



Министерство науки и высшего образования Российской
Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 6

Дисциплина Анализ алгоритмов

Тема Муравьиный алгоритм

Студент Боренко А. Д.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л.Л.

Содержание

Введение	4
1 Аналитическая часть	6
1.1 Задача коммивояжера	6
1.2 Решение полным перебором	6
1.3 Решение муравьиным алгоритмом	7
1.4 Вывод аналитической части	9
2 Конструкторская часть	10
2.1 Схемы алгоритмов	10
2.2 Структуры данных	12
2.3 Тестирование	13
2.4 Вывод конструкторской части	13
3 Технологическая часть	14
3.1 Требования к ПО	14
3.2 Выбор языка программирования	14
3.3 Структуры данных	14
3.4 Реализация алгоритмов	15
3.5 Тестирование	18
3.6 Вывод технологической части	19
4 Экспериментальная часть	20
4.1 Технические характеристики	20
4.2 Примеры работы	20
4.3 Замеры времени	21
4.4 Параметризация муравьиного алгоритма	22
4.5 Вывод экспериментальной части	23

Заключение	23
Список литературы	25
Приложение	26

Введение

Муравья нельзя назвать сообразительным. Отдельный муравей не в состоянии принять ни малейшего решения. Дело в том, что он устроен крайне примитивно: все его действия сводятся к элементарным реакциям на окружающую обстановку и своих собратьев. Муравей не способен анализировать, делать выводы и искать решения.

Эти факты, однако, никак не согласуются с успешностью муравьев как вида. Они существуют на планете более 100 миллионов лет, строят огромные жилища, обеспечивают их всем необходимым и даже ведут настоящие войны. В сравнении с полной беспомощностью отдельных особей, достижения муравьев кажутся немыслимыми.

Добиться таких успехов муравьи способны благодаря своей социальности. Они живут только в коллективах – колониях. Все муравьи колонии формируют так называемый роевой интеллект. Особи, составляющие колонию, не должны быть умными: они должны лишь взаимодействовать по определенным – крайне простым – правилам, и тогда колония целиком будет эффективна.

В колонии нет доминирующих особей, нет начальников и подчиненных, нет лидеров, которые раздают указания и координируют действия. Колония является полностью самоорганизующейся. Каждый из муравьев обладает информацией только о локальной обстановке, не один из них не имеет представления обо всей ситуации в целом – только о том, что узнал сам или от своих сородичей, явно или неявно. На неявных взаимодействиях муравьев, называемых стигмергией, основаны механизмы поиска кратчайшего пути от муравейника до источника пищи.

Каждый раз проходя от муравейника до пищи и обратно, муравьи оставляют за собой дорожку феромонов. Другие муравьи, почувствовав такие следы на земле, будут инстинктивно устремляться к нему. Поскольку эти муравьи тоже оставляют за собой дорожки феромонов, то чем больше муравьев проходит по определенному пути, тем более привлекательным он становится для их сородичей. При этом, чем короче путь до источника пищи, тем меньше времени требу-

ется муравьям на него – а следовательно, тем быстрее оставленные на нем следы становятся заметными.

В 1992 году в своей диссертации Марко Дориго (Marco Dorigo) предложил заимствовать описанный природный механизм для решения задач оптимизации. Имитируя поведение колонии муравьев в природе, муравьиные алгоритмы используют многоагентные системы, агенты которых функционируют по крайне простым правилам. Они крайне эффективны при решении сложных комбинаторных задач – таких, например, как задача коммивояжера, первая из решенных с использованием данного типа алгоритмов.

Целью данной работы является изучение и реализация двух алгоритмов:

1. полный перебор;
2. муравьиный алгоритм.

Задачами данной лабораторной являются:

1. исследование выше описанных алгоритмов для решения задачи коммивояжера;
2. реализация исследуемых алгоритмов;
3. сравнительный анализ реализованных алгоритмов;
4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1. Аналитическая часть

В данном разделе будут рассмотрены формальное описание алгоритмов.

1.1 Задача коммивояжера

В 19-м и 20-м веке по городам ездили коммивояжеры (сейчас их называют "торговые представители"). Они ходили по домам и предлагали людям купить разные товары. Тактика была такой: коммивояжёр приезжал в город, обходил большинство домов и отправлялся в следующий город. Города были небольшими, поэтому обойти всё было вполне реально. Чем больше городов посетит коммивояжёр, тем больше домов он сможет обойти и больше заработать с продаж. В задаче коммивояжера рассматривается n городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса.

1.2 Решение полным перебором

Эту задачу возможно решить полным перебором т. е. разобрать все возможные варианты и выбрать оптимальный. Но проблема такого решения в том, что с увеличением количества городов, время выполнения будет расти. Хотя такой подход и гарантирует точное решение задачи, уже при небольшом числе городов решение задачи за допустимое время не возможно.

1.3 Решение муравьиным алгоритмом

В то время как простой метод перебора всех вариантов чрезвычайно неэффективный при большом количестве городов, эффективными признаются решения, гарантирующие получение ответа за время, ограниченное полиномом от размерности задачи. В основе алгоритма лежит поведение муравьиной колонии – маркировка более удачных путей большим количеством феромона.

Каждый муравей хранит в памяти список пройденных им узлов. Этот список называют списком запретов (tabu list) или просто памятью муравья. Выбирая узел для следующего шага, муравей "помнит" об уже пройденных узлах и не рассматривает их в качестве возможных для перехода. На каждом шаге список запретов пополняется новым узлом, а перед новой итерацией алгоритма – то есть перед тем, как муравей вновь проходит путь – он опустошается.

Кроме списка запретов, при выборе узла для перехода муравей руководствуется $\frac{3}{4}$ привлекательностью ребер, которые он может пройти. Она зависит, во-первых, от расстояния между узлами (то есть от веса ребра), а во-вторых, от следов феромонов, оставленных на ребре прошедшими по нему ранее муравьями. Естественно, что в отличие от весов ребер, которые являются константными, следы феромонов обновляются на каждой итерации алгоритма: как и в природе, со временем следы испаряются, а проходящие муравьи, напротив, усиливают их.

Вероятность перехода из вершины i в вершину j определяется по формуле 1.1

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in J} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} \quad (1.1)$$

где

- τ_{ij} - расстояние от города i до j ,
- η_{ij} - количество феромонов на ребре ij ,
- α - параметр влияния длины пути,
- β - параметр влияния феромона.

Уровень феромона обновляется в соответствии с формулой 1.2.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij} \quad (1.2)$$

где

- ρ - доля феромона, которая испарится,
- τ_{ij} - количество феромона на дуге ij ,
- $\Delta\tau_{ij}$ - количество отложенного феромона, вычисляется по формуле 1.3.

$$\Delta\tau_{ij} = \tau_{i,j}^0 + \tau_{i,j}^1 + \dots + \tau_{i,j}^k \quad (1.3)$$

где

- где k - количество муравьев в вершине графа с индексами i и j .

Описание поведения муравьев при выборе пути:

- Муравьи имеют собственную "память". Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через $J_i k$ список городов, которые необходимо посетить муравью k , находящемуся в городе i .
- Муравьи обладают "зрением" видимость есть эвристическое желание посетить город j , если муравей находится в городе i . Будем считать, что видимость обратно пропорциональна расстоянию между городами.
- Муравьи обладают "обонянием" они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{ij}(t)$.
- Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано формулой 1.3.

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & \text{если } k\text{-ый муравей прошел по ребру } ij \\ 0 & \text{иначе} \end{cases}$$

где Q - количество феромона, переносимого муравьем.

1.4 Вывод аналитической части

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации алгоритмов для решения задачи коммивояжера.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут описаны пользовательские структуры данных, приведены классы эквивалентности для тестирования реализуемого ПО.

2.1 Схемы алгоритмов

2.1.1 Схема алгоритма полного перебора

На рисунке 2.1 показана схема алгоритма полного перебора.

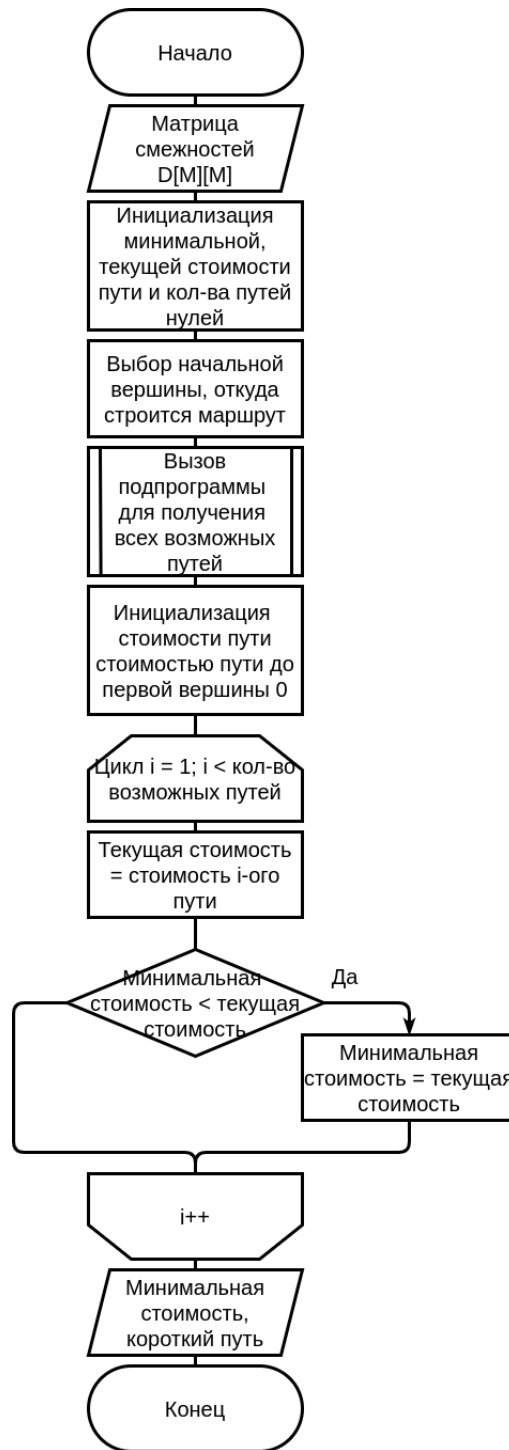


Рис. 2.1: Схема алгоритма полного перебора

2.1.2 Схема муравьиного алгоритма

На рисунке 2.2 показана схема муравьиного алгоритма.

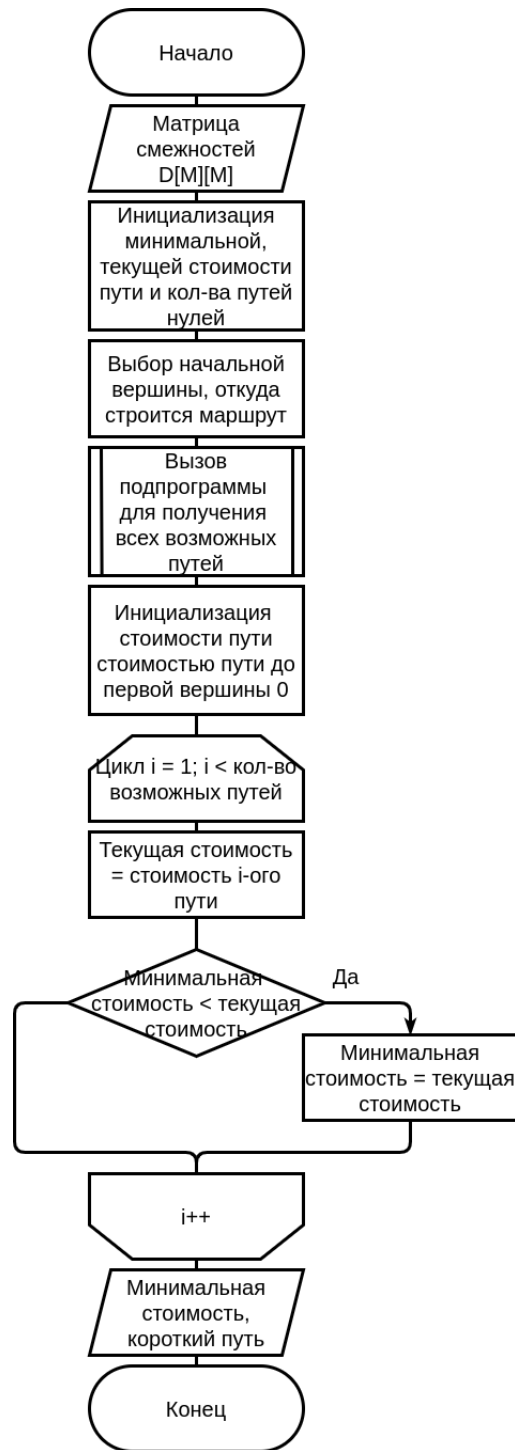


Рис. 2.2: Схема муравьиного алгоритма

2.2 Структуры данных

При реализации приведенных алгоритмов потребуются типы данных: массив, матрица, муравей, колония.

1. D - матрица смежности

2. T_{ao} - матрица феромонов
3. α - приоритет пути
4. β - приоритет феромона
5. q - переносимый муравьем феромон
6. p - коэффициент испарения
7. t_{max} - максимальное время жизни колонии

2.3 Тестирование

Для алгоритма умножения матриц можно выделить следующие классы эквивалентности:

1. ациклический ориентированный взвешенный граф;
2. циклический ориентированный взвешенный граф.

2.4 Вывод конструкторской части

На основе данных, полученных в аналитическом разделе, были построены схемы используемых алгоритмов, выделены необходимые для реализации структуры данных и методы тестирования.

3. Технологическая часть

3.1 Требования к ПО

Требования к программному обеспечению:

1. на вход подается матрица смежностей;
2. результат: список вершин - кратчайший путь в графе.

3.2 Выбор языка программирования

Был выбран язык go, поскольку он удовлетворяет требованиям задачи. Средой разработки выбрана Visual Studio Code.

3.3 Структуры данных

На листинге 3.1 представлено описание структуры муравья.

Листинг 3.1: Структура муравья

```
1 func mremove(s []int, i int) []int {  
2     return append(s[:i], s[i+1:]...)  
3 }  
4 type ant_t struct{  
5     J []int //cities_to_go  
6     end int
```

На листинге 3.2 представлено описание структуры колонии.

Листинг 3.2: Структура колонии

```
1 pos int
2 }
3
4 type colony_t struct{
5     D [][] float64
6     Tao [][] float64
7     alpha float64
8     betta float64
9     q float64
```

3.4 Реализация алгоритмов

На листинге 3.3 представлена реализация алгоритма полного перебора.

Листинг 3.3: Реализация алгоритма полного перебора

```
1 func search_all(contiguity float_matrix_t, min_weight*float64) []int{
2     var routs = generate_routs_arr(len(contiguity))
3     var rout = make_int_array(len(contiguity))
4
5     *min_weight = -1
6
7     for i:=0;i<len(routs);i++){
8         var cur_weight float64 = 0
9         var flag_success_route bool = contiguity.get_rout_weight(routs[i], &
10             cur_weight)
11
12         if (flag_success_route == true && (cur_weight < *min_weight || *
13             min_weight == -1)){
14             *min_weight = cur_weight
15             copy(rout, routs[i])
16         }
17     }
18     return rout
19 }
```

На листингах 3.4, 3.5, 3.6 представлена реализация муравьиного алгоритма.

Листинг 3.4: Реализация муравьиного алгоритма ч.1

```

1 func get_route(e colony_t, a *ant_t, l_k *float64)int{
2     var p_ijk = make([]float64, len(a.J))
3     var chis float64
4     chis = 1
5     var sum float64 = get_spec_summ(e, *a)
6     if (math.Abs(sum) < EPS){
7         return -1
8     }
9
10    for i:=0;i<len(a.J);i++){
11        if (math.Abs(e.D[a.pos][a.J[i]])<=EPS){
12            if (i > 0){
13                p_ijk[i] = p_ijk[i-1]
14            } else{
15                p_ijk[i] = 0
16            }
17            continue
18        }
19        var n_ij = 1/e.D[a.pos][a.J[i]]
20        var pheromon = e.Tao[a.pos][a.J[i]]
21        chis = math.Pow(pheromon, e.alpha) * math.Pow(n_ij, e.betta)
22        if (i > 0){
23            p_ijk[i] = p_ijk[i-1] + chis/sum*100
24        } else{
25            p_ijk[i] = chis/sum*100
26        }
27    }
28    var rand_num = rand.Intn(100)
29
30    for i:=0;i<len(p_ijk);i++){
31        if (float64(rand_num) <= p_ijk[i]){
32            var result = a.J[i]
33            a.J = mremove(a.J, i)
34            *l_k = e.D[a.pos][result]
35            a.pos = result
36
37            return result
38        }

```


Листинг 3.5: Реализация муравьиного алгоритма ч.2

```

1      }
2    }
3    return -1
4  }
5
6  func (e *colony_t)update_pheromons(a []ant_t, l[]float64) {
7    var del_t float64
8    del_t = 0
9    for k:=0;k<len(e.Tao);k++){
10     for i:=0;i< len(e.Tao[k]);i++){
11       if e.D[k][i] != 0{
12         if l[k] >0{
13           del_t = e.q / float64(e.D[k][i])
14         } else {
15           del_t = 0
16         }
17         e.Tao[k][i] = (1-e.p) * (float64(1)) + del_t
18       }
19       if e.Tao[k][i] <= 0{
20         e.Tao[k][i] = 0.1
21       }
22     }
23   }
24
25 }
26
27 func ant_alg(e colony_t, l_res *float64)[]int{
28   rand.Seed(time.Now().UnixNano())
29   var ants []ant_t
30   var l_k_res []float64 = make([]float64, len(e.D))
31   var t_res []int
32   *l_res = -1
33   for i:=0;i<e.t_max;i++){
34     ants = e.generate_all_ants()
35     for j:=0;j<len(ants);j++){
36       var l_k float64 = 0
37       var t_k[]int
38       t_k = append(t_k, ants[j].pos)

```

Листинг 3.6: Реализация муравьиного алгоритма ч.3

```
1      t_k = append(t_k, ants[j].pos)
2      var succes_route = true
3      for l:=0;len(ants[j].J)>0; l++{
4          var l_k_cur float64
5          t_k = append(t_k,get_route(e, &(amp;ants[j])), &l_k_cur))
6          if (t_k[len(t_k)-1] == -1){
7              //fmt.Println("BAD ERROR, cant go")
8              succes_route = false
9              l_k_res[j] = -1
10             break
11         }
12         l_k += l_k_cur
13     }
14     if (succes_route){
15         if (e.D[t_k[0]][t_k[len(t_k)-1]] <=0){
16             //fmt.Println("BAD ERROR, cant go")
17             succes_route = false
18             l_k_res[j] = -1
19             continue
20         }
21         l_k += e.D[t_k[0]][t_k[len(t_k)-1]]
22         l_k_res[j] = l_k
23         if (*l_res > l_k || *l_res == -1){
24             *l_res = l_k
25             t_res = t_k
26         }
27     }
28
29 }
30 e.update_pheromons(ants, l_k_res)
31 }
32 return t_res
33 }
```

3.5 Тестирование

Модульные тесты

В таблице 3.1 представленные тесты.

Таблица 3.1 – Тесты

N^o	Ввод					Вывод
1	0	3	1	6	8	15
	3	0	4	1	0	
	1	4	0	5	0	
	6	1	5	6	1	
	8	0	0	1	1	
2	0	10	15	20		80
	10	0	35	25		
	15	35	0	30		
	20	25	30	0		

Тесты пройдены.

3.6 Вывод технологической части

Были реализованы исследуемые алгоритмы, программа прошла тесты и удовлетворяет требованиям.

4. Экспериментальная часть

Оценка качества работы алгоритмов. Экспериментальное сравнение работы различных алгоритмов нахождения среднего арифметического матрицы (зависимость времени выполнения от размерности матриц).

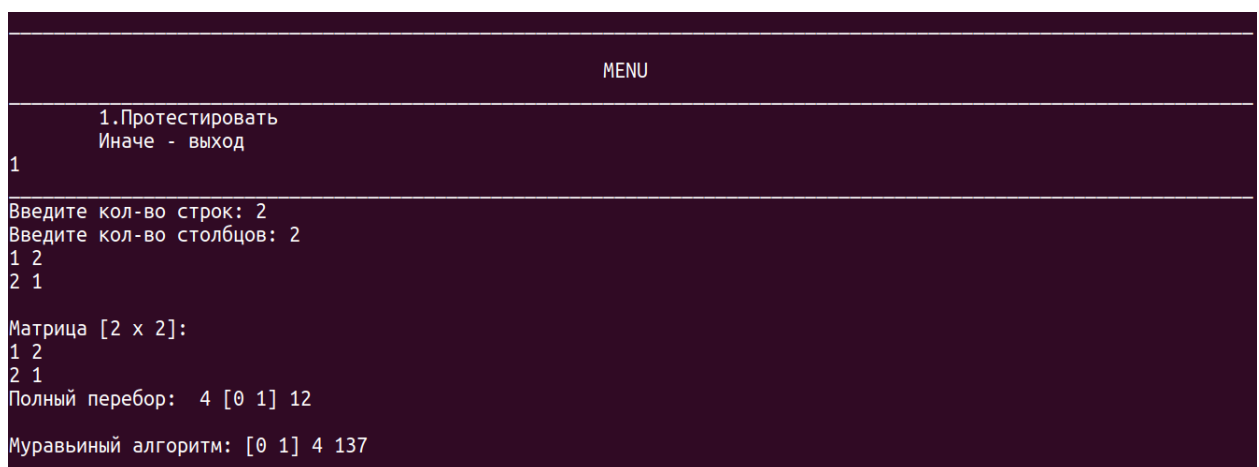
4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

1. процессор: Intel® Core™ i3-7100U CPU @ 2.40GHz × 4;
2. память: 11,6 GiB;
3. операционная система: Ubuntu 20.04.1 LTS.

4.2 Примеры работы

На рисунках 4.1, 4.2 показаны примеры работы.



```
MENU
1.Протестировать
Иначе - выход
1
Введите кол-во строк: 2
Введите кол-во столбцов: 2
1 2
2 1
Матрица [2 x 2]:
1 2
2 1
Полный перебор: 4 [0 1] 12
Муравьиный алгоритм: [0 1] 4 137
```

Рис. 4.1: Пример 1

MENU		
1.Протестировать 2.Общее тестирование Иначе - выход		
2		
size	func_name	time
2	all_search	12.100000
2	ant_search	134.000000
3	all_search	15.200000
3	ant_search	223.800000
4	all_search	27.200000
4	ant_search	357.600000
5	all_search	74.800000
5	ant_search	755.400000
6	all_search	360.300000
6	ant_search	1132.800000
7	all_search	1433.100000
7	ant_search	501.600000
8	all_search	12348.600000
8	ant_search	668.100000
9	all_search	120832.700000
9	ant_search	1051.100000
10	all_search	2717589.600000
10	ant_search	7412.900000

Обновление приложений

Рис. 4.2: Пример 2

4.3 Замеры времени

Таблица 4.1 содержит результаты замеров времени при 1, 2, 4, 8, 16, 32 потоках

На рисунках ?? показаны графические результаты сравнения исследуемых алгоритмов по времени. По оси X - размер матрицы смежности, по оси y - время выполнения в нсек.

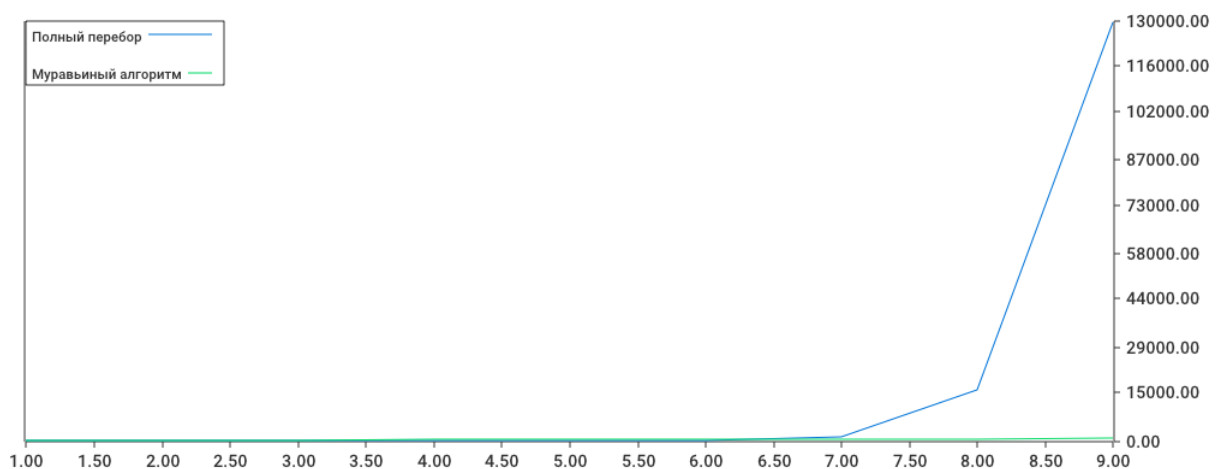


Рис. 4.3: Сравнение алгоритма полного перебора и муравьиного алгоритма

Таблица 4.1 – Замеры времени (в нсек)

Размер матрицы	Название алгоритма	time
2	Полный перебор	12.100000
2	Муравьиный алгоритм	134.000000
3	Полный перебор	15.200000
3	Муравьиный алгоритм	223.800000
4	Полный перебор	27.200000
4	Муравьиный алгоритм	357.600000
5	Полный перебор	74.800000
5	Муравьиный алгоритм	755.400000
6	Полный перебор	360.300000
6	Муравьиный алгоритм	1132.800000
7	Полный перебор	1433.100000
7	Муравьиный алгоритм	501.600000
8	Полный перебор	12348.600000
8	Муравьиный алгоритм	668.100000
9	Полный перебор	120832.700000
9	Муравьиный алгоритм	1051.100000
10	Полный перебор	2717589.600000
10	Муравьиный алгоритм	7412.900000

4.4 Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Рассмотрим матрицу смежностей размерностью 10×10 .

Результаты тестирования представлены в таблице 4.3

Параметризация метода решения задачи коммивояжера на основании муравьиного алгоритма проводилась для матрицы с элементами в диапазоне $[0, 2500]$. Количество дней было равно 50. Полный перебор определил оптимальную длину пути 6986. Столбец "результат" отвечает за результат работы муравьиного алгоритма. Столбец "разница" отвечает за разницу с оптимальной длиной.

Таблица 4.2 – Тестовая матрица

0	0	1	2	3	4	5	6	7	8	9
0	0	1790	200	1900	63	1659	1820	1395	2382	649
1	1790	0	1573	2435	1515	714	892	2193	1590	1003
2	200	1573	0	833	392	2404	962	902	141	1123
3	1900	2435	833	0	2283	1652	2362	2262	1512	2166
4	63	1515	392	2283	0	1322	290	1305	2100	969
5	1659	714	2404	1652	1322	0	256	78	2236	2041
6	1820	892	962	2362	290	256	0	1180	1547	1279
7	1395	2193	902	2262	1305	78	1180	0	1640	1161
8	2382	1590	141	1512	2100	2236	1547	1640	0	2212
9	649	1003	1123	2166	969	2041	1279	1161	2212	0

Таблица 4.3 – Тестовая матрица

alpha	beta	p	Результат	Разница
0.0	1.0	0.7	6986.0	0.0
0.0	1.0	0.8	6986.0	0.0
0.0	1.0	0.9	6986.0	0.0
0.0	1.0	1.0	6986.0	0.0
0.5	0.5	0.7	7139.0	153.0
0.5	0.5	0.8	7479.0	493.0
0.5	0.5	0.9	7139.0	153.0
0.5	0.5	1.0	6986.0	0.0
1.0	0.0	0.5	8277.0	1291.0
1.0	0.0	0.6	7329.0	343.0
1.0	0.0	0.7	8185.0	1199.0
1.0	0.0	0.8	6986.0	0.0

4.5 Вывод экспериментальной части

На основе проведенной параметризации (таблицы 4.3, 4.4) для матрицы смежности приведенной в таблице 4.2 рекомендуется использовать ($\alpha < \beta$, $\rho = \text{любое}$). При параметрах $\alpha = 0$, $\beta = 1$, количество правильно найденных оптимальных путей составило 8 единиц. При $\alpha = 0$, $\beta = 1$ 8 из 10 найденных путей были оптимальные.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при реализации алгоритма полного перебора и муравьиного алгоритма. Были рассмотрены схемы для решения задачи коммивояжера. Также были разобраны листинги , показывающие работу, описанных выше алгоритмов. Был произведен сравнительный анализ. Выполнены поставленные задачи.

Список литературы

- [1] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.12.2021.
- [2] Linux – Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.12.2021.
- [3] Макконнел. Дж. Анализ алгоритмов. Активный обучающий подход. – М. Режим доступа: <https://www.linux.org.ru/>. 2017. 267 с.
- [4] Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2004. No 4.
- [5] Go Documentation [Электронный ресурс]. Режим доступа: <https://go.dev/doc/>. Дата обращения: 29.01.2022.
- [6] Gratzner George A. More Math Into LaTeX. 4th изд. Boston: Birkhauser, 2007.

Приложение

Таблица 4.4 – Таблица коэффициентов. Часть 1

alpha	beta	p	Результат	Разница
0.0	1.0	0.0	6992.0	6.0
0.0	1.0	0.1	7139.0	153.0
0.0	1.0	0.2	6986.0	0.0
0.0	1.0	0.3	6986.0	0.0
0.0	1.0	0.4	6992.0	6.0
0.0	1.0	0.5	6986.0	0.0
0.0	1.0	0.6	6986.0	0.0
0.0	1.0	0.7	6986.0	0.0
0.0	1.0	0.8	6986.0	0.0
0.0	1.0	0.9	6986.0	0.0
0.0	1.0	1.0	6986.0	0.0
0.1	0.9	0.0	6986.0	0.0
0.1	0.9	0.1	6992.0	6.0
0.1	0.9	0.2	6986.0	0.0
0.1	0.9	0.3	6986.0	0.0
0.1	0.9	0.4	6992.0	6.0
0.1	0.9	0.5	7139.0	153.0
0.1	0.9	0.6	6986.0	0.0
0.1	0.9	0.7	6986.0	0.0
0.1	0.9	0.8	6986.0	0.0
0.1	0.9	0.9	6986.0	0.0
0.1	0.9	1.0	6986.0	0.0
0.2	0.8	0.0	6986.0	0.0
0.2	0.8	0.1	6992.0	6.0

0.2	0.8	0.2	7139.0	153.0
0.2	0.8	0.3	6992.0	6.0
0.2	0.8	0.4	6992.0	6.0
0.2	0.8	0.5	7139.0	153.0
0.2	0.8	0.6	6986.0	0.0
0.2	0.8	0.7	6986.0	0.0
0.2	0.8	0.8	6986.0	0.0
0.2	0.8	0.9	6986.0	0.0
0.2	0.8	1.0	6992.0	6.0
0.3	0.7	0.0	7139.0	153.0
0.3	0.7	0.1	7329.0	343.0
0.3	0.7	0.2	6986.0	0.0
0.3	0.7	0.3	6986.0	0.0
0.3	0.7	0.4	7139.0	153.0
0.3	0.7	0.5	6986.0	0.0
0.3	0.7	0.6	6992.0	6.0
0.3	0.7	0.7	6992.0	6.0
0.3	0.7	0.8	7139.0	153.0
0.3	0.7	0.9	6992.0	6.0
0.3	0.7	1.0	7139.0	153.0
0.4	0.6	0.0	7139.0	153.0
0.4	0.6	0.1	6986.0	0.0
0.4	0.6	0.2	7139.0	153.0
0.4	0.6	0.3	7139.0	153.0
0.4	0.6	0.4	7329.0	343.0
0.4	0.6	0.5	7217.0	231.0
0.4	0.6	0.6	7139.0	153.0
0.4	0.6	0.7	7139.0	153.0
0.4	0.6	0.8	6986.0	0.0
0.4	0.6	0.9	7139.0	153.0
0.4	0.6	1.0	6986.0	0.0
0.5	0.5	0.0	6986.0	0.0
0.5	0.5	0.1	6992.0	6.0

0.5	0.5	0.2	7139.0	153.0
0.5	0.5	0.3	6986.0	0.0
0.5	0.5	0.4	7318.0	332.0
0.5	0.5	0.5	7217.0	231.0
0.5	0.5	0.6	7315.0	329.0
0.5	0.5	0.7	7139.0	153.0
0.5	0.5	0.8	7479.0	493.0
0.5	0.5	0.9	7139.0	153.0
0.5	0.5	1.0	6986.0	0.0
0.6	0.4	0.0	7139.0	153.0
0.6	0.4	0.1	7329.0	343.0
0.6	0.4	0.2	7898.0	912.0
0.6	0.4	0.3	7139.0	153.0
0.6	0.4	0.4	7407.0	421.0
0.6	0.4	0.5	7165.0	179.0
0.6	0.4	0.6	7562.0	576.0
0.6	0.4	0.7	7248.0	262.0
0.6	0.4	0.8	7426.0	440.0
0.6	0.4	0.9	6992.0	6.0
0.6	0.4	1.0	6992.0	6.0
0.7	0.3	0.0	6992.0	6.0
0.7	0.3	0.1	7139.0	153.0
0.7	0.3	0.2	7838.0	852.0
0.7	0.3	0.3	7347.0	361.0
0.7	0.3	0.4	7248.0	262.0
0.7	0.3	0.5	7616.0	630.0
0.7	0.3	0.6	7165.0	179.0
0.7	0.3	0.7	7388.0	402.0
0.7	0.3	0.8	7315.0	329.0
0.7	0.3	0.9	7176.0	190.0
0.7	0.3	1.0	6986.0	0.0
0.8	0.2	0.0	7873.0	887.0
0.8	0.2	0.1	7139.0	153.0

0.8	0.2	0.2	7407.0	421.0
0.8	0.2	0.3	7722.0	736.0
0.8	0.2	0.4	7479.0	493.0
0.8	0.2	0.5	7388.0	402.0
0.8	0.2	0.6	7898.0	912.0
0.8	0.2	0.7	7612.0	626.0
0.8	0.2	0.8	7349.0	363.0
0.8	0.2	0.9	6986.0	0.0
0.8	0.2	1.0	6986.0	0.0
0.9	0.1	0.0	7349.0	363.0
0.9	0.1	0.1	7724.0	738.0
0.9	0.1	0.2	8740.0	1754.0
0.9	0.1	0.3	7347.0	361.0
0.9	0.1	0.4	7911.0	925.0
0.9	0.1	0.5	6992.0	6.0
0.9	0.1	0.6	8125.0	1139.0
0.9	0.1	0.7	7139.0	153.0
0.9	0.1	0.8	7479.0	493.0
0.9	0.1	0.9	7388.0	402.0
0.9	0.1	1.0	6992.0	6.0
1.0	0.0	0.0	7881.0	895.0
1.0	0.0	0.1	7874.0	888.0
1.0	0.0	0.2	8284.0	1298.0
1.0	0.0	0.3	8272.0	1286.0
1.0	0.0	0.4	8032.0	1046.0
1.0	0.0	0.5	8277.0	1291.0
1.0	0.0	0.6	7329.0	343.0
1.0	0.0	0.7	8185.0	1199.0
1.0	0.0	0.8	6986.0	0.0
1.0	0.0	0.9	7602.0	616.0
1.0	0.0	1.0	6986.0	0.0