

МГТУ им. БАУМАНА

РУБЕЖНЫЙ КОНТРОЛЬ №1

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

Эффективная реализация алгоритма

Работу выполнил: Мокеев Даниил, ИУ7-54

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Хэш-таблицы	3
1.2 Метод цепочек	3
2 Конструкторская часть	5
2.1 Требования к программе	5
Вывод	5
3 Технологическая часть	6
3.1 Выбор ЯП	6
3.2 Сведения о модулях программы	6
3.3 Листинг кода алгоритмов	6
Вывод	7
4 Исследовательская часть	8
4.1 Примеры работы	8
4.2 Исследование зависимости времени работы алгоритмов от размера графа	8
4.3 Выводы исследовательского раздела	8
Заключение	10
Список литературы	10

Задание

Задание существует таблица, содержащая следующие сущности: idPerson, idArticle, EntityTag (Рис. 1). Необходимо выбрать все idPerson, которые работали над одним параграфом.

idPerson	idEntity	EntityTag
1	3	"Event"
2	6	"Person"
...

Рис. 1: Результаты эксперимента

Задачи данной лабораторной работы:

- Разработать эффективный алгоритм решения задачи;
- сравнить его с полным перебором.

1 | Аналитическая часть

Для решения данной задачи было решено использовать структуру данных - хэш-таблицу

1.1 Хэш-таблицы

Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией. Хеш-таблица содержит некоторый массив H , элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками).

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превысит 50

В некоторых специальных случаях удаётся избежать коллизий вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, использующие подобные хеш-функции, не нуждаются в механизме разрешения коллизий, и называются хеш-таблицами с прямой адресацией.

Число хранимых элементов, делённое на размер массива H (число возможных значений хеш-функции), называется коэффициентом заполнения хеш-таблицы (load factor) и является важным параметром, от которого зависит среднее время выполнения операций.

1.2 Метод цепочек

Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

Операции поиска или удаления элемента требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления элемента нужно добавить элемент в конец или начало соответствующего списка и в случае, если коэффициент заполнения станет слишком велик, увеличить размер массива H и перестроить таблицу.

При предположении, что каждый элемент может попасть в любую позицию таблицы H с равной вероятностью и независимо от того, куда попал любой другой

элемент, среднее время работы операции поиска элемента составляет $O(1 + \alpha)$, где α — коэффициент заполнения таблицы.

2 | Конструкторская часть

В данном разделе будут рассмотрены основные требования к программе.

2.1 Требования к программе

Требования к вводу:

- Заранее известно сколько сущностей находится в таблице;
- сущности располагаются в таблице с небольшим разбросом.

Требования к программе:

- Алгоритм возвращает таблицу всех людей, работавших с одним параграфом.

Вывод

В данном разделе были рассмотрены требования к программе.

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран `golang`. Время работы алгоритмов было замерено с помощью `time`.

3.2 Сведения о модулях программы

Программа состоит из:

- `main.go`- главный файл программы, в котором располагается точка входа в программу.
- `perebor.go` - файл содержащий функции измерения времени.

3.3 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода полного перебора всех решений (Листинг 3.1) и реализации муравьиного алгоритма (Листинг 3.2)

Листинг 3.1: Перебор всех возможных вариантов

```
1
2 func get_articles_brute(ids_person , ids_entity []int , amount int) [][]
   int{
3   articles := make([][]int , 0)
4   for i:=0; i<amount; i++){
5     articles[i] = make([]int , 0)
6   }
7   for i:=0; i<len(ids_person); i++){
8     tmp := make([]int , 0)
9     tmp = append(tmp, 0)
10    for j:=0; j<len(ids_person); j++){
11      if ids_entity[j] == ids_entity[j]{
12        tmp = append(tmp, j)
13      }
14    }
15    articles = append(articles , tmp)
16  }
17  return articles
18 }
```

Листинг 3.2: Муравьиный алгоритм

```
1
2 func get_articles_hash(ids_person , ids_entity [] int , amount int) [][]
   int{
3   articles := make([][] int , amount, amount)
4   for i:=0; i<amount; i++{
5     articles[i] = make([] int , 0)
6   }
7   for i:=0; i<amount; i++{
8     for j:=0; j<len(ids_entity); j++{
9       if ids_entity[j] == i{
10        fl := true
11        for k:=0; k < len(articles[i]); k++{
12          if ids_person[j] == articles[i][k]{
13            fl=false
14          }
15        }
16        if fl{
17          articles[i] = append(articles[i], ids_person[j])
18        }
19      }
20    }
21  }
22  return articles
23 }
```

Вывод

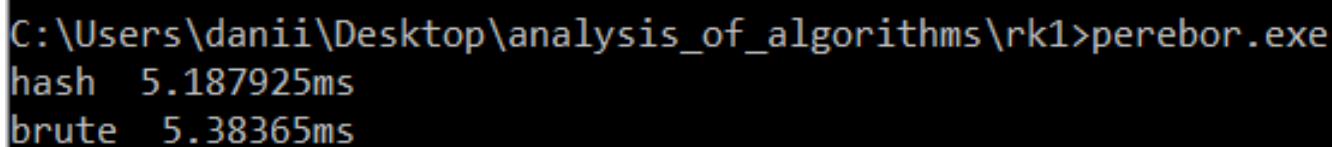
В данном разделе были рассмотрены основные сведения о модулях программы и листинг кода алгоритмов.

4 | Исследовательская часть

В данном разделе будет проведен сравнительный временной анализ алгоритмов и рассмотрена параметризация муравьиного алгоритма. Замеры времени были произведены на: Intel Core i5-6200U.

4.1 Примеры работы

В данном разделе приведен пример работы программы (Рис. 4.1)



```
C:\Users\danii\Desktop\analysis_of_algorithms\rk1>perebor.exe  
hash 5.187925ms  
brute 5.38365ms
```

Рис. 4.1: Пример работы программы

4.2 Исследование зависимости времени работы алгоритмов от размера графа

В данном разделе будут приведены результаты сравнения времени работы реализованных алгоритмов в зависимости от размера Входного файла (Рис. 4.2). Время измерено в миллисекундах.

4.3 Выводы исследовательского раздела

Была исследована зависимость времени работы реализованных алгоритмов от размера матрицы смежности графа. По результатам эксперимента на малых размерах графа полный перебор значительно выигрывает муравьиных алгоритм в скорости, однако на размера графа больше 8 сложность полного перебора растет очень быстро, а так как муравьиный алгоритм обладает полиномиальной сложностью, он работает быстрее перебора.

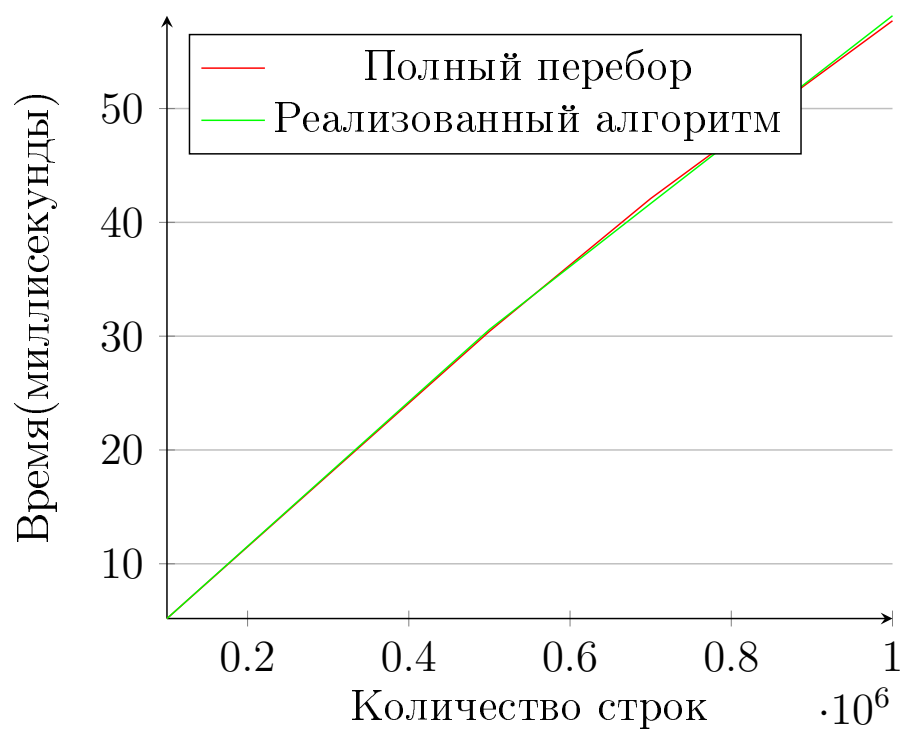


Рис. 4.2: Сравнение параллельного и обычного алгоритмов

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы решения задачи коммивояжера - полный перебор и муравьиный алгоритм.

Временной анализ показал, что неэффективно использовать полный перебор на графе размера больше 8.

Литература

- [1] Белоусов А.И., Ткачев С.Б(2006). Дискретная математика, 4-е издание.
- [2] Т.М. Товстик, Е.В. Жукова - Алгоритм приближенного решения задачи коммивояжера.
- [3] Задача коммивояжера[Электронный ресурс] - режим доступа <http://mech.math.msu.su/shvetz/54/inf/perl-problems/chCommisVoyageur.shtml>
- [4] Муравьиные алгоритмы[Электронный ресурс] - режим доступа <http://www.machinelearning.ru/wiki/index.php?title=>
- [5] Штовба С.Д. - Муравьиные алгоритмы.
- [6] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16