ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту по дисциплине

«Структуры и алгоритмы обработки данных»

на тему

ФИЛЬТР БЛУМА

|  |  |
| --- | --- |
| Выполнил студент | Григорьев Юрий Вадимович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИС-142 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | ст. преп. Кафедры ВС Д. М. Берлизов |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#__RefHeading___Toc4327_3153689509)

[1 Алгоритмы коллективных обменов информацией в распределённых ВС 4](#__RefHeading___Toc4329_3153689509)

[1.1 Трансляционный обмен информацией в распределённых ВС 4](#__RefHeading___Toc4331_3153689509)

[1.2 Трансляционно-циклический обмен информацией в распределённых ВС 5](#__RefHeading___Toc4333_3153689509)

[1.3 Выводы 5](#__RefHeading___Toc4335_3153689509)

[2 Экспериментальное исследование эффективности алгоритма 6](#__RefHeading___Toc4337_3153689509)

[2.1 Организация моделирования 6](#__RefHeading___Toc4339_3153689509)

[2.2 Результаты моделирования 6](#__RefHeading___Toc4341_3153689509)

[ЗАКЛЮЧЕНИЕ 7](#__RefHeading___Toc4343_3153689509)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#__RefHeading___Toc4345_3153689509)

[ПРИЛОЖЕНИЕ 9](#__RefHeading___Toc4347_3153689509)

[1 Исходный код программы 9](#__RefHeading___Toc4349_3153689509)

1.1 Файл "main.c"……………………………………………………………………

1.2 Файл "bits.c"……………………………………………………………………….

1.3 Файл "bits.h"………………………………………………………………………

1.4 Файл "bloom.c"…………………………………………………………………

1.5 Файл "bloom.h"………………………………………………………………….11

ВВЕДЕНИЕ

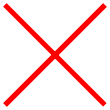
В

1 Алгоритмы коллективных обменов информацией в распределённых ВС

1.1 Трансляционный обмен информацией в распределённых ВС

Текст.

Иллюстрации оформляются по центру. Желательно использовать собственноручно сделанные иллюстрации в векторном формате (*SVG*). На каждый рисунок должна иметься ссылка в тексте отчёта.

Формулы следует набирать в редакторе формул, например . Если на формулу есть ссылка в тексте, она набирается по центру.

|  |  |
| --- | --- |
|  | (1.1) |

Пример оформления таблиц приведён ниже.

Таблица 1.1 – Основные типы данных, определённые в стандарте MPI

|  |  |
| --- | --- |
| Тип данных MPI | Соответствующий тип данных C |
| MPI\_CHAR | char |
| MPI\_SHORT | signed short int |
| MPI\_INT | signed int |
| MPI\_LONG | signed long int |
| MPI\_UNSIGNED | unsigned int |
| MPI\_FLOAT | float |
| MPI\_DOUBLE | double |

1.2 Трансляционно-циклический обмен информацией в распределённых ВС

Текст.

1.3 Выводы

Текст.

2 Экспериментальное исследование эффективности алгоритма

2.1 Организация моделирования

Текст.

2.2 Результаты моделирования

Текст.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы разработан и исследован фильтр Блума.

Осуществлено моделирование разработанной структуры данных. Показано, что

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Burton H. Bloom* Space/time trade-offs in hash coding with allowable errors // Communications of the ACM. – 1970. – T. 13. – №7. – С. 422-426.
2. *Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber* Bigtable: A Distributed Storage System for Structured Data // 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI). – Berkeley: USENIX, 2006. – С. 205-218.
3. *Adam Kirsch, Michael Mitzenmacher* Less Hashing, Same Performance: Building a Better Bloom Filter, edited by Yossi Azar, Thomas Erlebach, Algorithms // ESA 2006, 14th Annual European Symposium, Lecture Notes in Computer Science, Springer-Verlag, Lecture Notes in Computer Science. – 2006. – T. 4168. – С. 456–467.
4. *Alex Yakunin* Nice Bloom filter application // Alex Yakunin's blog URL: blog.alexyakunin.com (дата обращения: 20.11.2022).
5. *Jamie Talbot* What are Bloom filters? // Medium URL: https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff#.xlkqtn1vy (дата обращения: 22.11.2022).
6. *Michael Mitzenmacher, Eli Upfal* Probability and computing: Randomized algorithms and probabilistic analysis. – Cambridge: Cambridge University Press, 2005. – С. 107–112.

ПРИЛОЖЕНИЕ

1 Исходный код программы

1.1 Файл “main.c”

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include "bloom.h"  #define N 10 *// number of elements*  int main() {  BloomFilter \*f = bloom\_init(N);  **for**(int i = 0; i < 2; i++) {  bloom\_insert(f, "apple");  bloom\_insert(f, "coconut");  bloom\_insert(f, "aubergine");  bloom\_insert(f, "pomegranate");  bloom\_insert(f, "pineapple");  bloom\_insert(f, "grapes");  bloom\_insert(f, "orange");  bloom\_insert(f, "banana");  bloom\_insert(f, "potato");  bloom\_insert(f, "cucumber");  }  bloom\_lookup(f, "tomato");  **return** 0;  } |

1.2 Файл “bits.c”

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include "bits.h"  #include <inttypes.h>  void set\_bit(uint8\_t A[], int k)  {  int i = k / 8; *// index in the array A*  int pos = k % 8; *// bit position in A[i]*  unsigned int flag = 1; *// flag 00001*  flag = flag << pos; *// flag shifted k positions (e.g. 00100)*  A[i] = A[i] | flag; *// set the bit at k position in A[i] to 1*  }  void clear\_bit(uint8\_t A[], int k)  {  int i = k / 8;  int pos = k % 8;  unsigned int flag = 1;  flag = flag << pos;  flag = ~flag; *// negative (e.g. 11011)*  A[i] = A[i] & flag;  }  uint8\_t get\_bit(uint8\_t A[], int k)  {  int i = k / 8;  int pos = k % 8;  unsigned int flag = 1;  flag = flag << pos;  **return** ((A[i] & flag) != 0); *// compare xxXxx & 00100 - getting needed bit*  } |

1.3 Файл “bits.h”

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #pragma once  #include <inttypes.h>  void set\_bit(uint8\_t A[], int k);  void clear\_bit(uint8\_t A[], int k);  uint8\_t get\_bit(uint8\_t A[], int k); |

1.4 Файл “bloom.c”

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | #include "bloom.h"  #include "bits.h"  #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #define BITS\_FOR\_TYPE 8.0 *// for uint8\_t*  #define FNV\_32\_PRIME 16777619  #define FALSEPOSITIVE\_PROBABILITY 0.005 *// 0.5%*  char \*bloom\_prefix = "[bloom filter]";  unsigned int bloom\_FNVHash(**const** char \*key, int i) *// fnv-1a*  {  unsigned int hash\_value = 2166136261 - i; *// for different hash functions*  **while**(\*key) {  hash\_value ^= (unsigned int)\*key++;  hash\_value \*= FNV\_32\_PRIME;  }  **return** hash\_value;  }  int bloom\_optimalFilterSize(unsigned int n, double p)*// n - number of elements*  { *// p - probability of false positive*  **return** (unsigned int) ceil(-((double) n \* log(p)) / (log(2) \* log(2)));  }  int bloom\_optimalHashNumber(unsigned int n, unsigned int m)  { *// n - number of elements, m - filter size*  **return** (unsigned int) ceil(((double) m / (double) n) \* log(2));  }  BloomFilter \*bloom\_init(unsigned int n)  {  printf("%s n (elements) = %d**\n**", bloom\_prefix, n);  BloomFilter \*f = malloc(**sizeof**(BloomFilter));  **if**(f != NULL) {  printf("%s malloc \*f - success**\n**", bloom\_prefix);  }  f->m = bloom\_optimalFilterSize(n, FALSEPOSITIVE\_PROBABILITY);  printf("%s m (optimal filter size) = %d**\n**", bloom\_prefix, f->m);  int arraySize = (int) ceil(f->m / BITS\_FOR\_TYPE);  f->bits = malloc(**sizeof**(uint8\_t) \* arraySize);  **if**(f->bits != NULL) {  printf("%s malloc f->bits - success**\n**", bloom\_prefix);  }  **for**(int i = 0; i < arraySize; i++) {  f->bits[i] = 0;  }  f->k = bloom\_optimalHashNumber(n, f->m);  printf("%s k (optimal hash functions number) = %d**\n**", bloom\_prefix, f->k);  **return** f;  }  int bloom\_lookup(BloomFilter \*f, **const** char \*s)  {  **for**(int i = 0; i < f->k; i++) {  **if**(get\_bit(f->bits, bloom\_FNVHash(s, i) % f->m) == 0) {  printf("%s '%s' is not in filter**\n**", bloom\_prefix, s);  **return** 0;  }  }  printf("%s '%s' is probably in filter**\n**", bloom\_prefix, s);  **return** 1;  }  void bloom\_insert(BloomFilter \*f, **const** char \*s)  {  **if**(!bloom\_lookup(f, s)){  **for**(int i = 0; i < f->k; i++) {  set\_bit(f->bits, bloom\_FNVHash(s, i) % f->m);  }  printf("%s successfully inserted '%s'**\n**", bloom\_prefix, s);  }  } |
|  |  |

1.5 Файл “bloom.h”

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #pragma once  #include <inttypes.h>  **typedef** **struct** {  uint8\_t \*bits;  int m; *// filter size*  int k; *// number of hash functions*  } BloomFilter;  unsigned int bloom\_FNVHash(**const** char \*key, int i);  int bloom\_optimalFilterSize(unsigned int n, double p);  int bloom\_optimalHashNumber(unsigned int n, unsigned int m);  BloomFilter \*bloom\_init(unsigned int n);  int bloom\_lookup(BloomFilter \*f, **const** char \*s);  void bloom\_insert(BloomFilter \*f, **const** char \*s); |