- ☑ При поиске данных в таблицах во многих случаях можно предварительно оценить место расположения искомых записей (например, слово "янтарь" располагается в конце словаря и, следовательно, словарь целесообразно раскрыть ближе к последней странице)
- ⇘ Первоначально забракованный способ аналитического задания отношения "иметь имя" может быть использован для разработки способа представления таблиц (!)

2. Основные понятия и определения…

☑ Пусть для ключей записей имеется правило преобразования к значениям целого типа. Используем получаемые значения для определения места расположения записей

Пример 3.2. Пусть ключ есть строка не более чем из 6 символов. Каждый символ может быть преобразован в целое число (код символа). Тогда и ключ может рассматриваться как значение целого типа

$$A \sim 41_{16},\ B \sim 42_{16},\ \Rightarrow AB \sim 16706_{10}$$

Всего разных имен может быть $N=2^{48}$, тогда положив размер памяти для таблицы $M=2^{48}$, мы сможем организовать ***прямой доступ*** к записям

☑ Организация прямого доступа в рассмотренном виде приводит к неэффективному использованию памяти

– требуется большой памяти (M>>1)

– количество используемых ключей, как правило, существенно меньше теоретически возможного набора (Л.Н. Толстой "Война и мир" содержит всего 500 имен)

↳ Размер памяти для представления таблицы значительно меньше возможного количества имен (M<<N) и, как результат, получаемые по ключам числовые значения необходимо преобразовывать к диапазону номеров (адресов) строк памяти

Определение 3.10. Функция преобразования значения ключа к номеру (адресу) строки памяти для хранения записи

$$H : K \rightarrow L \quad (L=\{0,\ldots,M-1\})$$

называется *функцией* (*хеширования*, *перемешивания*, *рассеивания*) *расстановки* (hash - мешанина, путаница)

Определение 3.11. Таблицы, представление которых организуется при использовании функции расстановки, называются *таблицы с вычислимыми адресами* (*хеш-таблицы*, *перемешиваемые таблицы*)

☑ При $M<N$ функции расстановки является *взаимно-неоднозначной* (*неинъективной*)

$$\exists\, k_1, k_2 \in K : h(k_1) = h(k_2)$$

↳ Тем самым, при использовании функции расстановки могут возникать ситуации, когда получаемый функцией номер строки памяти для расположения записи уже является использованным

Определение 3.12. Ситуация, когда для расположения записи функцией расстановки определяется уже занятая строка памяти называется *относительным переполнением* (*коллизией*)

Требования (редкое возникновение коллизий)

- Быстрое вычисление
- Равномерное распределение имен
- Равномерное распределение часто используемых имен
- Равномерное распределение близких имен

Примеры

- Код левого (правого) символа ключа
- Числовой код ключа
- Представление ключа в виде нескольких полей, используемых для получения целого значения
- $h(k) = \eta(k) \bmod M$ $(M \neq 2^k)$

4. Открытое перемешивание - вставка…

Вставка (начальный вариант)

1. s = h(key) // применение функции расстановки
2. ЕСЛИ строка s свободна, ТО { K[s]=key; Останов }
3. ЕСЛИ K[s] == key, ТО { Дублирование; Останов }
4. (!) Коллизия { Изменение s и переход к п. 2 }

⌛ Как разрешать ситуации коллизии ?

↳ Возможное решение – использование строк s+1, s+2,… (такой способ может провести к появлению *сгущений* в структуре хранения таблицы)

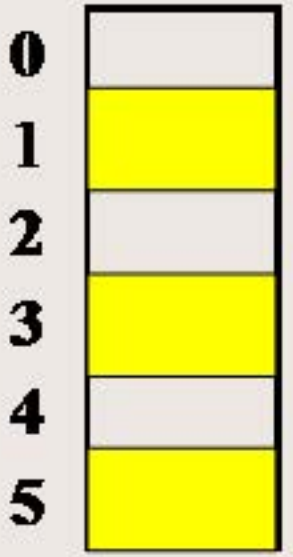
↳ Уменьшение эффекта сгущений может быть достигнуто при применении способа *открытого* или *линейного перемешивания*

$$s' = (s+p) \bmod M \quad (1 \le p < M)$$

4. Открытое перемешивание - вставка…

Рассмотрим выполнение процедуры вставки

Пусть N=6, s=3, p=2



⌛ Как обеспечить нахождение свободных строк ?

↳ Возможное решение состоит в выборе взаимно-простых значений для M и p

☑ В более общем виде правило разрешения коллизии может быть представлено как функция вторичного перемешивания

$$s'=h'(s)$$

4. Открытое перемешивание - вставка…

Теорема 3.2. Алгоритм открытого перемешивания при взаимно-простых M и p гарантирует нахождение свободных строк структуры хранения таблицы

Доказательство. Рассмотри множество $G=\{0,1,\ldots,M-1\}$ с операцией $a \oplus b=(a+b) \bmod M$.

Свойства операции:

- G замкнута относительно $\oplus$
- операция ассоциативна и коммутативна
- $\exists$ нулевой и обратные элементы

$\Rightarrow$ Множество G с операцией $\oplus$ является группой

Выделим подмножество $G'=\{0, a, a \oplus a,\ldots\}$. Такое множество G' с операцией $\oplus$ тоже является группой (группы такого вида называются *циклическими*)

4. Открытое перемешивание – вставка…

Обозначим $a \oplus a \oplus \ldots \oplus a$ через na (a - число повторений)

Если $n>0$, то минимальное значение n, при котором $na=0$, называется *порядком* элемента a и обозначается $|a|$ (т.е. порядок определяет количество итераций открытого перемешивания, после которого начнется повторение строк)

Целое значение в операции $(n\,a) / M$ получится только при $n=M$ (т.к. $a<M$ и для взаимно-простых a и M). Но это означает также, что $na=0$, и, тем самым, $|a|=M$. Отсюда следует $G=G'$.

4. Открытое перемешивание - вставка…

☑ При разрешении коллизии просматриваемые строки могут рассматриваться как список, в котором порядок следования определяется при помощи алгоритмического правила. Тем самым, удаление записи в середине подобного списка не должно разрушать связность записей. Это может быть достигнуто *специальной маркировкой* строк с удаленными записями.

↳ Строка структуры хранения имеет *три возможных состояния – свободное, занятое, пустое* (пустое состояние строки возникает после удаления хранимой в строке записи)

Вставка (окончательный вариант)

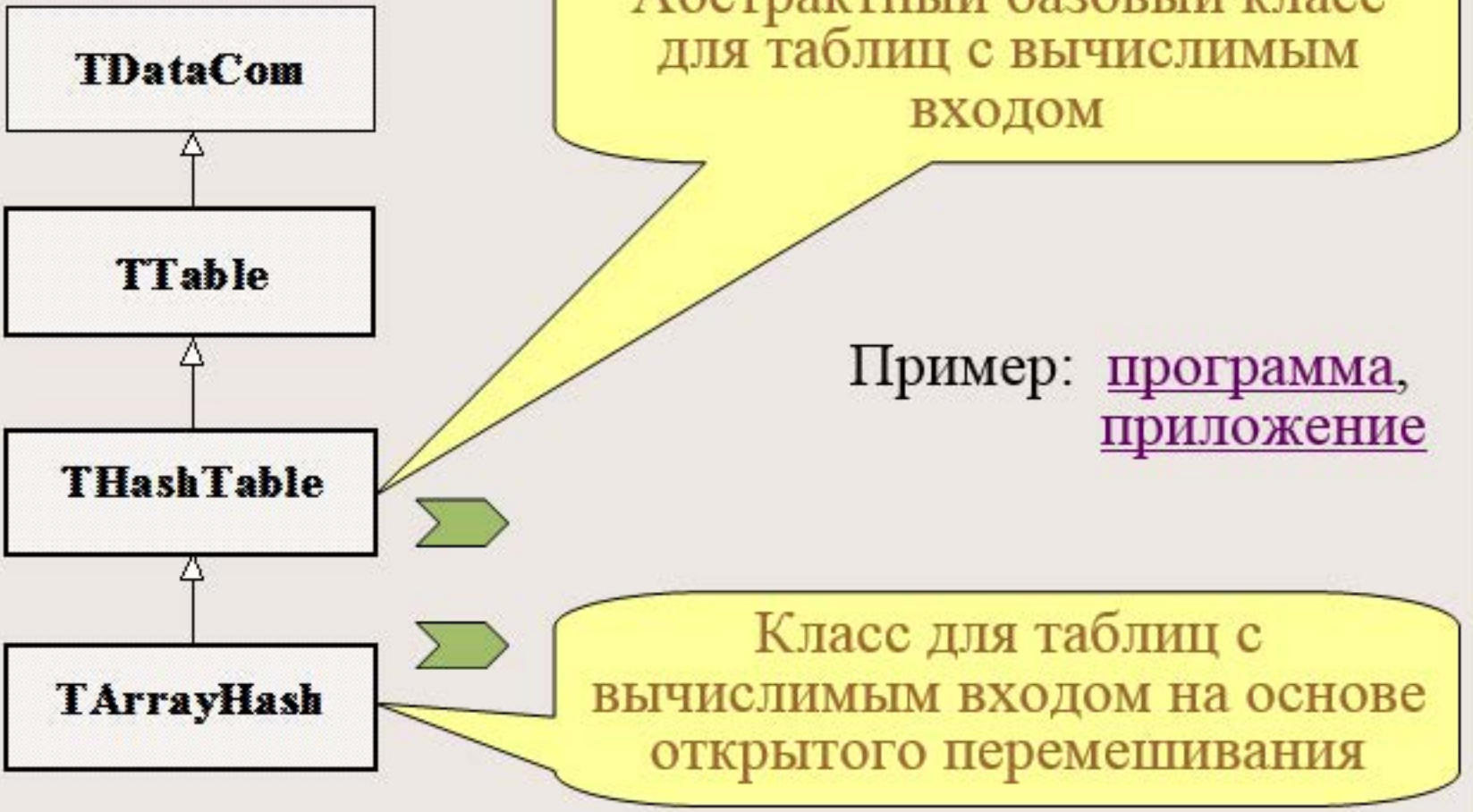
1. Если n==M, ТО { Переполнение; Останов }
2. f=-1 // f – номер первой найденной пустой строки
3. s = h(key) // применение функции расстановки
4. ЕСЛИ s занята и K[s]==key, ТО {Дублир.; Останов }
5. ЕСЛИ s пустая и (f<0), ТО { f = s }
6. ЕСЛИ s свободна и (f < 0), ТО { K[s]=key; Останов }
7. ЕСЛИ s свободна и (f >-1), ТО { K[f]=key; Останов }
8. (!) Коллизия {s = (s+p) mod M и переход к п. 4 }

Поиск

1. f=-1 // f – номер первой найденной пустой строки
2. s = h(key) // применение функции расстановки
3. ЕСЛИ s занята и K[s]==key, ТО { Останов }
4. ЕСЛИ s пустая и (f<0), ТО { f = s }
5. ЕСЛИ s свободна, ТО { Останов }
6. (!) Коллизия { s = (s+p) mod M и переход к п. 3 }

Удаление

1. Поиск записи
2. ЕСЛИ запись найдена,
   ТО { Отметить строку как пустую }


TDataCom
TTable
THashTable
TArrayHash
Абстрактный базовый класс для таблиц с вычислимым входом
Пример: программа, приложение
Класс для таблиц с вычислимым входом на основе открытого перемешивания

6. Оценка эффективности…

- $T_{min} = 1$
- $T_{max} = N$
- $T_{ср} = ?$

6. Оценка эффективности…

Пусть ключи равновероятны и функция хеширования равномерно рассеивает ключи по структуре хранения (M – количество строк памяти, n – количество записей)

Оценим среднее число сравнений при вставке $(k+1)$ ключа

$p_1 = \frac{M-n}{M}$ - **вероятность попадания на пустую строку на 1 сравнении**

$p_2 = \frac{n}{M}\frac{M-n}{M-1}$ - **вероятность попадания на пустую строку на 2 сравнении**

$$p_3 = \frac{n}{M}\frac{n-1}{M-1}\frac{M-n}{M-2}$$

...

$$p_i = \frac{n}{M}\frac{n-1}{M-1}\frac{n-2}{M-2}\cdots\frac{n-i+2}{M-i+2}\frac{M-n}{M-i+1}$$

Тогда

$$E_{n+1} = \sum_{i=1}^{n+1} i p_i = 1 \cdot \frac{M-n}{M} + 2 \cdot \frac{n}{M} \frac{M-n}{M-1} + \ldots +$$

$$+ (n+1)\left(\frac{n}{M} \frac{n-1}{M-1} \frac{n-2}{M-2} \cdots \frac{1}{M-n+1} \right)$$

$$= \frac{M+1}{M-n+1}$$

- ☑ Число сравнений при вставке равно числу сравнений при поиске

Оценим среднее количество сравнений при поиске в таблице с n записями

$$E = \frac{1}{n}\sum_{i=1}^{n} E_i = \frac{M+1}{n}\sum_{i=1}^{n}\frac{1}{M-i+2} = \frac{M+1}{n}\left(H_{M+1} + H_{M-n+1}\right)$$

где $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$ - есть гармоический ряд

$$H_n \cong \gamma + \ln n. \quad \text{(формула Эйлера, } \gamma \cong 0{,}577\text{)}$$

Введем $\alpha = n/(M+1)$ - *индекс заполненности*. Тогда

$$E = \frac{1}{\alpha}\left(\ln(M+1) - \ln(M-n+1)\right) = \frac{1}{\alpha}\ln\frac{M+1}{M-n+1} = \frac{1}{\alpha}\ln(1-\alpha)$$

6. Оценка эффективности

Эффективность зависит не от количества записей, а от степени заполненности структуры хранения (!!!)

α	E	О.П.
0.1	1.05	1.06
0.25	1.15	1.17
0.5	1.39	1.50
0.75	1.85	2.50
0.9	2.56	5.50
0.95	3.15	10.5
0.95	4.66	

Для открытого перемешивания сложность поиска оценивается как

$$E=(1-\alpha/2)/(1-\alpha)$$

Экспериментальная оценка сложности: программа, приложение

7. Разрешение коллизий – метод цепочек…

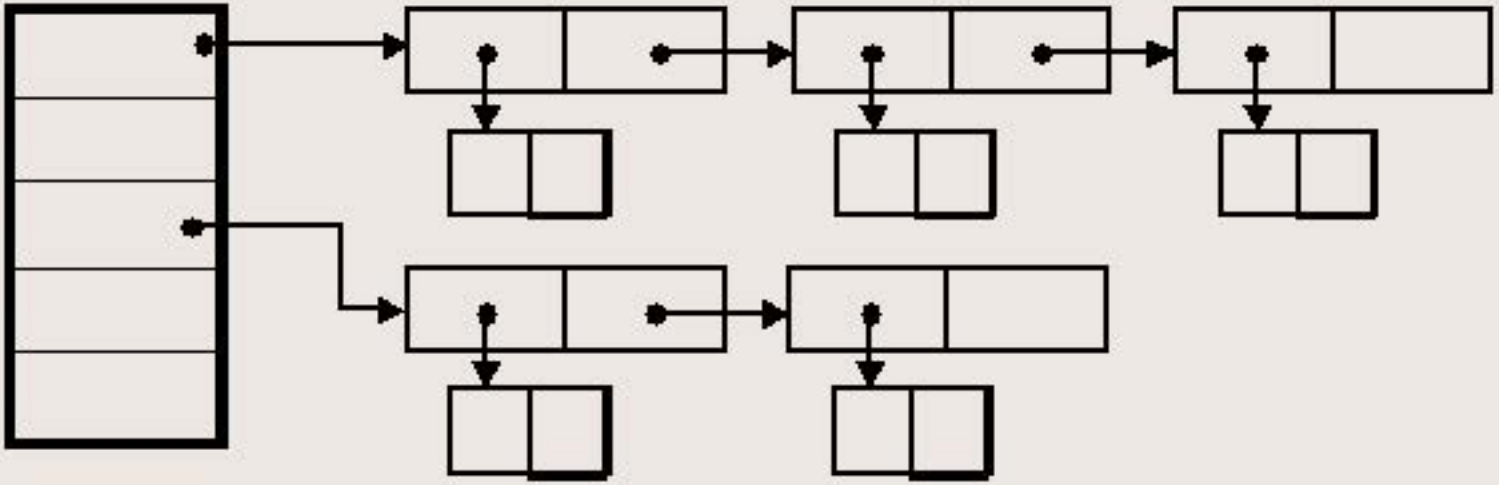
☑ Замечания к открытому перемешиванию как способу разрешения коллизий

- Размер памяти для таблицы фиксирован
- Хранение записей без упорядоченности по ключам

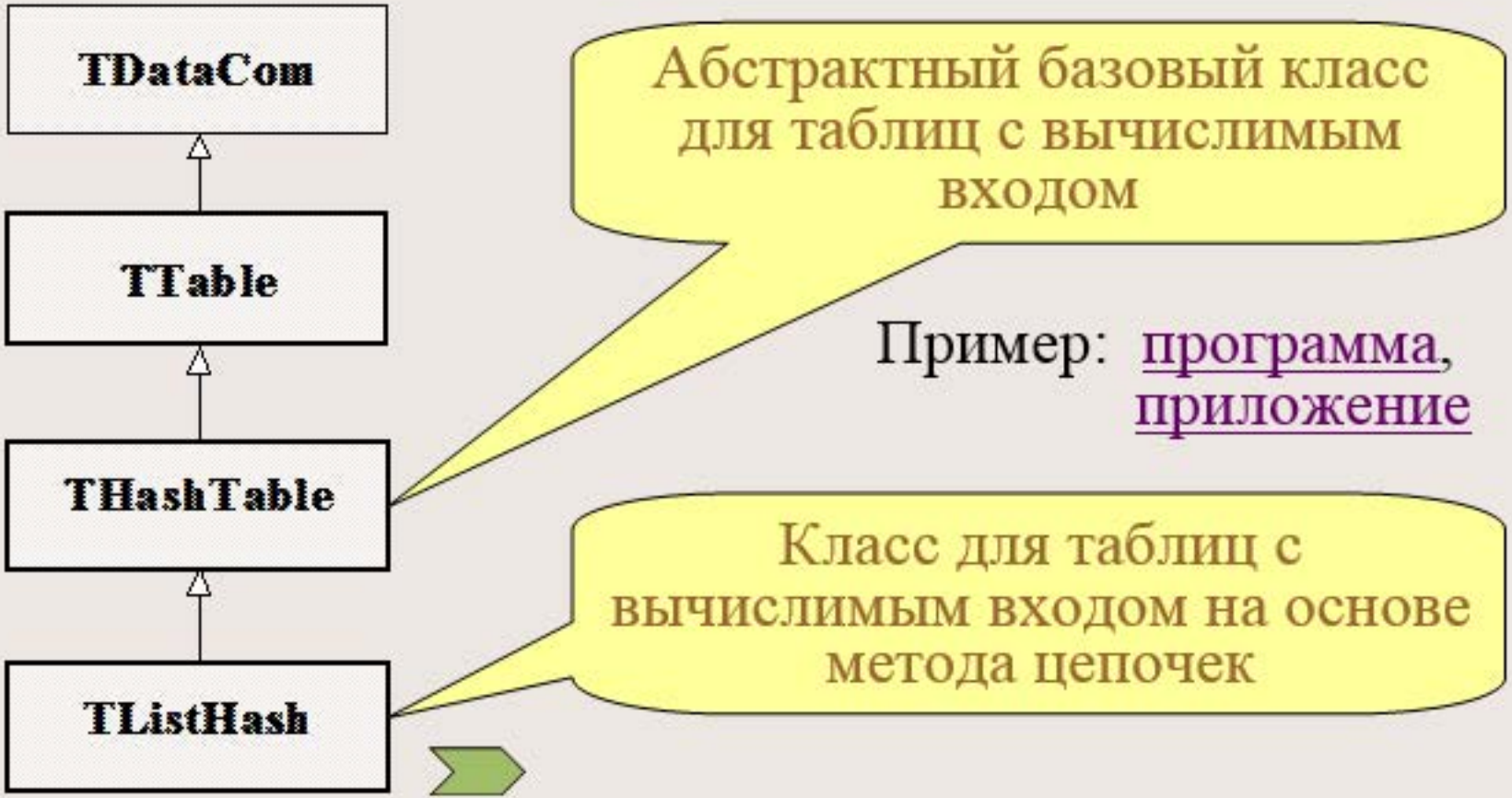
↳Широко используемый подход для разрешения коллизий – *метод цепочек*, когда все записи, для которых функция хеширования определяет одно и тоже значение, представляются в виде линейного списка

☑ Открытое перемешивание еще называют *закрытым хешированием*, метод цепочек – *открытое хеширование*

Структура хранения



Реализация


TDataCom
TTable
THashTable
TListHash
Абстрактный базовый класс для таблиц с вычислимым входом
Пример: программа, приложение
Класс для таблиц с вычислимым входом на основе метода цепочек

7. Заключение

- Использование хеширование позволяет разрабатывать эффективные способы представления таблиц
- Эффективность обработки таблиц с вычислимым входом зависит не от количества записей, а от степени заполненности структуры хранения
- Метод цепочек обеспечивает получение структуры хранения таблицы с динамическим распределением памяти