**Министерство образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

Факультет: Информатика и системы управления

Кафедра: Информационная безопасность (ИУ8)

**Аппаратные средства вычислительной техники**

**Домашнее задание №1 на тему:**

«Минимизация функций алгебры логики»

**Преподаватель:**

Рафиков А.Г.

**Студент**:

Кудрявцев И. А.

**Группа:**

ИУ8-65

Москва 2020

**Цель работы:**

Минимизировать исходную функцию алгебры логики (ФАЛ) тремя методами: методом карт Карно, Квайна – Мак-Класски и методом неопределённых коэффициентов.

**Ход работы:**

1. **Метод Квайна – Мак-Класски**

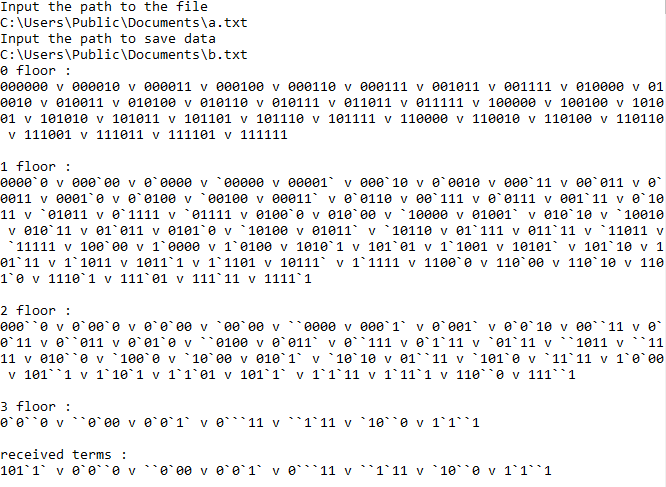
Данные метод состоит из нескольких этапов:

1. Сначала находят первичные импликанты. Для этогомономы ДНФ группируют согласно их весам на соответствующие группы. Затем попарно сравниваются наборы из соседних групп. Если наборы соседние, то они склеиваются, а переменную, по которой происходила склейка, заменяют символом ‘~’, получая при этом терм меньшей степени. Процесс склейки продолжается поэтапно до того момента, когда не останется возможных вариантов для склейки термов. Полученные термы и есть первичные импликанты.
2. Далее формируется таблица. Её столбцам соответствуют наборы из ДНФ, а строкам – первичные импликанты, полученные на предыдущем этапе. На пересечении строки i и столбца j ставится метка, если j-ый набор может быть поглощен соответствующей i-ой импликантой.
3. Следующим этапом является поиск существенных импликант. В полученной таблице ищут столбцы только с одной меткой и вычеркивают их. Соответствующая метке импликанта (является существенной), вместе со столбцом, имеющим одну метку, покрывает всю строку таблицы с сопутствующими метками. Столбцы с сопутствующими метками также вычеркиваются из таблицы, поскольку они тоже будут покрыты этой импликантой. Лишние первичные импликанты (все метки которых по строке покрыты) вычеркивают из таблицы.
4. В полученной в конечном итоге таблице могут остаться столбцы с несколькими метками. В таком случае выбирается такая совокупность первичных импликант, которая включает метки во всех столбцах (как минимум, по одной в каждом столбце). При нескольких возможных вариантах, предпочтение отдается варианту с минимальным суммарным числом переменных в простых импликантах, образующих покрытие. Данное покрытие можно осуществить, использую метод Петрика.

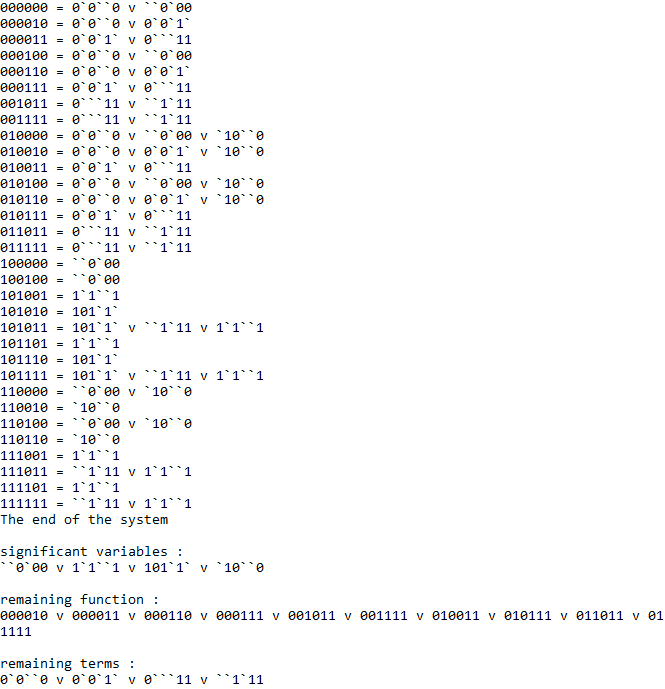
На основе вышеописанного алгоритма была написана программа на языке C++ (приложение А). На вход программе подаётся путь к файлу с расширением .txt, в котором записаны исходные данные ФАЛ в бинарном представлении (1 – переменная истинна, 0 – ложна, старший разряд в функции – старший бит), а также путь к .txt – файлу, в который запишется ответ (импликанты из МДНФ).

В процессе работы программы, в консоль выводятся (в этом же порядке, вывод в консоль изображен на рис. 1 и рис. 2):

1. Первичные импликанты каждого уровня;
2. Булевы функции исходной ФАЛ и соответствующие каждому значению функции первичные импликанты:
3. Существенные импликанты;
4. Значения ФАЛ, оставшиеся после вычеркивания лишних значений;
5. Импликанты, оставшиеся после вычеркивания существенных.

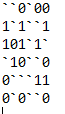


*Рис.1 (вывод консоли)*



*Рис.2 (вывод консоли, продолжение)*

После окончания работы программы, в выбранный нами файл будут записаны импликанты, соответствующие МДНФ (рис.3).



*Рис.3 (результат работы программы – импликанты, записанные в файл)*

Запишем полученную МДНФ:

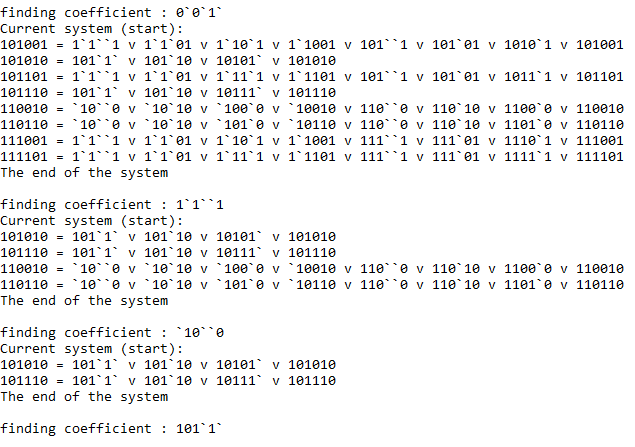
Таким образом, полученная МДНФ совпадает по сложности с решением методом карт Карно.

1. **Метод неопределённых коэффициентов**
2. Метод неопределённых коэффициентов заключается в построении системы уравнений, где в роли правой части уравнения выступает значение функции, соответствующее значениям переменных, а в роли левой – неопределённые коэффициенты ДНФ, индексы которых – значений исходных переменных.
3. Затем в полученной системе ищут уравнения с нулевой правой частью, и вычеркивают из всей системы уравнений коэффициенты, содержащиеся в этом уравнении. При этом само уравнение в дальнейшем также не рассматривается.
4. В конечном итоге в системе остаются уравнения, содержащие в правой части только единицу. В полученной системе последовательно выбирают коэффициенты с минимальным количеством индексов и максимальным числом вхождений в уравнения системы. Такой коэффициент вносят в МДНФ, а уравнения, в которых он содержится, вычеркиваются из системы. Данный пункт продолжают до тех пор, пока в системе не останется ни одного уравнения.

На основе вышеописанного алгоритма была написана программа на языке C++ (приложение Б). На вход программе подаётся путь к файлу с расширением .txt, в котором записаны исходные данные ФАЛ в бинарном представлении (1 – переменная истинна, 0 – ложна, старший разряд в функции – старший бит), а также путь к .txt – файлу, в который запишется ответ (импликанты из МДНФ).

В процессе работы программы, в консоль будет выводиться система (в качестве примера рис.4 – часть вывода из консоли). Каждый коэффициент системы будет выводиться комбинацией из n (где n – число переменных ФАЛ) символов, в которые входят: ‘~’ – переменная не включена, ‘0’ – включено отрицание переменной, ‘1’ – включена сама (например, коэффициент вида 101~1~ является коэффициентом, стоящим перед импликантой , то есть коэффициентом ).

Таким образом, сначала будет выведена система без нулевых значений функций и соответствующих им коэффициентов. Затем на каждом этапе будет находиться коэффициент, соответствующий импликанте с наименьшим числом переменных и покрывающий большее число уравнений, а после система будет упрощаться и выводиться в консоль. По окончанию работы программы в файл будут записаны коэффициенты соответствующих импликант МДНФ (рис. 5).

 *Рис.4 (частичный вывод консоли)*



*Рис.5 (результат работы программы – коэффициенты, записанные в файл)*

Запишем полученную МДНФ:

Таким образом, полученная МДНФ совпадает по сложности с решением методом карт Карно.

**Выводы:**

В результате выполнения работы были изучены три метода минимизации булевых функций: метод карт Карно, метод Квайна – МакКласски и метод неопределенных коэффициентов. Основными преимуществами методов Квайна – МакКласски и метода неопределенных коэффициентов является возможность их использования для большого числа переменных, в то время как с помощью карт Карно можно очень просто, компактно и наглядно минимизировать ФАЛ для небольшого числа переменных. Было установлено, что все три метода дали МДНФ одинаковой сложности, но разного вида, что говорит о не единственности получения МДНФ.

**Приложение А**

|  |  |
| --- | --- |
| #include <iostream>  #include <string>  #include <vector>  #include <fstream>  #include <sstream>  //преобразование данных из файла по заданному пути в vector  std::vector < std::vector < char >> input\_data(const std::string & path) {  std::vector < std::vector < char >> vector {};  std::ifstream fin(path);  if (fin.is\_open()) {  std::string time {};  while (std::getline(fin, time)) {  std::vector < char > time\_vec {};  time\_vec.push\_back('0');  for (unsigned int i = 0; i < time.size(); ++i) {  time\_vec.push\_back(time[i]);  }  vector.push\_back(time\_vec);  }  }  fin.close();  return vector;  }  void print\_function(const std::vector < std::vector < char >> & vector) {  for (unsigned int i = 0; i < vector.size(); ++i) {  for (unsigned int j = 1; j < vector[i].size(); ++j) {  std::cout << vector[i][j];  }  if (i < (vector.size() - 1)) std::cout << " v ";  }  std::cout << "\n\n";  }  void print\_system(const std::vector < std::vector < char >> & vector,  const std::vector < std::vector < char >> & coefs) {  std::cout << "Current system (start):" << std::endl;  for (unsigned int i = 0; i < vector.size(); ++i) {  for (unsigned int k = 1; k < vector[i].size(); ++k) {  std::cout << vector[i][k];  }  std::cout << " = ";  unsigned int count = 0;  for (unsigned j = 0; j < coefs.size(); ++j) {  bool flag = true;  for (unsigned int k = 1; k < coefs[j].size(); ++k) {  if (vector[i][k] != coefs[j][k] && coefs[j][k] != '`') {  flag = false;  break;  }  }  if (flag) {  if (count != 0) std::cout << " v ";  ++count;  for (unsigned int k = 1; k < coefs[j].size(); ++k) {  std::cout << coefs[j][k];  }  }  }  std::cout << std::endl;  }  std::cout << "The end of the system\n\n";  }  //проверка, включен ли included в vector (true - его нет в vector, false - иначе)  bool inclusion\_check(const std::vector < char > & included,  const std::vector < std::vector < char >> & vector) {  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag = true;  for (unsigned int j = 1; j < included.size(); ++j) {  if (vector[i][j] != included[j]) {  flag = false;  break;  }  }  if (flag) {  return false;  }  }  return true;  }  //преобразование vector с 'k' числом "тильд" к 'k+1'  std::vector < std::vector < char >> minimization(std::vector < std::vector < char >> & vector) {  std::vector < std::vector < char >> time\_vector {};  for (unsigned int i = 0; i < vector.size() - 1; ++i) {  for (unsigned int j = i + 1; j < vector.size(); ++j) {  unsigned int count = 0;  for (unsigned int k = 1; k < vector[i].size(); ++k) {  if (vector[i][k] != vector[j][k]) ++count;  }  if (count == 1) {  std::vector < char > time\_vec {};  time\_vec.push\_back('0');  vector[i][0] = '1';  vector[j][0] = '1';  for (unsigned int k = 1; k < vector[i].size(); ++k) {  if (vector[i][k] != vector[j][k]) time\_vec.push\_back('`');  else time\_vec.push\_back(vector[i][k]);  }  if (!time\_vector.empty()) {  if (inclusion\_check(time\_vec, time\_vector)) time\_vector.push\_back(time\_vec);  } else time\_vector.push\_back(time\_vec);  }  }  }  return time\_vector;  }  //вспомогательный метод для безызбыточного покрытия  std::vector < char > need\_implic(std::vector < std::vector < char >> & vector, std::vector < std::vector < char >> & value) {  unsigned int count = 0, num {};  for (unsigned int i = 0; i < value.size(); ++i) {  unsigned int time\_count = 0, time\_num = i;  for (unsigned int j = 0; j < vector.size(); ++j) {  bool flag = true;  for (unsigned int k = 1; k < vector[j].size(); ++k) {  if (value[i][k] != vector[j][k] && value[i][k] != '`') {  flag = false;  break;  }  }  if (flag) ++time\_count;  }  if (time\_count > count) {  count = time\_count;  num = time\_num;  }  }  std::vector < char > time = value[num];  std::vector < std::vector < char >> time\_vector {};  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag = true;  for (unsigned j = 1; j < value[num].size(); ++j) {  if (value[num][j] != vector[i][j] && value[num][j] != '`') {  flag = false;  break;  }  }  if (!flag) time\_vector.push\_back(vector[i]);  }  vector = time\_vector;  std::vector < std::vector < char >> time\_vector\_implic {};  for (unsigned int i = 0; i < value.size(); ++i) {  if (i != num) time\_vector\_implic.push\_back(value[i]);  }  value = time\_vector\_implic;  return time;  }  //поиск безызбыточного покрытия (существенных импликант нет)  std::vector < std::vector < char >> minimal\_vector\_implic(const std::vector < std::vector < char >> & vector,  const std::vector < std::vector < char >> & value) {  std::vector < std::vector < char >> time\_vector = vector;  std::vector < std::vector < char >> time\_value = value;  std::vector < std::vector < char >> final {};  while (!time\_vector.empty()) {  std::vector < char > time = need\_implic(time\_vector, time\_value);  final.push\_back(time);  }  return final;  }  //поиск безызбыточного покрытия (отделение существенных импликант). Вывод - вектор импликант (финальный)  std::vector < std::vector < char >> final\_minimization(const std::vector < std::vector < char >> & vector,  const std::vector < std::vector < char >> & value) {  std::vector < std::vector < char >> final {};  //поиск существенных импликант (записываются в final)  for (unsigned i = 0; i < vector.size(); ++i) {  unsigned int count = 0, num {};  for (unsigned int j = 0; j < value.size(); ++j) {  bool flag = true;  for (unsigned int k = 1; k < vector[i].size(); ++k) {  if (vector[i][k] != value[j][k] && value[j][k] != '`') {  flag = false;  break;  }  }  if (flag) {  ++count;  num = j;  }  }  if (count == 1) {  if (!final.empty()) {  bool flag = inclusion\_check(value[num], final);  if (flag) final.push\_back(value[num]);  } else final.push\_back(value[num]);  }  }  std::cout << "significant variables : " << std::endl;  print\_function(final);  std::vector < std::vector < char >> changed\_func {}; //вектор оставшися значений функции, оставшихся после нахождения и учитывания существенных  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag\_ = true;  for (unsigned int j = 0; j < final.size(); ++j) {  bool flag = true;  for (unsigned int k = 1; k < final[j].size(); ++k) {  if (vector[i][k] != final[j][k] && final[j][k] != '`') {  flag = false;  break;  }  }  if (flag) {  flag\_ = false;  break;  }  }  if (flag\_) changed\_func.push\_back(vector[i]);  }  std::cout << "remaining function : " << std::endl;  print\_function(changed\_func);  std::vector < std::vector < char >> final\_ {}; //вектор оставшихся импликант, после нахождения существенных  for (unsigned int i = 0; i < value.size(); ++i) {  bool flag = inclusion\_check(value[i], final);  if (flag) final\_.push\_back(value[i]);  }  std::cout << "remaining terms : " << std::endl;  print\_function(final\_);  //финальный вектор импликант (ответ)  std::vector < std::vector < char >> final\_\_ = minimal\_vector\_implic(changed\_func, final\_);  for (unsigned int i = 0; i < final\_\_.size(); ++i) {  final.push\_back(final\_\_[i]);  }  return final;  }  std::string main\_algorithm(std::vector < std::vector < char >> & vector) {  unsigned int count = 0;  std::cout << count << " floor : " << std::endl;  print\_function(vector);  std::ostringstream exit {};  std::vector < std::vector < char >> start\_function = vector;  std::vector < std::vector < char >> value {};  //переход векторов с 'k' числом "тильд" к 'k+1' и их обработка  while (true) {  std::vector < std::vector < char >> time\_vector = minimization(vector);  ++count;  //проверка, включен ли данный набор в последующие преобразование, если нет, то он включается в конечную ФАЛ  for (unsigned int i = 0; i < vector.size(); ++i) {  if (vector[i][0] == '0') {  if (!value.empty()) {  if (inclusion\_check(vector[i], value)) value.push\_back(vector[i]);  } else value.push\_back(vector[i]);  }  }  vector = time\_vector;  if (vector.empty()) break;  else {  std::cout << count << " floor : " << std::endl;  print\_function(vector);  }  }  std::cout << "received terms : " << std::endl;  print\_function(value);  print\_system(start\_function, value);  value = final\_minimization(start\_function, value);  for (unsigned int i = 0; i < value.size(); ++i) {  for (unsigned int j = 1; j < value[i].size(); ++j) {  exit << value[i][j];  }  exit << std::endl;  }  return exit.str();  }  int main() {  std::string path;  std::cout << "Input the path to the file" << std::endl;  std::getline(std::cin, path);  std::vector < std::vector < char >> vector = input\_data(path);  std::string path\_;  std::cout << "Input the path to save data" << std::endl;  std::getline(std::cin, path\_);  std::ofstream fout(path\_);  if (fout.is\_open()) {  fout << main\_algorithm(vector);  fout.close();  }  return 0;  } |  |

**Приложение Б**

|  |  |
| --- | --- |
| #include <iostream>  #include <vector>  #include <string>  #include <fstream>  #include <sstream>  //преобразование данных из файла по заданному пути в vector (значения булевой функции)  std::vector < std::vector < char >> input\_data(const std::string & path) {  std::vector < std::vector < char >> vector {};  std::ifstream fin(path);  if (fin.is\_open()) {  std::string time {};  while (std::getline(fin, time)) {  std::vector < char > time\_vec {};  for (unsigned int i = 0; i < time.size(); ++i) {  time\_vec.push\_back(time[i]);  }  vector.push\_back(time\_vec);  }  }  fin.close();  return vector;  }  //вывод системы в консоль  void print\_system(const std::vector < std::vector < char >> & vector,  const std::vector < std::vector < char >> & coefs) {  std::cout << "Current system (start):" << std::endl;  for (unsigned int i = 0; i < vector.size(); ++i) {  for (unsigned int k = 0; k < vector[i].size(); ++k) {  std::cout << vector[i][k];  }  std::cout << " = ";  unsigned int count = 0;  for (unsigned j = 0; j < coefs.size(); ++j) {  bool flag = true;  for (unsigned int k = 0; k < coefs[j].size(); ++k) {  if (vector[i][k] != coefs[j][k] && coefs[j][k] != '`') {  flag = false;  break;  }  }  if (flag) {  if (count != 0) std::cout << " v ";  ++count;  for (unsigned int k = 0; k < coefs[j].size(); ++k) {  std::cout << coefs[j][k];  }  }  }  std::cout << std::endl;  }  std::cout << "The end of the system\n\n";  }  //создание всех коэффициентов, нужных для системы  std::vector < std::vector < char >> generation\_all\_coefficients(const unsigned int & n) {  std::vector < std::vector < char >> final {};  std::vector < char > time {};  for (unsigned int i = 0; i < n; ++i) {  time.push\_back('`');  }  while (true) {  bool flag = true;  for (int i = n - 1; i >= 0; --i) {  if (time[i] == '`') {  time[i] = '0';  flag = false;  break;  } else if (time[i] == '0') {  time[i] = '1';  flag = false;  break;  } else if (time[i] == '1') {  time[i] = '`';  }  }  if (flag) break;  else final.push\_back(time);  }  return final;  }  //проверка, включен ли included в vector (true - его нет в vector, false - иначе)  bool inclusion\_check(const std::vector < char > & included,  const std::vector < std::vector < char >> & vector) {  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag = true;  for (unsigned int j = 0; j < included.size(); ++j) {  if (vector[i][j] != included[j]) {  flag = false;  break;  }  }  if (flag) {  return false;  }  }  return true;  }  //создание всех нулевых булевых функций  std::vector < std::vector < char >> generation\_zero\_vector(const std::vector < std::vector < char >> & vector) {  std::vector < std::vector < char >> final {};  std::vector < char > time {};  for (unsigned int i = 0; i < vector[0].size(); ++i) {  time.push\_back('0');  }  if (inclusion\_check(time, vector)) final.push\_back(time);  while (true) {  bool flag = true;  for (int i = vector[0].size() - 1; i >= 0; --i) {  if (time[i] == '0') {  time[i] = '1';  flag = false;  break;  } else if (time[i] == '1') {  time[i] = '0';  }  }  if (flag) break;  else {  if (inclusion\_check(time, vector)) final.push\_back(time);  }  }  return final;  }  //проверка, включен ли коэффициент included в vector - значения булевой функции (true - его нет в vector, false - иначе)  bool inclusion\_check\_coef(const std::vector < char > & included,  const std::vector < std::vector < char >> & vector) {  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag = true;  for (unsigned int j = 0; j < included.size(); ++j) {  if (vector[i][j] != included[j] && included[j] != '`') {  flag = false;  break;  }  }  if (flag) return false;  }  return true;  }  //удалени коэффициентов, соответствующим нулевым булевым функциям  std::vector < std::vector < char >> delete\_zero\_coefs(const std::vector < std::vector < char >> & coefs,  const std::vector < std::vector < char >> & vector) {  std::vector < std::vector < char >> time {};  for (unsigned int i = 0; i < coefs.size(); ++i) {  if (inclusion\_check\_coef(coefs[i], vector)) time.push\_back(coefs[i]);  }  return time;  }  //нахождение коэффициента, покрывающего больше всего булевых функций и редактирование системы  std::vector < char > take\_coef(std::vector < std::vector < char >> & vector, std::vector < std::vector < char >> coefs) {  unsigned int count = 0, num {};  for (unsigned int i = 0; i < coefs.size(); ++i) {  unsigned int time\_count = 0, time\_num = i;  for (unsigned int j = 0; j < vector.size(); ++j) {  bool flag = true;  for (unsigned int k = 0; k < vector[j].size(); ++k) {  if (vector[j][k] != coefs[i][k] && coefs[i][k] != '`') {  flag = false;  break;  }  }  if (flag) ++time\_count;  }  if (time\_count > count) {  count = time\_count;  num = time\_num;  }  }  std::vector < char > exit = coefs[num];  std::vector < std::vector < char >> time\_vector {}, time\_coefs {};  for (unsigned int i = 0; i < vector.size(); ++i) {  bool flag = true;  for (unsigned j = 0; j < exit.size(); ++j) {  if (exit[j] != vector[i][j] && exit[j] != '`') {  flag = false;  break;  }  }  if (!flag) time\_vector.push\_back(vector[i]);  }  for (unsigned int i = 0; i < coefs.size(); ++i) {  if (!inclusion\_check\_coef(coefs[i], time\_vector)) time\_coefs.push\_back(coefs[i]);  }  vector = time\_vector;  coefs = time\_coefs;  return exit;  }  //поиск коэффициентов в упрощенной системе (без нулевых значений функции и коэффициентов, соответствующих этим значением)  std::vector < std::vector < char >> search\_coefs(const std::vector < std::vector < char >> & vector,  const std::vector < std::vector < char >> & coefs) {  std::vector < std::vector < char >> final {}, vec = vector, coef = coefs;  while (!vec.empty()) {  std::vector < char > time = take\_coef(vec, coef);  final.push\_back(time);  std::cout << "finding coefficient : ";  for (unsigned int i = 0; i < time.size(); ++i) std::cout << time[i];  std::cout << std::endl;  if (!vec.empty()) print\_system(vec, coef);  }  return final;  }  std::string main\_algorithm(const std::vector < std::vector < char >> & vector,  const unsigned int & n) {  std::vector < std::vector < char >> coefs = generation\_all\_coefficients(n);  std::vector < std::vector < char >> zero\_vector = generation\_zero\_vector(vector);  std::vector < std::vector < char >> coefs\_upgrade = delete\_zero\_coefs(coefs, zero\_vector);  print\_system(vector, coefs\_upgrade);  std::vector < std::vector < char >> final = search\_coefs(vector, coefs\_upgrade);  std::ostringstream exit {};  for (unsigned int i = 0; i < final.size(); ++i) {  for (unsigned int j = 0; j < final[i].size(); ++j) {  exit << final[i][j];  }  exit << std::endl;  }  return exit.str();  }  int main() {  unsigned int n = 6;  std::string path;  std::cout << "Input the path to the file" << std::endl;  std::getline(std::cin, path);  std::vector < std::vector < char >> function = input\_data(path);  std::string path\_;  std::cout << "Input the path to save data" << std::endl;  std::getline(std::cin, path\_);  std::ofstream fout(path\_);  if (fout.is\_open()) {  fout << main\_algorithm(function, n);  fout.close();  }  return 0;  } |  |