МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Лабораторная работа №2.2
по дисциплине: «Дискретная математика»
по теме: «**Задачи выбора**»

Выполнил/а: ст. группы ПВ-231
Чупахина София Александровна
Проверили:
Островский Алексей Мичеславович
Рязанов Юрий Дмитриевич

Белгород, 2024

Цель работы: приобрести практические навыки в использовании алгоритмов порождения комбинаторных объектов при проектировании алгоритмов решения задач выбора.

Вариант 6:

Задание 1:

Задача для варианта 6: Задана матрица Квайна. Вывести все подмножества простых импликант, объединение которых даёт тупиковую нормальную форму Кантора.

Задание 2:

Текст задания: определить класс комбинаторных объектов, содержащий решение задачи (траекторию задачи).
Предположим, что данные импликанты обозначены числами от 1 до n; само множество импликант обозначим как I. предположим также, что номера конституент, входящих в СНФК данного выражения, обозначены номерами от одного до m; множество конституент, входящих в совершенную форму данного выражения, обозначим C. Множество конституент, которые покрывает некая простая импликанта с номером i, обозначим C_i. Классом комбинаторных объектов, содержащих решения задачи, будут подмножества множества I (некое подмножество множества I далее будем обозначать как IS).

Задание 3:

Текст задания: определить, что в задаче является функционалом и способ его вычисления.
За функционал в этой задаче примем массив из m элементов, обозначающий, сколько раз каждая из конституент покрывается некоторым набором импликант IS. Будем обозначать его как CS.

Задание 4:

Текст задания: определить способ распознавания решения по значению функционала.
Подмножество множества импликант будет являться решением тогда, когда импликанты множества IS полностью покрывают все множество конституент C, то есть в массиве CS не ни одного нуля; и при этом для каждой импликанты i можно найти конституенту, покрытую только ей, то есть среди позиций, которые она покрывает в массиве CS, есть хотя бы одна единица.

Задание 5:

Текст задания: реализовать алгоритм решения задачи.
Для того, чтобы реализовать этот алгоритм, потребуется ряд функций для работы с множествами на упорядоченных массивах и с матрицами. Для работы с множествами подключим к основному файлу уже знакомую нам библиотеку ordered_array_set с заголовочным файлом **ordered_array_set.h:**

```
#ifndef INC_ORDERED_ARRAY_SET_H
#define INC_ORDERED_ARRAY_SET_H

#include <stdint.h>
```

```c
#include <assert.h>
#include <memory.h>
#include <malloc.h>
#include <stdio.h>
#include <stdbool.h>


typedef struct ordered_array_set {
    int *data; //элементы множества
    size_t size; //количество элементов в множестве
    size_t capacity; //максимальное количество элементов в
множестве
} ordered_array_set;

//возвращает пустое множество, в которое можно вставить capacity
элементов
ordered_array_set ordered_array_set_create (size_t capacity);

//возвращает множество, состоящее из элементов массива a размера
size
ordered_array_set ordered_array_set_create_from_array (const int
*a, size_t size);

//Уменьшает емкость capacity множества по адресу a до его размера
size
//При этом перераспределяет память, отведенную под массив value,
чтобы он занимал минимальное ее количество
static void ordered_array_set_shrinkToFit (ordered_array_set *a);

//Увеличивает емкость capacity множества по адресу a на slots
единиц
static void ordered_array_set_increaseCapacity (ordered_array_set
*a, size_t slots);

//возвращает значение позицию элемента в множестве,
//если значение value имеется в множестве set, иначе - n
size_t ordered_array_set_in (ordered_array_set *set, int value);

//возвращает значение 'истина', если элементы множеств set1 и set2
равны, иначе - 'ложь'
bool ordered_array_set_isEqual (ordered_array_set set1,
ordered_array_set set2);

//возвращает значение 'истина', если subset является подмножеством
set, иначе - 'ложь'
bool ordered_array_set_isSubset (ordered_array_set subset,
ordered_array_set set);
```

```
//возбуждает исключение, если в множество по адресу set нельзя
вставить элемент
void ordered_array_set_isAbleAppend (ordered_array_set *set);

//добавляет элемент value в множество set
void ordered_array_set_insert (ordered_array_set *set, int value);

//удаляет элемент value из множества set
void ordered_array_set_deleteElement (ordered_array_set *set, int
value);

//возвращает объединение множеств set1 и set2
ordered_array_set join (ordered_array_set set1, ordered_array_set
set2);

//возвращает пересечение множеств set1 и set2
ordered_array_set intersection (ordered_array_set set1,
ordered_array_set set2);

//возвращает разность множеств set1 и set2
ordered_array_set difference (ordered_array_set set1,
ordered_array_set set2);

//возвращает симметрическую разность множеств set1 и set2
ordered_array_set symmetricDifference (ordered_array_set set1,
ordered_array_set set2);

//возвращает дополнение до универсума universumSet множества set
ordered_array_set complement (ordered_array_set set,
ordered_array_set universumSet);

//вывод множества set
void ordered_array_set_print (ordered_array_set set);

//освобождает память, занимаемую множеством set
void ordered_array_set_delete (ordered_array_set *set);
#endif
```

...И файлом реализации **ordered_array_set.c:**

```
#ifndef INC_ORDERED_ARRAY_SET_C
#define INC_ORDERED_ARRAY_SET_C

#include "ordered_array_set.h"
#include "array\array.c"
#include "C:\Users\sovac\Desktop\ОП, преимущественно
лабы\second_semester\libs\algorithms\math_basics\math_basics.c"
```

```
ordered_array_set ordered_array_set_create (size_t capacity) {
   return (ordered_array_set) {
       malloc(sizeof(int) * capacity),
       0,
       capacity
   };
}

ordered_array_set ordered_array_set_create_from_array (const int
*a, size_t size) {
    ordered_array_set result = ordered_array_set_create(size);
    for (size_t i = 0; i < size; i++) {
        ordered_array_set_insert(&result, a[i]);
    }
    qsort(result.data, size, sizeof(int), compare_ints);
    ordered_array_set_shrinkToFit(&result);
    return result;
}

static void ordered_array_set_shrinkToFit (ordered_array_set *a) {
    if (a->size != a->capacity) {
        a->data = (int *) realloc(a->data, sizeof(int) * a->size);
        a->capacity = a->size;
    }
}

static void ordered_array_set_increaseCapacity (ordered_array_set
*a, size_t slots) {
    a->data = (int *) realloc(a->data, sizeof(int) * (a->size +
slots));
    a->capacity += slots;
}

size_t ordered_array_set_in (ordered_array_set *set, int value) {
    size_t index = binarySearch_(set->data, set->size,value);
    return (index != SIZE_MAX) ? index : set->size;
}

bool ordered_array_set_isEqual (ordered_array_set set1,
ordered_array_set set2) {
    return memcmp(set1.data, set2.data, sizeof(int) * set1.size)
== 0;
}

bool ordered_array_set_isSubset (ordered_array_set subset,
```

```
ordered_array_set set) {
    bool is_subset = 1;
    for (size_t i = 0; i < subset.size; i++) {
        if (ordered_array_set_in(&set, subset.data[i]) ==
set.size) {
            is_subset = 0;
            break;
        }
    }
    return is_subset;
}

void ordered_array_set_isAbleAppend (ordered_array_set *set) {
    assert(set->size < set->capacity);
}

void ordered_array_set_insert (ordered_array_set *set, int value)
{
    ordered_array_set_isAbleAppend(set);
    if (set->size == 0 || value > set->data[(set->size)-1]) {
        append_(set->data, &(set->size), value);
    } else if (ordered_array_set_in(set, value) == set->size) {
        size_t start_index = binarySearchMoreOrEqual_(set->data,
set->size, value);
        insert_(set->data, &(set->size), start_index, value);
    }
}

void ordered_array_set_deleteElement (ordered_array_set *set, int
value) {
    if (ordered_array_set_in(set, value) != set-> size) {
        deleteByPosSaveOrder_(set->data, &(set->size),
ordered_array_set_in(set, value));
    }
}

ordered_array_set join (ordered_array_set set1, ordered_array_set
set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity + set2.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
```

```
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
        } else {
            ordered_array_set_insert(&result, set2.data[index2]);
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    while (index2 < set2.size) {
        ordered_array_set_insert(&result, set2.data[index2]);
        index2++;
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set intersection (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            index1++;
        } else {
            index2++;
        }
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set difference (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity);
```

```
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
        } else {
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set symmetricDifference (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity + set2.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
        } else {
            ordered_array_set_insert(&result, set2.data[index2]);
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    while (index2 < set2.size) {
        ordered_array_set_insert(&result, set2.data[index2]);
        index2++;
```

```
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set complement (ordered_array_set set,
ordered_array_set universumSet) {
    return difference(universumSet, set);
}

void ordered_array_set_print (ordered_array_set set) {
    if (set.size == 0) {
        printf("Empty set\n");
    } else {
        printf("{");
        for (int i = 0; i < set.size; i++) {
            printf("%d ", set.data[i]);
        }
        printf("\b}\n");
    }
}

void ordered_array_set_delete (ordered_array_set *set) {
    free(set->data);
    set->size = 0;
    set->capacity = 0;
}

#endif
```

Для работы с матрицами нам понадобится несколько функций из библиотеки matrix, которую мы также писали в рамках курса ОП. Выделим их в отдельные файлы. заголовочный файл **matrix.h :**

```
#ifndef INC_MATRIX_H
#define INC_MATRIX_H

#include <stdbool.h>
#include <stdint.h>

typedef struct {
    int **values; // элементы матрицы
    int nRows; // количество рядов
    int nCols; // количество столбцов
} matrix;

//размещает в динамической памяти и возвращает матрицу размером
nRows на nCols
```

```
matrix getMemMatrix(int nRows, int nCols);

//возвращает матрицу размера nRows на nCols, построенную из
элементов массива a
matrix createMatrixFromArray(const int *a, size_t nRows, size_t
nCols);

#endif
```

...И файл реализации **matrix.c:**

```
#ifndef INC_MATRIX_C
#define INC_MATRIX_C
#include "matrix.h"
#include <malloc.h>

matrix getMemMatrix(int nRows, int nCols) {
    int **values = (int **) malloc(sizeof(int*) * nRows);
    for (int row_ind = 0; row_ind < nRows; row_ind++) {
        values[row_ind] = (int *) malloc(sizeof(int) * nCols);
    }
    return (matrix){values, nRows, nCols};
}

//возвращает матрицу размера nRows на nCols, построенную из
элементов массива a
matrix createMatrixFromArray(const int *a, size_t nRows, size_t
nCols) {
    matrix m = getMemMatrix(nRows, nCols);
    int k = 0;
    for (int row_ind = 0; row_ind < nRows; row_ind++) {
        for (int col_ind = 0; col_ind < nCols; col_ind++) {
            m.values[row_ind][col_ind] = a[k++];
        }
    }
    return m;
}

#endif
```

Наконец, приступим к непосредственно решению задачи. Помимо стандартных функций для генерации подмножеств методом поиска с возвращением, нам понадобятся функции вычисления функционала и определения, является ли подмножество решением, по этому функционалу.

Функция вычисления функционала будет иметь следующий вид. Ей передается адрес, в который надо записать данные массива CS, подмножество множества импликант IS и матрица Квайна quine, на основании которой функционал будет

высчитываться. Сначала все элементы массива CS (их будет столько же, сколько конституент - и сколько столбцов в матрице quine) инициализируются нулями, потом перебираются все импликанты из массива IS, и для каждой импликанты перебираются все конституенты. Если они покрываются выбранной импликантой, соответствующие им позиции массива CS увеличиваются на 1.

Стоит отметить, что в множестве I нумерация импликант идет с единицы, в то время как строки матрицы нумеруются с нуля. Поэтому, используя элемент множества как индекс, его необходимо уменьшить на 1

```
//Возвращает функционал - массив CS, показывающий, сколько раз
определенный столбец покрывается подмножеством множества
конституент IS
void getCS(int *CS, ordered_array_set IS, matrix quine) {
    for (int i = 0; i < quine.nCols; i++) {
        CS[i] = 0;
    }
    //Перебираются все импликанты из массива IS
    for (int cur_i_ind = 0; cur_i_ind < IS.size; cur_i_ind++) {
        for (int cur_const_ind = 0; cur_const_ind < quine.nCols;
cur_const_ind++) {
            if (quine.values[IS.data[cur_i_ind]-1][cur_const_ind])
{
                CS[cur_const_ind] += 1;
            }
        }
    }
}
```

Функция определения по функционалу, является ли подмножество решением, будет иметь следующий вид. Она может возвращать «ложь» в двух случаях: если покрытие матрицы строками не полное или если оно излишнее, то есть найдется строка, которую можно исключить. Первый случай обрабатывается условным оператором в самом начале: если в массиве CS есть хотя бы один ноль, значит, не все столбцы покрыты данным множеством импликант IS; возвращается 0. Если это не так, нужно проверить, у каждой ли импликанты есть хотя бы одна конституента, покрываемая только ей, т. е. есть ли на позициях, за которые она отвечает, хотя бы одна единица. Перебираются все импликанты из массива IS, и для каждой импликанты перебираются все конституенты. Если они покрываются выбранной импликантой и соответствующие им позиции массива CS равны 1, вхождение строки в тупиковую форму обязательно. Если же нашлась хотя бы одна необязательная импликанта, подмножество IS нельзя считать тупиковой формой.

```
//Возвращает 1, если функционал CS указывает на то, что
объединение подмножества множества импликант IS является тупиковой
формой, и 0 в противном случае
bool isISASolution (int *CS, ordered_array_set IS, matrix quine) {
```

```
    if (linearSearch_(CS, quine.nCols, 0) != quine.nCols) {
        return false;
    }
    bool is_dead_end_form = true;
    for (int cur_i_ind = 0; cur_i_ind < IS.size; cur_i_ind++) {
        bool isINecessary = false;
        for (int cur_const_ind = 0; cur_const_ind < quine.nCols;
cur_const_ind++) {
            if (quine.values[IS.data[cur_i_ind]-1][cur_const_ind]
&& CS[cur_const_ind] == 1) {
                isINecessary = true;
                break;
            }
        }
        if (!isINecessary) {
            is_dead_end_form = false;
            break;
        }
    }
    return is_dead_end_form;
}
```

И наконец, дополним код стандартными функциями поиска с возвращением, с небольшими дополнениями. В частности, функция createSubsetByBinaryVector не претерпела никаких изменений. А вот функция генерации подмножеств, которая здесь называется generateAllDeadEndForms_, теперь принимает дополнительный аргумент, матрицу Квайна quine, которая необходима для вычисления функционала CS. Если полученный вектор достиг нужной длины, генерируется IS, подмножество исходного множества I, но оно не выводится на экран сразу; на его основе создается функционал CS. Если функционал указывает на то, что IS – решение, то IS выводится на экран.

```
//Возвращает подмножество множества set, заданное двоичным
вектором vector
ordered_array_set createSubsetByBinaryVector(ordered_array_set
set, int vector) {
    assert (1<<set.size > vector);
    ordered_array_set result =
ordered_array_set_create(set.capacity);
    for (int i = 0; i < set.capacity; i++) {
        if (vector >> i & 1) {
            ordered_array_set_insert(&result, set.data[i]);
        }
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}
```

```
//Выводит на экран все номера импликант, объединение которых даст
тупиковую форму Кантора
//Рекурсивная функция, опирающаяся на каждой итерации на текущий
двоичный вектор, задающий подмножество, номер формируемого
разряда, множество номеров импликант I и матрицу Квайна quine
void generateAllDeadEndForms_(int cur_vector, int cur_digit,
ordered_array_set I, matrix quine) {
    for (int i = 0; i <= 1; i++) {
        int plus_cur_vector = (cur_vector << 1) + i;
        if (cur_digit == I.size) {
            ordered_array_set cur_IS =
createSubsetByBinaryVector(I, plus_cur_vector);
            int cur_CS[quine.nCols];
            getCS(cur_CS, cur_IS, quine);
            if (isISASolution(cur_CS, cur_IS, quine)) {
                ordered_array_set_print(cur_IS);
            }
        } else {
            generateAllDeadEndForms_(plus_cur_vector, cur_digit+1,
I, quine);
        }
    }
}
```

И наконец, функция-обёртка generateAllDeadEndForms не только вызывает функцию generateAllDeadEndForms_ с нулевым вектором и номером разряда 1, но и генерирует множество I, на основе которого и будут генерироваться подмножества, а также обрабатывает сценарий. когда найти решение не представляется возможным.

```
//Выводит на экран все номера импликант, объединение которых даст
тупиковую форму Кантора
//Обертка функции generateAllDeadEndForms_, начинающая работу с
пустым стартовым вектором cur_vector, номером обрабатываемого
разряда 1
//и множеством импликант I, сгенерированным на основе переданной
матрицы Квайна
void generateAllDeadEndForms(matrix quine) {
    if (quine.nCols == 0 || quine.nRows == 0) {
        printf("It's not a quine matrix.");
    } else {
        ordered_array_set I =
ordered_array_set_create(quine.nRows);
        for (int i = 1; i <= quine.nRows; i++) {
            ordered_array_set_insert(&I, i);
        }
```

```
        generateAllDeadEndForms_(0, 1, I, quine);
    }
}
```

Задание 6:

Текст задания: подготовить тестовые данные и решить задачу.

Для тестирования получившегося кода используем четыре матрицы Квайна. Первая является произвольной (не отражает никакую реальную функцию), но все равно пригодна для тестирования:

	1	2	3	4	5
1	1	0	1	1	1
2	0	0	0	1	1
3	1	1	1	1	0

Легко заметить, что ядро этой матрицы – строка 3 (только она способна покрыть второй столбец), и что единственный столбец, не покрытый ядром – пятый, и его могут покрывать либо первая, либо вторая строка. Значит, множество решений содержит 2 элемента: {{1, 3}, {2, 3}}.

Вторую матрицу для тестирования мы возьмем из методических указаний (лекции 3):

	1	2	3	4	5	6	7	8
1	1	0	0	1	0	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	1	1	0	0	0	0
4	0	0	0	0	1	1	0	0
5	0	0	0	0	1	0	0	1
6	0	1	1	0	0	0	1	1

В этой матрице ядром будет являться строка 6 (только она покрывает второй и седьмой столбцы), непокрытыми ей остаются первый, четвертый, пятый и шестой столбцы. Воспользовавшись методом Петрика, получим следующее выражение – конъюнктивное представление матрицы: (1 ИЛИ 2) И (1 ИЛИ 3) И (4 ИЛИ 5) И (2 ИЛИ 4), а после раскрытия скобок получим 1 И 4 ИЛИ 1 И 2 И 5 ИЛИ 2 И 3 И 4 ИЛИ 2 И 3 И 5. Прибавив к каждой из конъюнкций ядро матрицы, получаем множество решений: {{1, 4, 6}, {1, 2, 5, 6}, {2, 3, 4, 6}, {2, 3, 5, 6}}.

Третью матрицу мы возьмем из лабораторной работы по теме «нормальные формы Кантора» (вариант 9):

	1	2	3	4	5
1	1	1	0	0	0
2	0	0	1	1	0

3	0	1	0	0	1
4	0	0	0	1	1

В этой матрице ядром будут являться строки 1 и 2 (только первая строка покрывает первый столбец и только вторая – третий столбец), непокрытыми ядром остается лишь пятый столбец, который могут покрывать строки 3 и 4. Значит, множество решений имеет следующий вид: {{1, 2, 3}, {1, 2, 4}}.

Наконец, снова воспользуемся произвольной матрицей для рассмотрения случая, когда тупиковая форма у функции всего лишь одна:

	1	2	3	4	5	6
1	1	1	0	1	0	0
2	0	0	1	0	0	1
3	0	1	1	0	1	0

Здесь все строки являются ядром матрицы, и множество решений будет выглядеть как {{1, 2, 3}}.

В программном виде все эти матрицы и вывод решений на экран будут заданы следующим образом:

```
int main () {
    matrix quine_arbitrary = createMatrixFromArray(
            (int[]) {
                    1, 0, 1, 1, 1,
                    0, 0, 0, 1, 1,
                    1, 1, 1, 1, 0
            }, 3, 5
    );
    matrix quine_from_manual = createMatrixFromArray(
            (int[]) {
                    1, 0, 0, 1, 0, 0, 0, 0,
                    1, 0, 0, 0, 0, 1, 0, 0,
                    0, 0, 1, 1, 0, 0, 0, 0,
                    0, 0, 0, 0,1, 1, 0, 0,
                    0, 0, 0, 0, 1, 0, 0, 1,
                    0, 1, 1, 0, 0, 0, 1, 1

            }, 6, 8
    );

    matrix quine_from_lab_1_2 = createMatrixFromArray(
            (int[]) {
                    1, 1, 0, 0, 0,
                    0, 0, 1, 1, 0,
                    0, 1, 0, 0, 1,
                    0, 0, 0, 1, 1
```

```
            }, 4, 5
    );

    matrix quine_arbitrary_one_solution = createMatrixFromArray(
            (int[]) {
                    1, 1, 0, 1, 0, 0,
                    0, 0, 1, 0, 0, 1,
                    0, 1, 1, 0, 1, 0
            }, 3, 6
    );

    printf("Solutions of arbitrary quine matrix:\n");
    generateAllDeadEndForms(quine_arbitrary);
    printf("\nSolutions of quine matrix from manual:\n");
    generateAllDeadEndForms(quine_from_manual);
    printf("\nSolutions of quine matrix from lab 2.1, variant
9:\n");
    generateAllDeadEndForms(quine_from_lab_1_2);
    printf("\nSolutions of arbitrary quine matrix with one
solution:\n");
    generateAllDeadEndForms(quine_arbitrary_one_solution);
}
```

Вывод на экран будет совпадать с решениями, указанными выше, что говорит нам о правильности алгоритма.

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 2.2\dead_end_forms_search — копия.exe"
Solutions of arbitrary quine matrix:
{1 3}
{2 3}

Solutions of quine matrix from manual:
{1 4 6}
{2 3 4 6}
{1 2 5 6}
{2 3 5 6}

Solutions of quine matrix from lab 2.1, variant 9:
{1 2 3}
{1 2 4}

Solutions of arbitrary quine matrix with one solution:
{1 2 3}

Process finished with exit code 0
```

Вывод:

Если задача имеет конечное множество решений, представимых как комбинаторные объекты, то такая задача называется задачей выбора. Для их решения, как и для порождения комбинаторных объектов, подходит метод поиска с возвращением. Однако для задач выбора необходимо не просто породить все возможные комбинаторные объекты, но и сформулировать функционал – числовую или иную характеристику, указывающую на то, является ли объект решением – и найти его для каждого полученного объекта. Решением будут являться только те объекты, для которых функционал равен определенному значению или удовлетворяет некоему условию.