 41.$

Такая функция $H(k)$ хороша как рассеивающая функция, приводящая к рандомизации ключей.

5. Метод хеширования, основанный на алгебраической теории кодирования − идея аналогична методу деления по *mod N*, только вместо деления на целое число N используется деление на многочлен по *mod* 2.

2. Методы разрешения коллизий

Для разрешения коллизий используются две группы методов: – методы открытой адресации и – метод цепочек.

2.1. Методы открытой адресации - внутренне разрешение в самой таблице.

При возникновении коллизии она разрешается в самой таблице (массиве) посредством вставки элемента в еще свободное место массива *A*.

Чтобы включить элемент с ключом *k*, выбирается позиция массива *A* с индексом $H(k)=i$ если $A[i]=$*ПУСТО*, если же $A[i]\neq$*ПУСТО* то нужно попытаться вставить элемент с ключом *k* в следующие свободные ячейки массива *A* Рассматриваемые методы называются повторным хешированием или открытой адресацией.

2.1.1. Метод линейных проб

Используется функция повторного хеширования $RH = H_i,\ i=\overline{1,N-1},$ которая воспринимает один индекс в массиве и выдает другой индекс. Если ячейка массива $H(k)$ уже занята некоторой записью с другим ключом, то функция RH применяется к значению $H(k)$ для того, чтобы найти другую ячейку, куда может быть помещена эта запись. Если ячейка *RH(H(k))* также занята, то хеширование повторяется еще раз и проверяется ячейка *RH(RH(H(k)))*. Этот процесс повторяется до тех пор, пока не будет найдена пустая ячейка или же окончится массив *A*.

В случае метода линейных проб функция повторного хеширования имеет вид:

$H_0 = H(k)$

$H_i = (H_0+i) \bmod N,\ i=\overline{1,N-1}$

или $H_i = (H_{i-1}+c) \bmod N$, где *c* − константа, такая, что *c* и *N* взаимно просты.

Свойство хорошей функции повторного хеширования состоит в том, что для любого ключа *k* последовательно выполняемые повторные хеширования располагаются на максимально возможное число целых чисел от 0 до (*N*-1).

Любая функция вида (2) удовлетворяет этому свойству.

Недостаток функций повторного хеширования вида (1) и (2) состоит в возможности группирования (скучивания) вокруг первичных ключей.

Явление группирования (скучивания) состоит в том, что два ключа, которые хешируются в разные значения, конкурируют друг с другом при повторных хешированиях.

Пример: $H_0 = H(k) = k \bmod N$

$H_i = (H_{i-1}+c) \bmod N$

$N = 1000$

$C = 21$, *C* и *N* взаимно просты

Если позиции 10, 31, 52, 73 и 94 заняты, то любая запись, чей ключ являлся одним из этих чисел, будет помещена в позицию 115.

$H_0=H(10) \bmod N=10$

$H_1=(H_0 +C) \bmod N=(10 +21)=31$

$H_2=(H_1 +C) \bmod N=(31 +21)=52$

$H_3=(H_2 +C) \bmod N=(52 +21)=73$

$H_4=(H_3 +C) \bmod N=(73 +21)=94$

$H_5=(H_4+C)\ mod\ N=(94+21)=115$

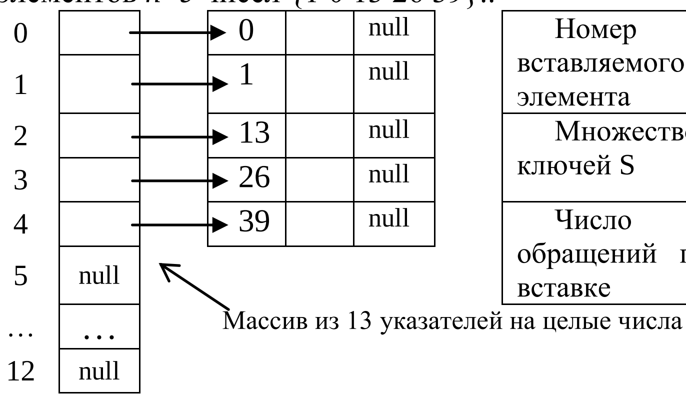
В действительности любая функция, которая повторяет хеширование зависящее только от индекса, будет вызывать скучивание.

От этого недостатка свободны следующие функции (и методы) повторного хеширование.

Пример. Метод линейных проб - $H_i = (H_0+i)\ mod\ N$

$H_0 = H(k) = k\ mod\ N,\ H_i = (H_0+i)\ mod\ N$

Операция вставки. Пусть размерность массива N=13, число вставляемых элементов n=5 чисел {1 0 13 26 39}..



Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	3	4	5

Вставка ключа k=1.
$H_0=H(1)=k\ mod\ N=1\ mod\ 13=1$ Число обращений при вставке 1 = 1.
Вставка ключа k=0.
$H_0=H(0)=0$ Число обращений при вставке 0 = 1.
Вставка ключа k=13.
$H_0=H(13)=13\ mod\ 13=0$ – коллизия, на этом месте уже есть элемент.
$H_1=(H_0+1)=1\ mod\ 13=1$– коллизия, на этом месте уже есть элемент.
$H_2=(H_0+2)=2\ mod\ 13=2$ Число обращений при вставке 13 = 3.
Вставка ключа k=26.
$H_0=H(26)=26\ mod\ 13=0$
$H_1=(H_0+1)=(H_0+1)=1\ mod\ 13=1$
$H_2=(H_0+2)=2\ mod\ 13=2$
$H_3=(H_0+3)=3\ mod\ 13=3$ Число обращений при вставке 26 = 4.
Вставка ключа k=39.
$H_0=H(39)=39\ mod\ 13=0$
$H_1=(H_0+1)=1\ mod\ 13=1$
$H_2=(H_0+2)=2\ mod\ 13=2$
$H_3=(H_0+3)=3\ mod\ 13=3$
$H_4=(H_0+4)=4\ mod\ 13=4$ Число обращений при вставке 39 = 5.

Среднее число обращений при вставке = (1+1+3+4+5)/5=14/5=2,8.

Коэффициента заполнения таблицы $\alpha=n/N=5/13$.

Результативный поиск при методе линейных проб $H_i = (H_0+i)\ mod\ N$,
Поиск ключа k=1.

H(1) = 1 A(1)=1=k=1? Да   Число обращений при результативном поиске 1 = 1.
Поиск ключа *k*=0.
H(0) = 0 A(0)=0=k=0? Да
Поиск ключа *k*=13.
H(13) = 0 A(0)=0=k=13? Нет
H_1=(H_0+1) = 1 A(1)=1=k=13? Нет
H_2=(H_1+2) = 2 A(2)=13=k=13? Да
Поиск ключа *k*=26.
H(26) = 0 A(0)=0=k=26? Нет
H_1=(H_0+1) = 1 A(1)=1=k=26? Нет
H_2=(H_1+2) = 2 A(2)=13=k=26? Нет
H_3=(H_2+3) = 3 A(3)=26=k=26? Да

Номер элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при поиске	1	1	3	4	5

Число обращений при результативном поиске 26 = 4.

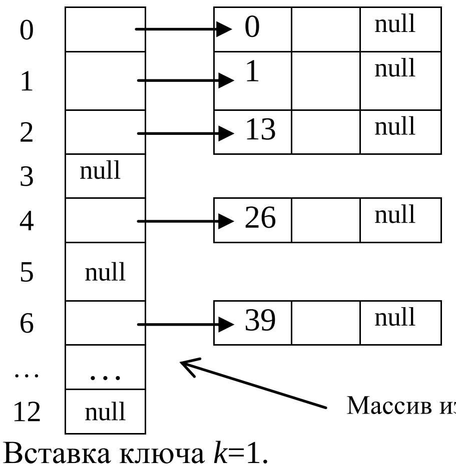
Поиск ключа *k*=39.
H(39) = 0 A(0)=0=k=39? Нет
H_1=(H_0+1) = 1 A(1)=1=k=39? Нет
H_2=(H_1+2) = 2 A(2)=13=k=39? Нет
H_3=(H_2+3) = 3 A(3)=26=k=39? Нет
H_4=(H_3+4) = 4 A(4)=39=k=39? Да   Число обращений при результативном поиске 39 = 5.

Среднее число обращений при поиске данных ключей = (1+1+3+4+5)/5=14/5=2,8.

Этот пример показывают технику вставки и результативного поиска.

Пример. Метод линейных проб - $H_i = (H_{i-1}+c) \ mod \ N$

Вставка (при *N*=13, *c*=2) *n*=5 чисел {1 0 13 26 39}, *N* и *c* взаимно просты.



Массив из 13 указателей на целые числа

Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	2	3	4

Вставка ключа *k*=1.
H_0=H(1) =*k mod N*=1 *mod* 13= 1   Число обращений при вставке 1 = 1.
Вставка ключа *k*=0.
H_0=H(0) = 0   Число обращений при вставке 0 = 1.
Вставка ключа *k*=13.
H_0=H(13) = 13 *mod* 13=0
H_1=(H_0+c) =(H_0+2) = 2 *mod* 13=2   Число обращений при вставке 13 = 2.
Вставка ключа *k*=26.
H_0=H(26) = 26 *mod* 13=0
H_1=(H_0+c) =(H_0+2) = 2 *mod* 13=2
H_2=(H_1+2) = 4 *mod* 13=4   Число обращений при вставке 26 = 3.

Вставка ключа k=39.
H_0=H(39) = 39 mod 13=0
H_1=(H_0+c) =(H_0+2) = 2 mod 13=2
H_2=(H_1+2) = 4 mod 13=4
H_3=(H_2+2) = 6 mod 13=6 Число обращений при вставке 39 = 4.

Среднее число обращений при вставке = (1+1+2+3+4)/5=11/5=2,2.
Коэффициента заполнения таблицы α=n/N=5/13.

2.1.2. Метод квадратичных проб

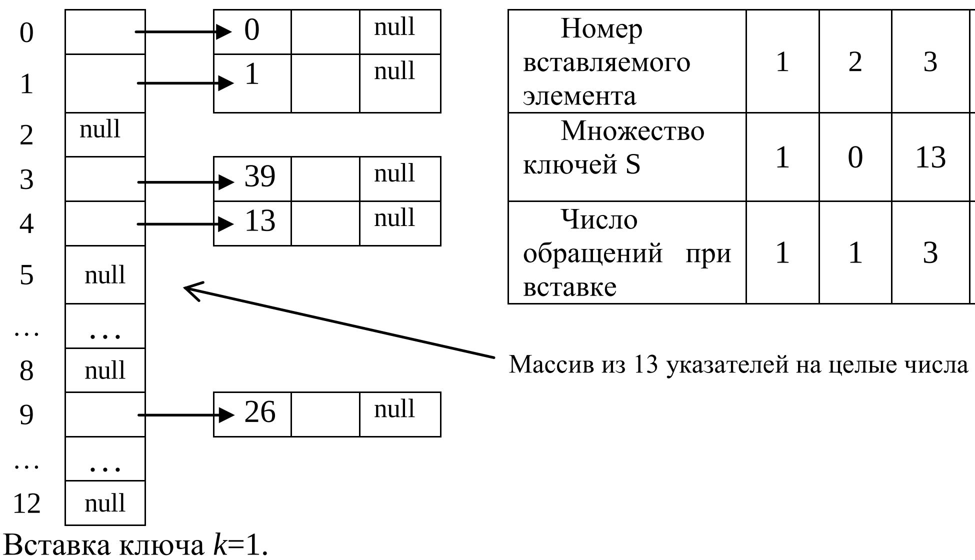
$H_0 = H(k)$
$H_i = (H_0+i^2) \ mod \ N, \ \ i > 0$ (или в общем виде $H_i = (H_0+c_1 \cdot i + c_2 \cdot i^2) \ mod \ N$ ($c_2 \neq 0$))

Недостаток: при поиске пробуются не все строки таблицы, то есть при включении элемента может не найтись свободного места, хотя на самом деле оно есть.

Если N − простое число, то при квадратичных пробах просматривается по крайней мере половина таблицы. Этот недостаток не столь существенен, так как N/2 вторичных попыток при разрешении конфликта встречаются очень редко − когда таблица почти заполнена.

Пример. Метод квадратичных проб - $H_i = (H_0+i^2) \ mod \ N$

Вставка при N=13 n=5 чисел {1 0 13 26 39}.



Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	3	4	5

Вставка ключа k=1.
H_0=H(1) =k mod N=1 mod 13= 1 Число обращений при вставке 1 = 1.
Вставка ключа k=0.
H_0=H(0) =0 mod 13= 0 Число обращений при вставке 0 = 1.
Вставка ключа k=13.
H_0=H(13) = 13 mod 13=0
H_1=(H_0+1^2) =(H_0+1) = 1 mod 13=1
H_2=(H_0+2^2) =(H_0+4) = 4 mod 13=4 Число обращений при вставке 13 = 3.
Вставка ключа k=26.

H_0=H(26) = 26 *mod* 13=0
H_1=(H_0+1^2) =(H_0+1) = 1 *mod* 13=1
H_2=(H_0+2^2) = 4 *mod* 13=4
H_3=(H_0+3^2) = 9 *mod* 13=9 Число обращений при вставке 26 = 4.
Вставка ключа *k*=39.
H_0=H(39) = 39 *mod* 13=0
H_1=(H_0+1^2) =(H_0+1) = 1 *mod* 13=1
H_2=(H_0+2^2) = 4 *mod* 13=4
H_3=(H_0+3^2) = 9 *mod* 13=9
H_4=(H_0+4^2) = 16 *mod* 13=3 Число обращений при вставке 39 = 5.

Среднее число обращений при вставке = (1+1+3+4+5)/5=14/5=2,8.

Коэффициента заполнения таблицы α=n/*N*=5/13.

2.1.3. Метод двойного хеширования − используются две хеш-функции *H(k)* и $H^1(k)$. Идея метода состоит в том, что чтобы избегать коллизии "по модулю", берут в качестве последовательно просматриваемых адресов следующие адреса.

$H(k)+H^1(k)$, $H(k)+2H^1(k),\ldots,$ причем все эти значения вычислены по *mod* *N*.

То есть функция повторного хеширования имеет вид

$H_i=(H(k)+iH^1(k))\ mod\ N,\ i>0$

где $H(k) =\varphi(k) mod\ N$ − исходная хеш-функция,

$H^1(k) =\varphi(k) mod\ L$, причем *L* - взаимно простое с *N*, проще всего взять *L=N-2*. В этом случае говорят о методе двойного совпадения, из него можно извлечь 2 преимущества:

- последовательность адресов $(H(k)+iH^1(k))\ mod\ N,\ i=\overline{0,N-1}$ включает все значения от 0 до (*N*-1);
- на уровне вторичных адресов исключается коллизии по модулю, т.е. случаи где $k \neq k^1, \varphi(k) \neq \varphi(k^1),\ но\ \varphi(k) = \varphi(k^1) \bmod N$.

Действительно, в этом случае получают

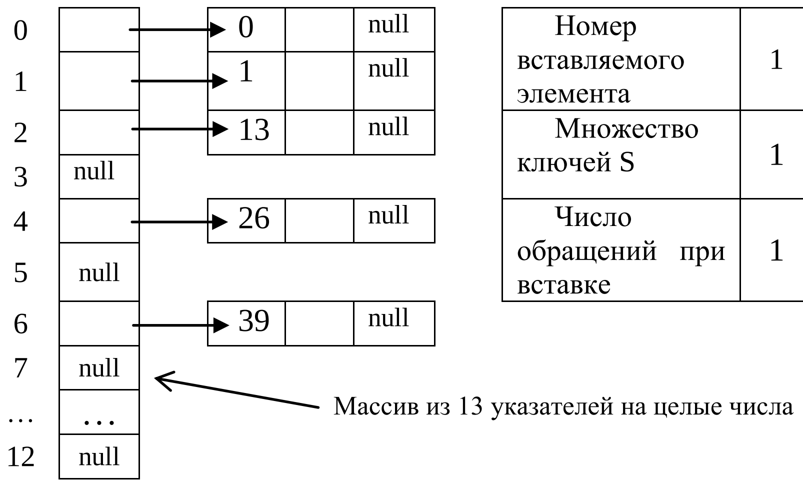
$$\begin{cases} \varphi(k) = \varphi(k^1) \bmod N \\ \varphi(k) = \varphi(k^1) \bmod (N-2) \end{cases}$$

только если $\varphi(k)=\varphi(k^1) mod\ N\ (N-2)$, что мало вероятно.

Пример. Метод двойного хеширования

$H(k)=k\ mod$ 13, $H'(k)= k\ mod$ 11, H_i = ($H(k)+i\cdot H'(k)$) *mod* 13.

Вставка *n*=5 чисел {1 0 13 26 39}.



Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	2	2	2

Вставка ключа k=1.
H(1) =k *mod* N=1 *mod* 13= 1 Число обращений при вставке 1 = 1.
Вставка ключа k=0.
H(0) =0 *mod* 13= 0 Число обращений при вставке 0 = 1.
Вставка ключа k=13.
H(13) = 13 *mod* 13=0
$H_1=(H(k)+i\cdot H'(k))$ *mod* 13= $(H(13)+1\cdot H'(13))$ *mod* 13 = (0+2) *mod* 13=2
Число обращений при вставке 13 = 2.
Вставка ключа k=26.
H(26) = 26 *mod* 13=0
$H_1=(H(26)+1\cdot H'(26))$ *mod* 13 =(0+26 *mod* 11) *mod* 13= 4 *mod* 13=4
Число обращений при вставке 26 = 2.
Вставка ключа k=39.
H(39) = 39 *mod* 13=0
$H_1=(H(39)+1\cdot H'(39))$ *mod* 13 =(0+39 *mod* 11) *mod* 13= 6 *mod* 13=6
Число обращений при вставке 39 = 2.

Среднее число обращений при вставке = (1+1+2+2+2)/5=8/5=1,6.

Коэффициента заполнения таблицы α=n/N=5/13.

2.1.4. Метод двух аргументов − функцию повторного хеширования делают зависящей от числа раз, которые эта функция применяется к некоторому конкретному значению хеширования. Функция *RH* при этом является функцией от двух аргументов - $RH(i,j) = f(H(k), j)$, функция $RH(i, j)$ выполняет повторное хеширование целого числа $i= H(k)$, если для данного ключа повторное хеширование выполняется j-й раз.

Одним примером такой функции является функция:

- 1-е повторное хеширование $RH(i, 1) = (H(k)+1) \bmod N$,
- 2-е повторное хеширование

$$RH(i, 2)= (RH(i, 1)+2) \bmod N =(H(k)+1+2) \bmod N =(H(k)+\sum_{j=1}^{2} j) \bmod N,$$

- p-е повторное хеширование

$$RH(i, p)= (RH(i, p-1)+p) \bmod N =(H(k)+\sum_{j=1}^{p} j) \bmod N.$$

Пример. Метод двух аргументов

$RH(i, j) = f(H(k), j)$, где $i=H(k)$, j – номер повторного хеширования

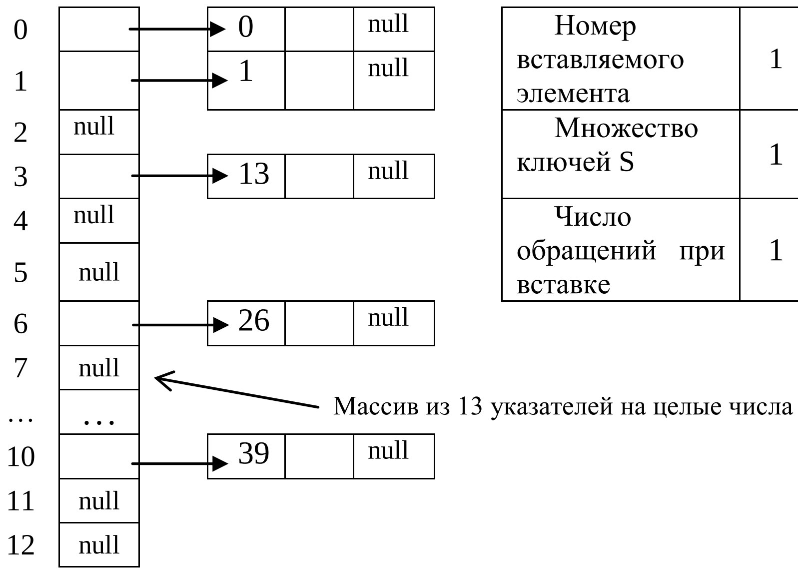
1-е повторное хеширование - $RH(i, 1) = (H(k)+1) \bmod N$
2-е повторное хеширование

$$RH(i, 2) = (RH(i, 1)+2) \bmod N = (H(k)+1+2) \bmod N = (H(k)+\sum_{j=1}^{2} j) \bmod N$$

p-е повторное хеширование

$$RH(i, p) = (RH(i, p-1)+p) \bmod N = (H(k)+\sum_{j=1}^{p} j) \bmod N$$

Вставка n=5 чисел {1 0 13 26 39}.



Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	3	4	5

Вставка ключа k=1.
H(1) =k *mod* N=1 *mod* 13= 1 Число обращений при вставке 1 = 1.
Вставка ключа k=0.
H(0) =0 *mod* 13= 0 Число обращений при вставке 0 = 1.
Вставка ключа k=13.
H(13) = 13 *mod* 13=0
RH(H(k), 1)=(H(13) +1) *mod* 13=(0 +1) *mod* 13 =1
Так как H(k)=0, RH(0, 2)=(0+1+2) *mod* 13=3 Число обращений при вставке 13 = 3.
Вставка ключа k=26.
H(26) = 26 *mod* 13=0
RH(0, 1)=(0+1) *mod* 13=1
RH(0, 2)=(0+1+2) *mod* 13=3
RH(0, 3)=(0+1+2+3) *mod* 13=6 Число обращений при вставке 26 = 4.
Вставка ключа k=39.
H(39) = 39 *mod* 13=0
RH(0, 1)=(0+1) *mod* 13=1
RH(0, 2)=(0+1+2) *mod* 13=3
RH(0, 3)=(0+1+2+3) *mod* 13=6
RH(0, 4)=(0+1+2+3+4) *mod* 13=10 Число обращений при вставке 39 = 5.

Среднее число обращений при вставке = (1+1+3+4+5)/5=14/5=2,8.

Коэффициента заполнения таблицы α=n/N=5/13.

Случайный метод разрешения коллизий

Чтобы избежать группирующего свойства линейного хеширования используют "случайную" методику $H_i=(H_0(k)+d_i) \bmod N$, где $d_1, d_{2,\dots} d_{N-1}$ − случайные перестановки чисел 0, 1, 2, …, N-1.

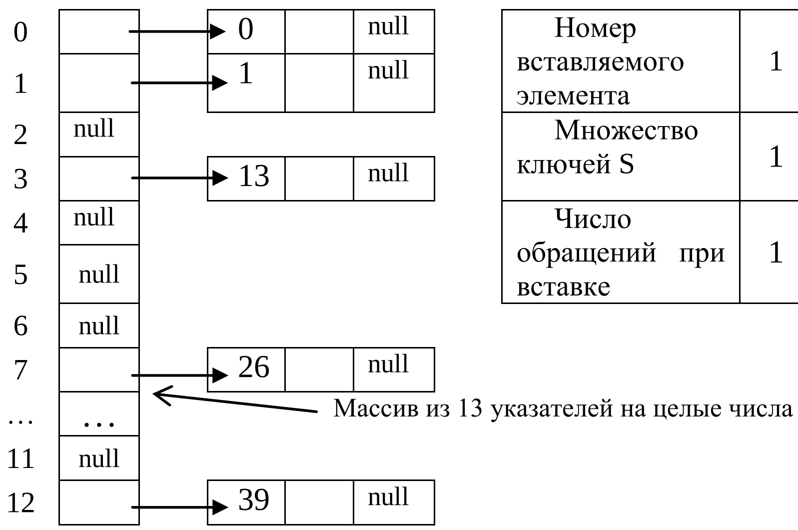
Генерация "хороших случайных" чисел - достаточно сложная задача.

Пример. Случайный метод - $H_i = (H_0(k)+d_i) \bmod N$

$H_0=H(k)=k \bmod N$, $H_i = (H_0(k)+d_i) \bmod N$, N = 13

Сформируем случайную перестановку адресов 0, 1, …, (N - 1), например, с помощью датчика равномерно распределенных чисел (случайный выбор числа d_i из {0, 1, …, (N - 1)} с последующим его удалением). Пусть, например, d = { 0, 3, 7, 12, 1, 10, 2, 4, 11, 5, 8, 6 ,9}.

Вставка n=5 чисел {1 0 13 26 39}.



Номер вставляемого элемента	1	2	3	4	5
Множество ключей S	1	0	13	26	39
Число обращений при вставке	1	1	3	4	5

Вставка ключа k=1.

H(1) =$k \bmod N$=1 $\bmod$ 13= 1 Число обращений при вставке 1 = 1.

Вставка ключа k=0.

H(0) =0 $\bmod$ 13= 0 Число обращений при вставке 0 = 1.

Вставка ключа k=13.

H(13) = 13 $\bmod$ 13=0, H_0=H(13)=0

$H_1=(H_0+d_1) \bmod 13= (0+0) \bmod 13=0$

$H_2=(H_0+d_2\cdot) \bmod 13= (0+ 3) \bmod 13 = 3$ Число обращений при вставке 13 = 3.

Вставка ключа k=26.

H(26) = 26 $\bmod$ 13=0, H_0=H(26)=0

$H_1=(H_0+d_1) \bmod 13= (0+0) \bmod 13=0$

$H_2=(H_0+d_2\cdot) \bmod 13= (0+ 3) \bmod 13 = 3$

$H_3=(H_0+d_3\cdot) \bmod 13= (0+ 7) \bmod 13 = 7$ Число обращений при вставке 26 = 4.

Вставка ключа k=39.

H(39) = 39 $\bmod$ 13=0, H_0=H(39)=0

$H_1=(H_0+d_1) \bmod 13= (0+0) \bmod 13=0$

$H_2=(H_0+d_2\cdot) \bmod 13= (0+ 3) \bmod 13 = 3$

$H_3=(H_0+d_3\cdot)$ *mod* 13= (0+ 7) *mod* 13 = 7
$H_4=(H_0+d_4\cdot)$ *mod* 13= (0+ 12) *mod* 13 = 12 Число обращений при вставке 26 = 5.

Среднее число обращений при вставке = (1+1+3+4+5)/5=14/5=2,8.
Коэффициента заполнения таблицы $\alpha=n/N$=5/13.

<u>Эффективность методов открытой адресации</u> оценивается с помощью показателей:

1. $S_i=S_i(\alpha)$ – среднее число обращений к таблице при вставке как функция от коэффициента заполнения таблицы α=n/N, где n − количество имеющихся в таблице элементов,

2. $S_f=S_f(\alpha)$ – среднее число обращений к таблице при результативном поиске как функция от коэффициента заполнения таблицы α.

Коэффициент заполнения таблицы α равен 0 в случае пустой таблицы (n = 0) и равен 1 (n = N) в случае полностью заполненной таблицы, т.е. $0 \le \alpha \le 1$.

Очевидно, что

– среднее число обращений к таблице при результативном поиске равно среднему числу обращений к таблице при удалении,

– среднее число обращений к таблице при вставке равно среднему числу обращений к таблице при безрезультативном поиске.

<u>Недостатки повторного хеширования</u> для обработки коллизий:

1. повторное хеширование предполагает фиксированный размер таблицы, то есть необходимо чтобы число записей $n \le N$. Если число записей n превысит размер таблицы *N*, то новые записи невозможно вставлять без выделения таблицы большего размера и повторного вычисления значений хеширования для ключей всех записей, находящихся уже в таблице, используя новую хеш-функцию.

2. трудность удаления записей из такой таблицы. Пусть в позиции p находится запись 21. При добавлении записи 22, чей ключ k_2 хешируется в *p*, эта запись должна быть вставлена в первую свободную позицию *RH(p), RH(RH(p)),…*. Предположим, что 21 затем удаляется, так что позиция *p* становится свободной. Поиск записи 22 начинается с позиции $H(k_2)=p$. Но так как эта позиция уже свободна, процесс поиска может ошибочно сделать вывод, что записи 22 в таблице нет.

2.2 <u>Метод цепочек</u> − внешнее разрешение коллизий, метод прямого связывания [*direct chaning*].

<u>Метод цепочек</u> представляет собой организациию связанного списка из всех записей, чьи ключи хешируются в одно и тоже значение.

<u>Пример</u>: Множество ключей - {*k*}={75, 66, 42, 192, 91, 40, 49, 87, 67, 16, 417, 130, 372, 227}, $H(k) = k \bmod N$, *N = 10*,

A - массив указателей на списки, содержащие записи {*a*} с ключами {*k*}.

Структура узла

k - поле для ключа	*a* - поле для записи с ключом *k*	*next* - указатель на следующий узел в списке

0		40			130		null						
1		91		null									
2		42			192			372		null			
3	null												
4	null												
5		75		null									
6		66			16		null						
7		87			67			417			227		null
8	null												
9		49		null									

Список (цепочка) для $H(k_i)=7$

Рисунок 1 – Разрешение коллизий методом цепочек

Основным недостатком метода цепочек является то, что для узлов указателей требуется дополнительное пространство. Однако в алгоритмах, использующих метод цепочек, первоначальный массив меньше, чем при повторном хешировании. Так как при методе цепочек не так катастрофично, если весь массив становится заполненным. Всегда есть возможность выделить дополнительные узлы и добавить их к различным спискам. Конечно, если эти списки станут очень длинными, то теряет смысл вся идея хеширования - прямая адресация и в результате − эффективный поиск.

Если каждый сегмент в среднем будет иметь $||S|| / N$ ключей, то операторы В (*Insert*), У (*Delete*), П (*Member*) будут выполняться в среднем за время $O(1+||S|| / N)$, здесь константа 1 соответствует поиску сегмента, а $||S|| / N$ − поиску ключа в сегменте. Если $||S||\approx N$, то время выполнения операторов не зависит от N − $O(1)$.

Таким образом, процесс заполнения хеш-таблицы (вставки N ключей в таблицу) имеет сложность $O(N(1+||S|| / N))$, если $||S|| = N$, то получим сложность $O(N)$.