КАСКАДНОЕ ХЕШИРОВАНИЕ

Peter Kjellberg и Torben U. Zahle

Факультет компьютерных наук Копенгагенского университета
Sigurdsgade 41, DK-220 Копенгаген N, Дания

Аннотация

Каскадное Хеширование (ориг. «*Cascade Hashing*») -- это новая динамическая схема хэширования, основанная на Спиральном Хранении (ориг. «*Spiral Storage*»).

Цель этой статьи -- во-первых, дать унифицированное изложение Линейного Хеширования (ориг. «*Linear Hashing*»), Спирального Хранения и других динамических схем хэширования, а во-вторых, описать новый метод хранения записей переполнения. Метод хранит записи переполнения в основном файле и кластеризует записи переполнения из каждой основной корзины в одну или несколько корзин переполнения.

Вычисления загрузки файла обещают длительность поиска, очень близкую к единице даже при заполнении хранилища выше 90%, что делает метод более приемлемым, чем любая существующая динамическая схема хэширования.

1. Введение.

С начала семидесятых годов были описаны методы, которые позволяют хэш-файлу изменять свой размер динамически вместе с изменением количества хранимых записей.

Первые предложения, названные Расширяемым Хешированием (ориг. «*Expandable Hashing*») [Knot71], Динамическим Хешированием (ориг. «*Dynamic Hashing*») [Lars78] и Растягиваемым Хешированием (ориг. «*Extendible Hashing*») [Fagi79], основаны на одной и той же идее; когда корзины переполнены, то вместо создания записи переполнения -- растягивают файл с помощью новой корзины, в результате чего записи переполненной корзины разделяются между старой и новой корзиной. Для отслеживания разделения, создаётся индекс или каталог, который увеличивается и уменьшается вместе с размером файла.

Для индекса используются разные типы древовидных структур, но Виртуальное Хэширование (ориг. «*Virtual Hashing*») [Litw78] уменьшает индекс до битовой таблицы с одним битом для каждой корзины, вводя ряд хеш-функций, по одной для каждого «уровня» разделения. На любом этапе запись основной корзины может быть достигнута без доступа к другим корзинам.

Виртуальное Хеширование также позволяет отложить разделение на определённое количество переполнений. Версии Растягиваемого Хеширования с переполнением описаны в [Scho81] и [Tamm82]. Довольно сложная версия с индексом ограниченного размера приведена в [Lome83].

В конце семидесятых были изобретены методы, которые полностью избегали индексов: Спиральное Хранение [Mart79] и Линейное Хеширование [Litw80]. В то время как Спиральное Хранение осталось незамеченным, Линейное Хэширование привлекло большое внимание. Основой этих методов по-прежнему является разделение корзин, но вместо разделения переполненных корзин, все корзины разделяются в линейном порядке 0,1,2,..., тем самым уменьшая индекс до одного указателя, показывающего, какая корзина должна быть разделена следующей.

Линейное Хеширование с Частичным Расширением [Lars80] -- это обобщение Линейного Хеширования, которое даёт лучшую производительность. Дальнейшее обобщение Линейного Хеширования дано Ramamohanarao и Lioyd [Рама82].

[Mul181] и [Lars82] описывают версии Линейного Хеширования и Линейного Хеширования с Частичными Расширениями соответственно, где переполнение хранится в основном файле, в отличие от более ранних версий, где переполнение хранилось в отдельной области переполнения.

Схемы без индексов будут рассмотрены более подробно далее в статье.

Количество записей, которое может храниться в корзине, называется блок-коэффициентом и обозначается буквой b; b' -- это блок-коэффициент возможного отдельного хранилища переполнения. Если основной файл и отдельное хранилище переполнения содержит M и M' корзин соответственно, файл может содержать не более $bM+b'M'$ записей. Когда текущее количество записей соответствует x, тогда $x/(bM+b'M')$ называется использованием хранилища, а x/bM называется коэффициентом загрузки, обозначаемым f. Следует отметить, что когда используется отдельное хранилище переполнения, коэффициент загрузки может быть больше единицы. Когда переполнение хранится в основном файле, коэффициент загрузки равен использованию хранилища.

В настоящей статье мы рассмотрим только так называемое «контролируемое разделение», где разделение выполняется только тогда, когда файл заполнен до определенного предела что, таким образом, позволяет размеру файла расти линейно с количеством сохранённых записей.

2. Концептуальное Линейное Хэширование.

Сначала мы кратко опишем Линейное Хеширование, введя термины, которые будут использоваться в следующих описаниях.

Линейное Хэширование расширяет файл, разделяя корзины одну за другой в линейном порядке 0,1,2,…

Когда корзина разделяется примерно половина её записей удаляется и восстанавливается в новой корзине в конце файла. Никакой индекс не нужен, только единственный указатель p показывающий, какая корзина разделяется следующей. Когда все корзины в файле разделены, положение указателя переводится на первую корзину (0), и процесс начинается снова, но теперь с файлом вдвое большего размера. Записи переполнения обрабатываются явными указателями на отдельную область переполнения.

Мы рассмотрим только случай, когда начальный размер файла равен 1.

После L, завершённых удвоений или расширений файла, его размер составляет 2^L. L называется файловым уровнем.

Используя ряд функций разделения $h_L(k)$, по одной для каждого файлового уровня, где каждый $h_L(k)$ хэш имеет случайное значение на интервале $[0, 2^L[$, мы получаем хеш-функцию

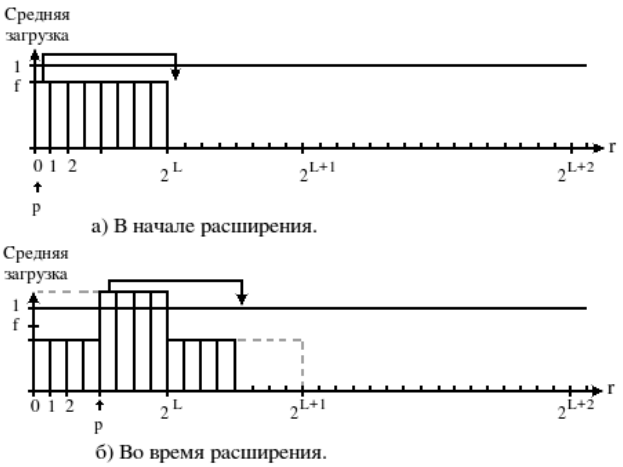
$$h(k)=\begin{cases} h_L(k) & \text{если } h_L(k)\geqslant p \\ h_{L+1}(k) & \text{иначе.} \end{cases} \qquad (2)$$

$h(k)$ является действительным числом; для получения целого числа корзин мы используем

$$H(k)=\lfloor h(k)\rfloor .$$

В следующем обсуждении мы используем вещественный номер корзины и принимаем усечение, чтобы получить целочисленный номер корзины.

Пусть $D(k)$ функция равномерно распределяющая записи по большому числу корзин по мере необходимости. Тогда пример функции Линейного Хеширования показан на рис. 1.



$$r=h^M(k)=\begin{cases} D(k) \bmod 2^L & \text{если } D(k)\bmod 2^L\geqslant p \\ D(k)\bmod 2^{L+1} & \text{иначе} \end{cases}$$

где $L=\lfloor \log_2 M\rfloor$ и $p=M-2^L$

Рис.1. Линейное Хеширование.

Высота корзин на рисунке иллюстрирует загрузку, где f -- коэффициент загрузки файла. Корзины, превышающие 1, переполнены; всё что выше 1 фактически хранится в другом месте.

Обратите внимание, что любая функция разделения $h_{L+1}(k)$ хранит k либо в той же корзине, что и предыдущая функция $h_L(k)$, либо в корзине, в которую $h_L(k)$ была перенесена после разделения.

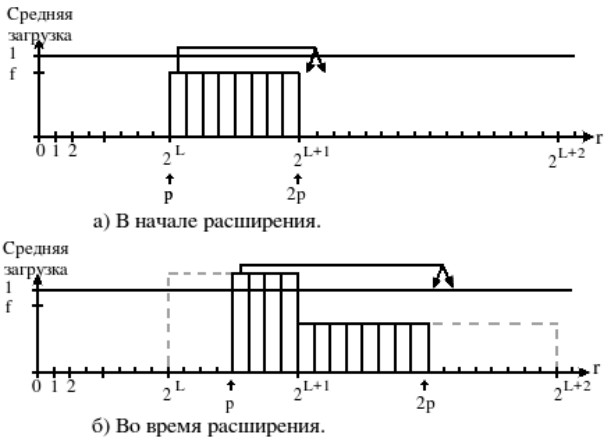
По мере роста файла до определенного размера (M) можно определить уровень L и позицию разделения p. В случае Линейного Хеширования имеем $L=\lfloor \log_2 M\rfloor$ и $p=M-2^L$.

В следующем примере мы пометим хэш-функцию $h(k)$ надстрочным индексом M, чтобы указать, сколько корзин содержит файл: $h^M(k)$.

Разделение в Линейном Хэшировании можно рассматривать как процесс, в котором первая корзина в файле удаляется, а ее записи восстанавливаются в двух новых корзинах, добавленных в конец файла. На рис. 2 показан этот процесс, называемый

Концептуальным Линейным Хешированием (ориг. «*Conceptual Linear Hashing*»), который мы будем рассматривать как Линейное Хеширование в остальной части статьи.

$d(k)$ является функцией, равномерно распределяющей записи на отрезке [0,1[,



а) В начале расширения.

б) Во время расширения.

$$r = h^M(k) = \begin{cases} 2^L(1+d(k)) & \textit{если } 2^L(1+d(k)) \geqslant p \\ 2^{L+1}(1+d(k)) & \textit{иначе} \end{cases}$$

где $p = M$ и $L = \lfloor \log_2 p \rfloor$

Рис.2. Концептуальное Линейное Хеширование

При разделении первой корзины (корзина 1), файл удваивается и уже состоит из корзин {2,3}. Когда обе эти корзины разделяются, файл снова удваивается и теперь состоит из корзин {4,5,6,7}. Этот процесс может быть продолжен, и после завершения L расширенияй файла состоящего из корзин $\{ 2^L , \ldots, 2^{L+1}-1 \}$.

В любое время, корзина p для следующего разделения является первой корзиной в файле. M количество корзин в файле равно p, так что обычно файл состоит из корзин $\{ p, \ldots, 2p-1 \}$. Когда корзина p разделяется, её записи восстанавливаются в корзинах $2p$ и $2p+1$. Файловый уровень L является функцией p: $L = \lfloor \log_2 p \rfloor$.

Компьютерная система обычно позволяет областям данных расти и сокращаться только с одного конца. Во всяком случае, эту проблему легко преодолеть, поскольку существует простой алгоритм, который преобразует (концептуальные) номера корзин выше в физические номера корзин или номера страниц 0,1,2,...

На рис. 3a номера существующих концептуальных корзин перечислены для первых нескольких M-ов.

Алгоритм, преобразующий их в номера страниц, должен удовлетворять двум критериям:

		Концептуальный номер корзины: n												
		1	2	3	4	5	6	7	8	9	10	11	12	13
M	1	1												
	2		2	3										
	3			3	4	5								
	4				4	5	6	7						
	5					5	6	7	8	9				
	6						6	7	8	9	10	11		
	7							7	8	9	10	11	12	13

(а) Существующие концептуальные номера корзин

		Фактический номер корзины: $P(n)$						
		0	1	2	3	4	5	6
M	1	1						
	2	2	3					
	3	4	3	5				
	4	4	6	5	7			
	5	8	6	5	7	9		
	6	8	6	10	7	9	11	
	7	8	12	10	7	9	11	13

(б) Размещение концептуальных корзин

Рис.3. Концептуальное преобразование числа фактической корзины

1) Должны использоваться все номера страниц от 0 до $M-1$.
2) Когда корзина хранится на странице, она должна оставаться на этой же странице на протяжении всего срока ее службы.

Эти требования могут быть легко выполнены: при расширении файла первая из новых корзин размещается на той же странице, что и удаляемая корзина, а вторая новая корзина может размещаться на первой свободной странице (рис. 3b).

Один взгляд на столбцы рис. 3b сразу открывает шаблон, который используется в следующем алгоритме, преобразуя n -- концептуальный номер корзины в $P(n)$ -- это номер страницы.

```
function P(n);          (Алг. 1a)
integer n;
begin
    if n mod 2 = then P(n) := P(n/2)
    else P(n) := (n-1)/2;
end;
```

Алгоритм, написанный без рекурсии:

```
function P(n);          (Alg. 1b)
integer n;
begin
    integer m;
    m := n;
    while m mod 2 = 0 do m := m/2;
    P(n) := (m-1)/2;
end;
```

3. Метод экспоненциального распределения.

На рис. 2 также показана основная проблема Линейного Хеширования: поскольку каждая функция разделения $h_L(k)$ распределяет записи равномерно по своему диапазону корзин, то при расширении файла всё ещё неразделённые корзины будут содержать в два раза больше записей, чем те, которые уже разделены. Сохраняя коэффициент загрузки постоянным (и предпочтительно высоким), изменение количества записей переполнения происходит из неразделённых корзин и приводит к изменению производительности.

Следующее основано на двух основополагающих идеях. Во-первых, это может привести к меньшему переполнению, чтобы было асимметричное распределение записей, для того чтобы корзины, разделяемые позднее, были не так полны. Во-вторых, было бы неплохо выбрать распределение, которое позволило бы сохранить стабильную производительность.

Несмотря на то, что Линейное Хэширование имеет хорошее значение средней производительности, во многих приложениях количество записей в файле стабилизируется на более короткие или более длительные периоды, и точка стабилизации может быть там, где производительность хуже всего.

Чтобы удалить колебания производительности, мы должны построить хеш-функцию, которая даёт файлу одинаковый шаблон загрузки для любого количества корзин. Функция должна смещать распределение записи таким образом, чтобы в последней корзине файла содержалось приблизительно половина записей первой корзины. Затем две новые корзины, возникающие в результате разделения, будут хорошо вписываться в смещенную загрузку.

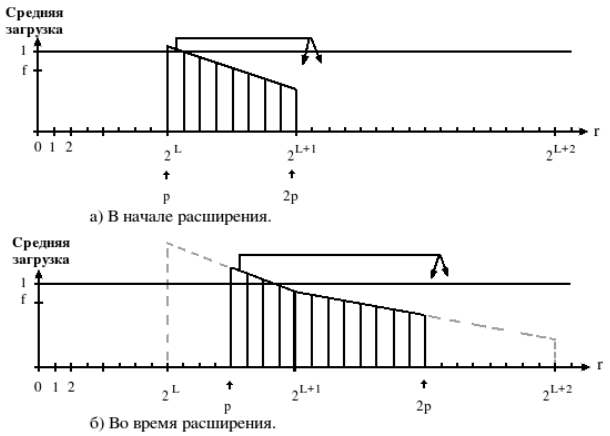


Рис.4. Смещённое Линейное Хэширование.

При такой смещённой загрузке постоянный коэффициент загрузки равный 75% может поддерживаться практически без записей переполнения, что приводит к уменьшению длины поиска практически до единицы.

Конструкция хеш-функции, которая создаёт одинаковое распределение записей для любого количества корзин, не является тривиальной. Если используется наиболее прямолинейное смещение, постепенно уменьшающее распределение, производительность всё равно будет колебаться.

Однако метод, названный Спиральным Хранением, описанный в университетском докладе G.N.N. Martin [Mart79], на самом деле является схемой подобной смещённому линейному хэшированию. И к тому же метод оказывается довольно простым!

Поскольку отчёт Martin, на наш взгляд, трудно понять, и поскольку метод не объясняется в терминах, аналогичных другой литературе по хэшированию, мы будем в следующем обобщать результаты Martin, используя терминологию, введённую в разделе 2.

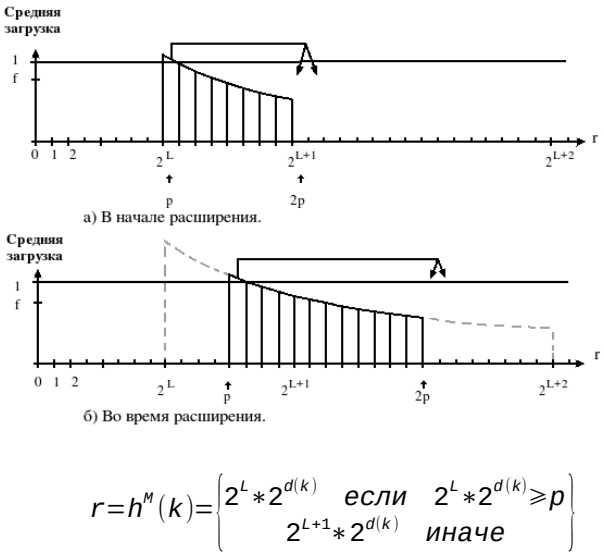
Суть Спирального Хранения заключается в том, что вместо равномерного распределения, получаемого из функций разделения $2^L(1+d(k))$, получается смещённое распределение с использованием функций разделения

$$h_L(k)=2^L*2^{d(k)}, \qquad (3)$$

т.е.

$$h^M(k)=\begin{cases} 2^L*2^{d(k)} & \text{если } 2^L*2^{d(k)}\geqslant p \\ 2^{L+1}*2^{d(k)} & \text{иначе} \end{cases}, \qquad (4)$$

где $p=M$ и $L=\lfloor \log_2 p \rfloor$.



$$r=h^M(k)=\begin{cases} 2^L*2^{d(k)} & \text{если } 2^L*2^{d(k)}\geqslant p \\ 2^{L+1}*2^{d(k)} & \text{иначе} \end{cases}$$

где $p=M$ и $L=\lfloor \log_2 p \rfloor$.

Рис.5. Спиральное Хранение.

В предыдущем отчёте [Kjel84] мы показали, что средняя загрузка для данного хэш-значения r равна:

$f/\ln 2 * p/r$

так что при заданной M и, значит p, средней загрузке следует обратная функция $1/r$. Это относится к обеим функциям разделения, поэтому нам удалось получить плавное распределение загрузки без разрыва на пересечении двух функций разделения.

Далее видно, что загрузка в начале файла ($r=p$) есть $f/\ln 2$, а загрузка в файле ($r=2p$) есть $1/2*f/\ln 2$.

Интересно отметить, что (4) фактически создаёт файл с искомыми свойствами. Загрузка последней корзины (немного больше) составляет половину загрузки первой корзины, а относительное распределение записей в файле не изменяется.

Если загрузка в начале файла установлена в 1, и уравнение затем решено для f, то максимальный коэффициент загрузки файла, который идеально не имеет переполнения, может быть найден:

$f/\ln 2=1 <=> f=\ln 2=0.6931$.

Спиральное Хранение производит постоянное количество переполнения, относительный размер которого получен в [Kjel84]. Кроме того, получены средние и максимальные относительные величины переполнения с использованием Линейного Хэширования с использованием тех же допущений, что и для Спирального Хранения. Результаты показаны в таблице 1. Видно, что Спиральное Хранение не только устраняет колебания в размере переполнения, но также производит меньше переполнения, чем среднее значение для Линейного Хеширования!

f	С.Х.	ЛХ ср.	ЛХ max
0.5000	0.0000	0.0000	0.0000
0.6931	0.0000	1.6848	4.5408
0.7500	0.4367	2.9871	6.7347
0.8000	1.4143	4.3741	8.7722
0.8500	2.8077	5.9421	10.8611
0.9000	4.5179	7.6523	12.9687
0.9500	6.4702	9.4712	15.0727
1.0000	8.6071	11.3706	17.1573

Таблица 1. Относительное количество переполнения с использованием Спирального Хранения и Линейного Хеширования (среднее и максимальное) для разных коэффициентов загрузки (f). В pct.

В заключение этого раздела отметим, что хеш-функция для Спирального Хранения фактически может быть записана как одно выражение:

$$H(k)=\lfloor 2^{\lceil \log_2 p-d(k)\rceil+d(k)} \rfloor \qquad (5)$$

4. Разделение из нескольких корзин.

P.--A.Larson показал в [Lars80], что по сравнению с Линейным Хэшированием лучшая производительность может быть достигнута путём архивирования с использованием обобщённого Линейного Хэширования, где расширение выполняется в серии частичных расширений (что, по словам Ларсона, во многих случаях является хорошим компромиссом) метод работает следующим образом: в первых группах частичных расширений разбиваются две корзины, перемещая часть (приблизительно, 1/3) своих записей в новую корзину (рис. 6a).

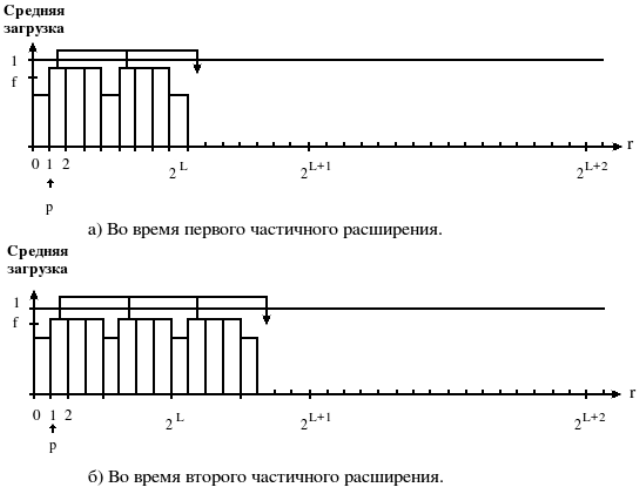


Рис.6. Линейное Хеширование с 2 частичными расширениями.

Когда во время этого первого частичного расширения все исходные корзины были разделены, файл был расширен в 1.5 раза по сравнению с исходным размером. Там - после того, как начинается второе частичное расширение, теперь разбиваются группы по три корзины, перемещая приблизительно 1/4 их записей в новую корзину (рис. 6b). Когда второе частичное расширение заканчивается, файл удваивается, и метод запускается снова с первого частичного расширения

Rammamohanarao & Lloyd [Ramma82] рассмотрели эту идею и дали другой метод, в котором рост состоит из единичных расширений, в которых всегда группируется s разделённых корзин (рис. 7a).

Когда все корзины разделены, начинается новое расширение, снова разделяющее на s групп корзин. Поскольку размер файла не всегда делится на s, в файл добавляется несколько пустых «сборных корзин» (рис. 7б).

Очевидно, что оба метода дают лучшую производительность поиска, чем Линейное Хеширование, поскольку происходит меньшее переполнение. В Линейном Хэшировании всё ещё

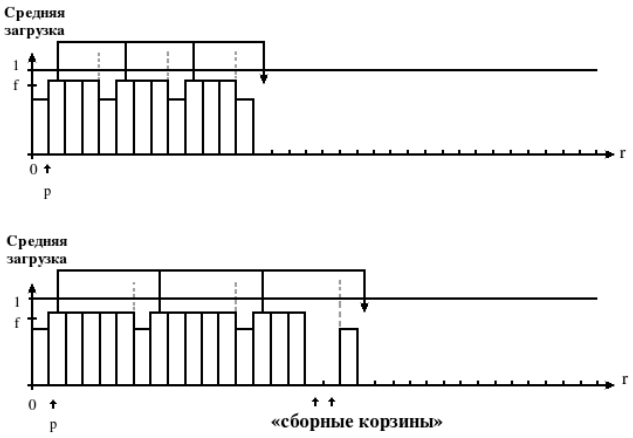


Рис.7. Линейное Хеширование Ramamohanarao и Lloyd ($s=3$) при двух последовательных расширениях.

неразделённые корзины содержат в два раза больше записей, чем разделённые. При 2 частичных расширениях коэффициент составляет 3:2 во время первого частичного расширения и 4:3 во время второго. С методом Ramamohanarao и Lloyd отношение всегда $(s+1):s$.

Более удивительно, что, когда количество частичных расширений (соответственно значение s) является разумным, также вставки быстрее, чем с Линейным Хешированием, несмотря на то, что разделение дороже! Причина в том, что увеличение стоимости разделения более чем перевешивается уменьшением длин переполнения цепочек.

Но всё же производительность колеблется, и в обоих случаях хеш-функция сложнее, чем с Линейным Хешированием.

Можно ожидать, что разделение из нескольких корзин в Спиральном Хранилище (далее называемое Обобщенным Спиральным Хранилищем (ориг. «*Generalized Spiral Storage*»)) также покажет улучшенную производительность как для времени поиска, так и для вставки. Кроме того, он не показывает никаких неудобств схем обобщенного Линейного Хеширования указанных выше.

Обобщенное спиральное хранилище напоминает метод Ramamohanarao & Lloyd (но на самом деле ещё более общий). Производительность одинакова для любого количества корзин, хеш-функция не сложнее, чем простое Спиральное Хранение, и для неё не нужны никакие «сборные корзины».

Идея обобщённого Спирального Хранилища очень проста: каждое расширение выполняется путём: 1) добавления нескольких корзин, скажем t , к верхнему концу файла и 2) удаления некоторого числа, скажем s , из нижнего конца файла, и распределением записей s удалённых корзин на t новых корзин! Мы говорим, что s корзины разделены на t или, что, то же самое, что 1 корзина разделяется на t/s . Когда $t=2$ и $s=1$ это простое Спиральное Хранение.

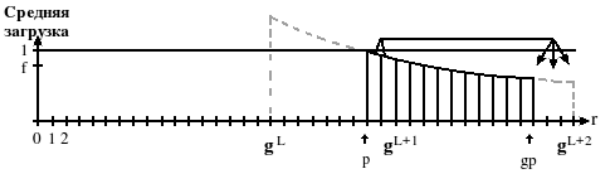
Допустим $g=t/s$. В общем случае «уровень файла» определяется как $L=\lfloor \log_g p \rfloor$. Обратите внимание, что теперь L это не количество удвоений файла. (Фактически, увеличение L на единицу эквивалентно расширению файла на коэффициент g , который обычно не является целым числом корзин!)

Если, опять же, $d(k)$ – функция, равномерно распределяющая все записи на отрезке [0,1[, то

$$h^M(k)=\begin{cases} g^L * g^{d(k)} & \text{если } g^L * g^{d(k)} \geqslant p \\ g^{L+1} * g^{d(k)} & \text{иначе} \end{cases} \qquad (6)$$

где $p=M/(g-1)$ и $L=\lfloor \log_g p \rfloor$,

является хэш-функцией для обобщённого Спирального Хранения.



а) В начале расширения.

$$r=h^M(k)=\begin{cases} g^L * g^{d(k)} & \text{если } g^L * g^{d(k)} \geqslant p \\ g^{L+1} * g^{d(k)} & \text{иначе} \end{cases}$$

где $p=\dfrac{M}{g-1}$ и $L=\lfloor \log_g p \rfloor$,

Рис.8. Обобщённое Спиральное Хранение с $g=\dfrac{3}{2}$.

Средняя загруженность для данного хеш-значения r равна ([Kjel84]):

$$f(g-1)/\ln(g)*p/r$$

что означает, что мы получаем равномерное распределение загруженности в общем случае.

Загруженность в начале и в конце файла равна $f(g-1)/\ln(g)$ и $1/g*f(g-1)/\ln(g)$, соответственно. Это означает, что когда, например, при $g=3/2$ последние корзины файла загружены (чуть больше) на две трети загрузки первой корзины.

Мы достигли более равномерного распределения записи по файлу, уменьшив количество записей переполнения; цена -- это большее количество обращений для выполнения разделения.

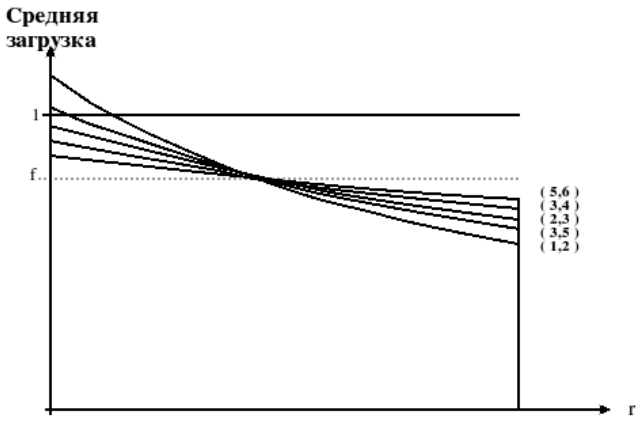


Рис.9. Средняя загруженность файла обобщённого Спирального Хранения для выбора (s,t)-пар.

В случае, когда $t=s+1$ эквивалентно случаям Ramamohanarao и Lloyd: каждое разделение производит одну дополнительную корзину. В других случаях из каждого разделения получается более одной корзины (тем самым провоцируя разделение более редко), но во всех остальных отношениях не сложнее. Например, случай $(s,t)=(3,5)$ генерирует две дополнительные корзины, разделяя три (которые можно рассматривать как разделение 1 корзины на 5/3 корзин три раза) и фактически во всех отношениях он действует как компромисс между (1,2) и (2,3) (два раза по 1 корзине на 3/2 корзин).

Опять же, мы можем найти максимальный коэффициент загруженности файла, который в идеале не имеет переполнения, равный загруженности в начале файла (см. выше) с 1 и решением для f :

$$f=\ln(g)/(g-1) .$$

s,t	f_{max}
1,2	0.6931
3,5	0.7662
2,3	0.8109
5,7	0.8412
3,4	0.8630
4,5	0.8926
5,6	0.9116

Таблица 2. Максимальный коэффициент загруженности для обобщённого Спирального Хранения без переполнения для выбора s,t-пар.

Как и в случае с простым Спиральным Хранением, относительная величина переполнения выводится в [Kjel84]:

f	1,2	3,5	2,3	5,7	3,4	4,5	5,6
0.70	0.01						
0.75	0.44						
0.80	1.41	0.18					
0.85	2.81	1.02	0.27	0.02			
0.90	4.52	2.40	1.29	0.66	0.30	0.02	
0.95	6.47	4.22	2.93	2.11	1.55	0.85	0.46
1.00	8.61	6.36	5.06	4.20	3.59	2.79	2.28

Таблица 3. Относительная величина переполнения с использованием обобщённого Спирального Хранения для выбора s,t-пар и разных коэффициентов загруженности (f). В pct.

Единственной проблемой, оставшейся при работе с обобщённым Спиральным Хранением является обобщение алгоритма 1, преобразующего номера концептуальных корзин в (физические) номера страниц. Алгоритм работает точно так же, как алг. 1 и приводится здесь без дальнейших комментариев.

Алгоритм преобразует номер n концептуальной корзины в номер страницы $P(n)$ в случае, когда s корзин разбиваются на t :

```
function P(n);                    (Алг . 2a)
integer n;
begin
    if n mod t < s
    then p(n) :=
        P( ⌊n/t⌋ *s + n mod t)
    else P(n) :=
         ⌊n/t⌋ *(t-s)+(n-s) mod t;
end;
```

Алгоритм, написанный без рекурсии:

```
function P(n);              (Алг. 2b)
integer n;
begin
    integer m;
    m := n;
    while m mod t < s do
        m := ⌊m/t⌋ *s + m mod t;
    P(n)  := ⌊m/t⌋ *(t-s) + (m-s) mod t;
end;
```

Можно заметить, что в случаях, когда $t=s+1$ алг. 2 сводится к:

```
function P(n);                    (Алг. 3)
integer n;
begin
    if n mod t != s
    then P(n) := P( ⌊n/t⌋ *s + n mod t)
    else P(n) := (n-s)/t;
end;
```

Наконец, отметим, что хеш-функция для обобщённого Спирального Хранения также может быть записана как одно выражение:

$$H^M(k)=\lfloor g^{[\log_g p-d(k)]+d(k)} \rfloor \qquad (7)$$

где $p=M/(g-1)$.

5. Обработка переполнения.

Самый простой способ решить проблему переполнения -- это, конечно, просто избежать переполнения вообще. Из-за предвзятого распределения записей в файле, мы должны ожидать, что переполняющаяся корзина будет расположена в начале файла, и мы можем просто разделить корзины до той и включая ту, которая переполняется. Однако эта схема имеет два основных недостатка, которые обусловлены тем фактом, что довольно длинная последовательность разделений может быть результатом одиночных вставок:

-- Коэффициент загруженности файла может сильно уменьшиться.

-- Время от времени может происходить очень длительное время вставки.

По этой причине и для увеличения коэффициента загруженности за пределами значений, перечисленных в таблице 2, ниже описана новая схема обработки переполненных записей.

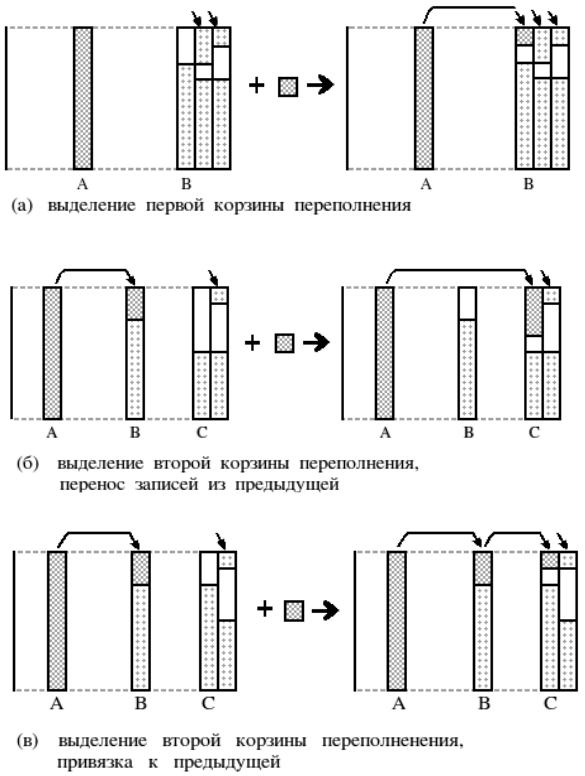
Основной целью метода является включение переполнения в основной файл, тем самым уменьшая область данных до одного файла, линейно увеличивающегося по размеру с количеством хранимых записей.

Как уже упоминалось во введении, две схемы Линейного Хеширования с переполнением в основном файле были описаны Mullin [Mull81] и Larson [Lars82]. Оба метода связывают записи переполнения с основной корзиной. Разница между ними заключается в том, что, хотя Mullin использует пустое пространство в других корзинах для переполнения, Larson полностью откладывает каждую k -ую корзину для переполнения.

Larson завершает свою статью, сообщая, что переполненные записи были кластеризованы в одну или несколько корзин переполнения. Это вторая цель настоящего метода. Для достижения этой цели метод использует, например, Mullin, пустое пространство в других корзинах для переполнения, но вместо сцепления одиночных записей переполнения он связывает корзину переполнения, т.е. пустое пространство в корзине используется полностью для переполнения из другой корзины.

Успех этой идеи связан с предвзятым распределением записей. Переполнение происходит «внизу» конца файла, в то время как пустое пространство находится «вверху» конца.

Метод работает следующим образом:



(а) выделение первой корзины переполнения

(б) выделение второй корзины переполнения, перенос записей из предыдущей

(в) выделение второй корзины переполненения, привязка к предыдущей

Рис. 10. Цепь корзин переполнения в Каскадном Хэшировании.

При попытке вставки записи (рис.10а) в полную основную корзину (А) выделяется корзина с наивысшим номером с незанятым пространством переполнения (В), запись вставляется здесь, и эта новая корзина переполнения привязана к основной корзине. Дополнительные записи из основной корзины А вставляются в корзину переполнения Б, по мере наличия свободного пространства.

Когда в В больше нет места, таким же образом выбирается новая корзина переполнения (С): корзина с наибольшим числом незаполненного пространства переполнения.

Так как С обладают обычно более высоким номером, чем В (файл предположительно был разбит несколько раз из-за вставок), то можно ожидать, что пространство переполнения в С больше, чем в В, а затем, как правило, все записи переполнения А могут быть сохранены в С, таким образом сохраняя длину цепочки переполнения А равной 1. Поэтому, если это возможно, записи переполнения в В переносятся на С и цепочка перекомпонуется (рис.10б), мы говорим, что переполнение записей происходит каскадом от В до С. Дальнейшие записи переполнения вставляются с использованием того же механизма.

Если С не может содержать все записи переполнения, то в С вставляется только новая запись, а С привязывается к В (рис.

10в).

Цель состоит в том, чтобы сохранить цепочки переполнения короткими. Поскольку основной операцией в хэш-файле является поиск, во время вставки выгодно реорганизовать цепочки переполнения, чтобы сократить время поиска - пока реорганизация не оказывает серьёзного влияния на время вставки и удаления.

Есть ещё два важных способа сохранить цепочки короткими. Во-первых, во время вставок ни одна запись не должна быть записью переполнения, если только её основная корзина не заполнена своими собственными записями. Если основная корзина заполнена, но содержит записи переполнения из другой корзины, «незнакомцы» выталкиваются и вставляются в другом месте.

Во-вторых, все корзины в цепочке, кроме последней, должны быть заполнены. Когда запись удаляется, некоторые другие записи с конца цепочки возвращаются, чтобы заполнить пустоту.

Алгоритмы для вставки, удаления, разделения и группировки включены в приложение А.

Нам всё равно нужно убедиться в том, что этот метод работает для всех случаев. В [Kjel84] мы показываем относительное количество переполненных корзин, предполагая идеальное распределение,

$f/\ln(g)-1/(g-1)$.

Это показано в таблице 4. В таблице показано, что для каждой переполненной корзины будет по крайней мере одна свободная корзина но, как правило, намного больше.

f	1,2	3,5	2,3	5,7	3,4	4,5
0.70	0.01					
0.75	8.2					
0.80	15.4	6.6				
0.85	22.6	16.4	9.6	2.6		
0.90	29.8	26.2	22.0	17.5	12.8	3.3
0.95	37.1	36.0	34.3	32.3	30.2	25.7
1.00	44.3	45.8	46.6	47.2	47.6	48.1

Таблица 4. Относительное число переполненных корзин для выбора (s,t)-пар и различных коэффициентов загруженности (f). В pct.

Конечно, может случиться так, что фактические записи не распределяются равномерно с помощью $d(k)$, поэтому существует риск того, что при необходимости не будет найдено свободной корзины переполнения. Эту проблему можно решить, разрешив объединение цепочек переполнения, т.е., что цепочки переполнения из нескольких основных корзин используют одну и ту же корзину переполнения [Vitt82, Knut73]. Но это, вероятно, не стоит получаемых проблем; гораздо более простым решением является выполнение одного или нескольких разделений. В конечном итоге появится некоторое свободное пространство.

Это решение может привести к снижению коэффициента загруженности ниже требуемого уровня, но этого не должно происходить. Фактически, если это так, это просто предупреждение о том, что, либо коэффициент загрузки установлен слишком высоко, либо используемая функция распределения $d(k)$ недостаточно однородна! (Дополнительное разделение можно рассматривать как «изящное ухудшение»).

Типичная ситуация показана на рис. 11.

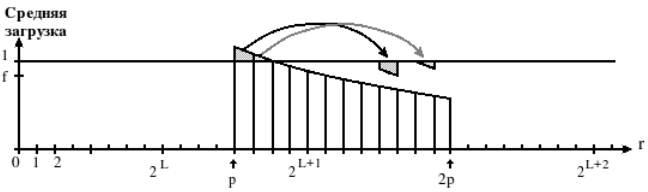


Рис 11. Типичная ситуация с f = 0,8 и g = 2.

Мы завершаем этот раздел ещё несколькими результатами. В [Kjel84] мы выводим номер последней корзины в файле (n_{last}), когда корзина n_{full} переполняется первой. Выбирая n_{last} в качестве корзины переполнения для n_{full} , мы получаем наивысший коэффициент загруженности f_{max} , при котором эта корзина переполнения может удерживать переполнение n_{full} на протяжении всего времени жизни n_{full} . Эти коэффициенты загруженности приведены в первом столбце таблицы 5. Таким образом, с коэффициентом загруженности до f для любой корзины требуется только одна корзина переполнения, и эту корзину переполнения никогда не нужно перемещать!

На практике записи не распределяются идеально, и, следовательно, некоторым корзинам необходимо больше пространства переполнения. Но в файле всё ещё есть место для каскадирования записей переполнения далее в другую корзину.

По этой причине количество записей, хранящихся во второй (или более высокой) корзине переполнения, может считаться незначительным и, таким образом, путём вычисления ожидаемого количества записей переполнения и ожидаемого количества корзин переполнения, мы можем указать ожидаемую длину удачного и неудачного поиска. Это также сделано в таблице 5 -- мы позволим цифрам говорить самим за себя.

В настоящее время ведётся работа по моделированию метода со стохастическим входом.

s,t	f_{max}	0	уп	$0_к$	нп
1,2	0.8568	3.0	1.030	23.6	1.236
3,5	0.8993	2.4	1.024	26.0	1.260
2,3	0.9224	2.0	1.020	27.5	1.275
5,7	0.9369	1.7	1.017	28.5	1.285
3,4	0.9469	1.4	1.014	29.2	1.292
4,5	0.9597	1.1	1.011	30.1	1.301
5,6	0.9675	1.0	1.010	30.7	1.307

Таблица 5. Максимальный коэффициент загруженности для Каскадного Хэширования с одной фиксированной корзиной-переполнения в основную корзину для выбора s,t-пар.

При этих максимальных коэффициентах загруженности таблица также показывает:

0 : Относительное значение переполнения (в pct)

уп : Поиск ожидаемой длины успешного поиска.

$0_к$: Относительное значение переполненных корзин (в pct.)

нп : Поиск ожидаемой длины неудачного поиска.

6. Заключение.

В этой статье описана новая схема линейного хэширования, называемая Каскадным Хешированием. Она основана на двух фундаментальных идеях:

1) Что используется смещённое распределение записей -- как, например, в Спиральном Хранении и
2) То, что записи переполнения кластеризуются из корзины и хранятся в основном файле.

Вычисления, по-видимому, показывают, что производительность в отношении успешного поиска, неудачного поиска, вставки и удаления превосходит другие известные динамические схемы хэширования.

Благодарности

Авторы выражают благодарность Per-Åke Larson, который впервые представил нас в докладе Г.Н.Н.Мартина.

Также благодарим Bent Pedersen из Копенгагенского университета за указание сходства между Линейным Хешированием с Частичными Расширениями и обобщённым Спиральным Хранением.

Ссылки

Fagi79 Fagin, Ronald; Nievergeltm Jurg; Pippenger, Nicholas; Strong, H. Raymond -- Растягиваемое хеширование -- Метод Быстрого Доступа для Динамических Файлов. ACM Trans. по Системам Баз Данных 4, 3, (Сентябрь 1979), 315-344

Kjel84 Kjellberg, Peter; Zahle, Torben U. -- Каскадное Хеширование. DIKU доклад 84/6. Институт Datalogy, Копенгагенский университет, Копенгаген, Дания, 1984

Knot71 Knott, Gary D. -- Расширяемое Хеширование и Извлечение Хэш-Таблиц с Открытой Адресацией. Proc. 1971 ACM--SINGFIDET Семинар по Описанию Данных, Доступу и Контролю, Сан-Диего, Калифорния, 11-12 ноября 1971, стр. 187-206

Knut73 Knuth, Donald E. -- Искусство Компьютерного Программирования, Том 3/ Сортировка и поиск. Addison-Wisley, Рединг, Массачусетс, 1973

Lars78 Larson, Per-Åke -- Динамическое хеширование. BIT 18, 2, (1978), 184-201

Lars80 Larson, Per-Åke -- Линейное хеширование с частичными расширениями. Proc. 6-й Международный. Конф. об Очень Больших Базах Данных, Монреаль, Канада, 1-3 октября 1980, стр. 224--231

Lars82 Larson, Per-Åke -- однофайловая версия линейного хэширования с частичными расширениями. Proc. 8-й Международный. Конф. об Очень Больших Базах Данных, Мехико, сентябрь 1982 г., стр. 300-309

Litw78 Litwin, Witold -- Виртуальное Хеширование: Динамическое Изменяемое Хеширование!. Proc. 4-й Междун. Конф. об Очень Больших Базах Данных, Западный Берлин, Германия, 13-15 сентября 1978 г., стр. 517-523

Litw80 Litwin, Witold -- Линейное хеширование: новый инструмент для адресации файлов и таблиц. Proc. 6-й Междун. Конф. об Очень Больших Базах Данных, Монреаль, Канада, 1--3 октября, 1980, стр. 212-223

Lome83 Lomet, David B. -- Ограниченный Индекс Экспоненциального Хеширования. ACM Trans. по Системе Баз Данных 8, 1, (Март 1983), 136--165

Mart79 Martin, G.N.N. -- Спиральное хранение: постепенно увеличивающееся хэш-адресное хранение. Унив. Уорвика, Доклад по Теории Вычислений ном. 27. Ковентри, Англия, Март 1979

Mull81 Mullin, James K. -- Жёстко Контролируемое Линейное Хеширование Без Отдельного Хранилища Переполнения. BIT 21, 4, (1981), 390--400

Rama82 Ramamohanarao, K.; Lloyd, John W. -- Схемы Динамического Хеширования. Компьютерный Журнал 25, 4, (ноябрь 1982), 478--485

Scho81 Scholl, Michel -- Новые Файловые

Примитивы на Основе Динамического Хеширования. ACM Trans. о Системах Баз Данных 6, 1, (Март 1981), 194--211

Tamm82 Tamminan, Markku -- Расширяемое Хеширование с Переполнением. Inf. Proc. Письма 15, 5, (декабрь 1982), 911--926

Vitt82 Vitter, Jeffrey Scott - Реализации для Объединённого Хеширования. Comm. ACM 25, 12, (Дек. 1982), 911-926

Приложение А

Ниже мы приводим алгоритмы для вставки, удаления, разбиения и группировки. Чтобы упростить алгоритмы, мы предположили, что достаточно большое количество корзин может одновременно находиться в центре, и в ряде ситуаций были опущены более сложные решения для обработки особых случаев, хотя они могли бы сохранить доступ.

Некоторые детали реализации будут кратко рассмотрены в конце приложения.

Алгоритмы предполагают существование параметров s, t и p, поскольку они использовались ранее в этой статье, т.е. файл содержит $(g-1)p$ корзин с числами p ,..., $gp-1$ (где $g=t/s$), а разделение выполняется путём удаления s корзин и добавления t корзин.

Алгоритмы относятся к нескольким «типам корзин»:

Рис 12. Типы корзин

Алгоритм I (Вставка записи k)

1 [Разделение] Если максимальный коэффициент загруженности превышен, выполните разделение.

2 [Поиск] Поиск ключа k. Если ключ найден, вставка завершается неудачей. (Поиск заканчивается последней корзиной в цепочке переполнения для k -- возможно, основной корзиной -- как текущей). Пусть B <-- текущая корзина. (B может быть любого типа, кроме $c1$ и $c2$.)

3 [Корзины типа a или d, вставка] Если B имеет тип a или d (в основной корзине есть место для k), тогда вставьте k в B и завершите.

4 [Корзина типа $e1$ или $e2$] Если B имеет тип $e1$ или $e2$ (основная корзина заполнена, но содержит переполнение из другой корзины), тогда

4.1 [Выбор записи переполнения] Выберите одну из 'внешних' записей переполнения в B, скажем k', и удалите её (временно!).

4.2 [Вставка] Вставьте k в B.

4.3 [Реорганизация и поиск конца цепочки k'].

4.3.1 Если B имеет тип $e1$, а k' была единственной записью переполнения в B, то пусть B <-- предыдущая корзина в цепочке для k'.

4.3.2 Если B имеет тип $e2$, то

4.3.2.1 Если k' была единственной записью переполнения в B, тогда свяжите B с цепочкой k'.

4.3.2.2 Следуйте за цепочкой для k' до конца и пусть B <-- последняя корзина в цепочке.

4.4 [Переключение на вставку k'] Пусть k <-- k'. (Теперь известно, что B имеет тип b, $c3$ или $c4$.)

5 [Корзина типа $c3$, вставка] Если B имеет тип $c3$ (B имеет пространство k), тогда вставьте k и завершите.

6 [Выбор новой корзины переполнения] (теперь B имеет тип b или $c4$.) Выберите новую корзину переполнения B'. Если ничего не найдено, выполните разделение, напишите предупреждение обслуживающему персоналу и вернитесь к I2.

7 [Вставка] Вставьте k в B'.

8 [Реорганизация цепочки] Если B -- корзина переполнения, а B' может содержать все записи переполнения из B, то переместите записи переполнения из B в B' и пусть B <-- предшественник B' в цепочке. Повторяйте I8 как можно большее количество раз.

9 [Связывание] Связать B' с B.

Алгоритм D (Удаление записи k)

1 [Поиск] Поиск ключа k. Если ключ не найден, удаление завершается неудачно. (поиск заканчивается корзиной, содержащей k в качестве текущей записи.) Пусть B <-- текущая корзина.

2 [Удаление] Удаление k из B.

3 [Простое удаление] Если B имеет тип a, b, $c3$, $c4$, d или $e1$ (цепочки не нуждаются в реорганизации после удаления k), тогда перейдите к D7. (В случае $c3$, $c4$, d и $e1$ может быть выполнено действие, аналогичное I8).

4 [Найти конец цепочки] (B теперь имеет тип $c1$, $c2$ или $e2$. Цепочка переполнения нуждается в реорганизации.) Следуйте за цепочкой, исходящей от B до конца; пусть B' <-- последняя корзина в цепочке.

5 [Выбор записи] Выберите запись переполнения в B', скажем k'. Удалите k' из B'. Если k' была единственной записью переполнения в B', тогда удалите связь B' с цепочкой.

6 [Восстановить k'] Вставить k' в B. («Пустота» после заполнения k).

7 [Группа] Если теперь коэффициент загруженности ниже минимального, выполните группировку.

Алгоритм S (Разделения)

1 [Сканирование корзин для разделения]. Пусть B - это корзина с номером p, $p+1$, ..., $p+s-1$ раз. Для каждой из этих корзин сделайте

1.1 [Чтение] Чтение B. (Это может быть любой тип основной корзины, т.е. все, кроме $c2$, $c3$ и $c4$.)

1.2 [Чтение собственного переполнения] Если B имеет тип $c1$, то в его цепочке считываются корзины переполнения, записи переполнения удаляются (но позже сохраняются!), а пространство переполнения в корзинах переписывается помечаясь как пустое.

1.3 [Реорг. 'Внешней' цепочки переполнения] Если B имеет тип d, $e1$ или $e2$ (эта ситуация должна быть очень редкой) – B связан с чужой цепочкой переполнения, то 'внешние' записи переполнения откладываются на некоторое время.

2 [Приращение] Пусть p <-- $p+s$.

3 [Перехеширование] Перехешируйте все записи, включая собственные записи переполнения из S1.2, но исключая 'внешние' записи переполнения из S1.3. (Хеш записи в новых корзинах $gp-t$,..., $gp-1$.) Если некоторые из новых корзин должны переполняться (это должно быть редкой ситуацией), то переполнение откладывается на некоторое время.

4 [Перезапись] Перепишите новые корзины.

5 [Вставка переполнения] Используйте алгоритм I для вставки записей переполнения, отложенных в S1.2 или S3 -- если они есть.

Алгоритм G (Группировка)

1 [Сканировать корзины, которые нужно сгруппировать]. Пусть B - номер корзины $gp-t$,..., $gp-1$. Для каждой корзины сделайте

1.1 [Чтение] Прочтите корзину B. (Она может быть любого типа, кроме $c2$, $c3$ и $c4$.)

1.2 [Чтение собственного переполнения] Если B имеет тип $c1$, то читаются все корзины переполнения в цепочке, записи переполнения из них удаляются (но сохраняются позже!), а освободившееся пространство корзин переписывают помечая его как пустое. (Такая ситуация должна быть редкой.)

1.3 [Перемещение 'внешнего' переполнения] Если B имеет тип d, $e1$ или $e2$, тогда

1.3.1 Если B имеет тип $e2$, тогда воспользуйтесь ссылкой B из цепочки, следуйте цепочке до конца и пусть B' <-- последняя корзина в цепочке. Если B имеет тип d или $e1$, то пусть B' <-- предшественник B в цепочке.

1.3.2 Выберите новую корзину переполнения B_0. Если ничего не найдено, то воссоздайте первоначальную ситуацию, напишите предупреждение обслуживающему персоналу и завершите работу.

1.3.3 Вставьте как можно больше записей переполнения из B в B_0, свяжите B_0 с B' и пусть B' <-- B_0. Повторяйте G1.3.2--G1.3.3 до тех пор, пока все записи переполнения из B не будут восстановлены.

2 [Уменьшение] Пусть p <-- $p-s$.

3 [Перехеширование] Перехешируйте все записи, включая собственные записи переполнения из G1.2. (Хеш записи в новых корзинах p,..., $p+s-1$.)

4 [Перезапись] Для каждой из новых корзин, пусть это будет B', сделайте

4.1 [Перезапись основной корзины] Перезапишите B'.

4.2 [Перезапись переполнения] Сохраните записи переполнения из B', используя ту же процедуру, что и G1.3.2--G1.3.3.

Детали реализации

Цепочка переполнения может быть либо односвязным, либо двусвязным списком. Любая структура имеет как преимущества, так и недостатки. Двусвязному списку требуется больше доступа для обновления указателей, в то время как односвязному списку требуется больше доступа для поиска предшественника в списке (но это легко сделать, просто путём хэширования ключа любой записи и чтения через цепочку из основной корзины).

Алгоритмы используют простые решения для реорганизации цепочек переполнения. Так как все корзины в цепочке, как правило, должны быть прочитаны в любом случае, более совершенное решение всегда может минимизировать количество использованных корзин переполнения!

Чтобы выбрать новую корзину переполнения, все корзины с пустым пространством переполнения могут быть связаны друг с другом. Это означает быстрый выбор, но также доступ в других местах для поддержания цепочки. Если корзины не связаны друг с другом, выбор может быть ускорен путём привязки указателя к самому высокому номеру корзины, который, вероятно, имеет свободное пространство переполнения.