

**КУРСОВАЯ РАБОТА**  
**«Использование бинарных решающих диаграмм**  
**для решения логических задач. Библиотека BuDDy»**  
**по дисциплине «Математическая логика»**

Выполнила студентка  
группы 3530202/80202:

Козлова Е.А.

Преподаватель:

Шошмина И.В.

## Постановка задачи

Необходимо решить следующую задачу (12 вариант):

- Пусть имеется  $N=9$  объектов. Расположены объекты следующим образом:

|  |  |  |
|--|--|--|
|  |  |  |
|  |  |  |
|  |  |  |

- “Соседские” отношения между объектами определены в индивидуальном варианте относительно центрального объекта:

|   |   |  |
|---|---|--|
| * |   |  |
|   | 0 |  |
| * |   |  |

- Необходимо выбрать  $M=4$  свойств, принимающих  $N$  различных значений.
  - Задать  $n_1 = 3$  ограничений 1-го типа. ( $n_1 = 3$ )
  - Задать  $n_2 = 5$  ограничений 2-го типа. ( $n_2 = 5$ )
  - Для ограничений типа 3 и типа 4 использовать “соседские” отношения. Придумать, как ограничения, подобные типу 3, 4, выражаются в соседских отношениях, задать  $n_3 = 4$  ограничений подобных отношениям типа 3 и  $n_4 = 4$  ограничений подобных отношениям типа 4. ( $n_3 = 4$ ,  $n_4 = 4$ )
- Необходимо описать дополнительный тип ограничения  $n_7$ . Пусть сумма свойств объектов-соседей не должна быть больше  $K$ , где  $K$  – некоторое число от 0 до  $N \cdot M$ .  $K$  выбирается студентом.
- Необходимо найти все возможные решения и придумать физическую интерпретацию.
- Если задача имеет не одно решение, следует добавить и/или изменить некоторые ограничения так, чтобы задача имела только одно единственное решение.
- Если задача не имеет решений, следует удалить и/или изменить некоторые ограничения так, чтобы задача имела только одно единственное решение.
- Ограничение типа  $n_7$  удалять нельзя.

## Описание решения

Поскольку весь двумерный массив при программировании рассматривается как одномерный, то «\*», оказавшиеся слева от нуля будут считаться левыми «соседями», а те, что справа – правыми.

|                |  |  |  |   |                 |   |  |  |
|----------------|--|--|--|---|-----------------|---|--|--|
| *              |  |  |  | 0 |                 | * |  |  |
| Левые «соседи» |  |  |  |   | Правые «соседи» |   |  |  |

Отсюда получаем новые маски для отношений типа «А слева от Б» и «А справа от Б» о ограничения на расположения меток «А» и «Б»:

|                 |   |  |  |  |  |   |  |   |  |  |   |
|-----------------|---|--|--|--|--|---|--|---|--|--|---|
| «А слева от Б»  | <table><tr><td>А</td><td></td><td></td></tr><tr><td></td><td>Б</td><td></td></tr><tr><td></td><td></td><td></td></tr></table> | А  |  |  |  | Б |  |   |  |  | «Б» не может находиться в первой строке и первом столбце    |
|                 |   | А  |  |  |  |   |  |   |  |  |   |
|                 | Б   |  |  |  |  |   |  |   |  |  |   |
|                 |   |  |  |  |  |   |  |   |  |  |   |
|                 |   | «А» не может находиться в последнем столбце и последней строке |  |  |  |   |  |   |  |  |   |
| «А справа от Б» | <table><tr><td></td><td></td><td></td></tr><tr><td></td><td>Б</td><td></td></tr><tr><td>А</td><td></td><td></td></tr></table> |  |  |  |  | Б |  | А |  |  | «Б» не может находиться в первом столбце и последней строке |
|                 |   |  |  |  |  |   |  |   |  |  |   |
|                 | Б   |  |  |  |  |   |  |   |  |  |   |
| А               |   |  |  |  |  |   |  |   |  |  |   |
|                 |   | «А» не может находиться в первой строке и последнем столбце    |  |  |  |   |  |   |  |  |   |

Отношение типа  $n_4$  «рядом», соответственно, комбинирует эти два отношения, то есть «А» справа от «Б» или «А» слева от «Б».

Задали ограничения, требуемые условиями задачи.

Ограничения 1-го типа для  $n_1 = 3$ :

- 1) У объекта 0 свойство 0 имеет значение 1;
- 2) У объекта 2 свойство 1 имеет значение 8;
- 3) У объекта 8 свойство 1 имеет значение 7.

Ограничения 2-го типа для  $n_2 = 5$ :

- 1) Если 0-е свойство объекта имеет значение 8, то 1-ое свойство этого же объекта имеет значение 1. Аналогично и наоборот;
- 2) Если 1-е свойство объекта имеет значение 3, то 2-е свойство этого же объекта имеет значение 5. Аналогично и наоборот;
- 3) Если 2-е свойство объекта имеет значение 0, то 3-ое свойство этого же объекта имеет значение 6. Аналогично и наоборот.
- 4) Если 3-е свойство объекта имеет значение 3, то 2-ое свойство этого же объекта имеет значение 2. Аналогично и наоборот.
- 5) Если 0-е свойство объекта имеет значение 2, то 3-ое свойство этого же объекта имеет значение 4. Аналогично и наоборот.

Ограничения 3-го типа для  $n_3 = 4$ :

- 1) Если у объекта свойство 0 имеет значение 3, то его соседом слева является объект со значением 2 свойства 3. Если у объекта свойство 3 имеет значение 2, то его соседом снизу слева является объект со значением 3 свойства 0;
- 2) Если у объекта свойство 0 имеет значение 6, то его соседом справа является объект со значением 8 свойства 2. Если у объекта свойство 2 имеет значение 8, то его соседом слева является объект со значением 6 свойства 0;
- 3) Если у объекта свойство 2 имеет значение 1, то его соседом справа является объект со

значением 7 свойства 0. Если у объекта свойство 0 имеет значение 7, то его соседом слева является объект со значением 1 свойства 2;

- 4) Если у объекта свойство 2 имеет значение 7, то его соседом слева является объект со значением 7 свойства 3. Если у объекта свойство 3 имеет значение 7, то его соседом снизу слева является объект со значением 7 свойства 2.

Ограничения 4-го типа для  $n_4 = 4$ :

- 1) Если у объекта свойство 1 имеет значение 6, то он находится слева или справа от объекта со значением 0 свойства 0;
- 2) Если у объекта свойство 1 имеет значение 4, то он находится слева или справа от объекта со значением 5 свойства 1.
- 3) Если у объекта свойство 3 имеет значение 5, то он находится слева или справа от объекта со значением 4 свойства 2.
- 4) Если у объекта свойство 2 имеет значение 3, то он находится слева или справа от объекта со значением 2 свойства 3.

Для однозначного решения поставленной задачи данных ограничений оказалось недостаточно. Чтобы получить единственную интерпретацию, необходимо добавить следующие ограничения:

1-го типа:

- 1) У объекта 3 свойство 8 имеет значение 5;
- 2) У объекта 2 свойство 1 имеет значение 0;
- 3) У объекта 1 свойство 8 имеет значение 5;
- 4) У объекта 3 свойство 3 имеет значение 8;
- 5) У объекта 0 свойство 0 имеет значение 7;
- 6) У объекта 0 свойство 6 имеет значение 3;
- 7) У объекта 2 свойство 0 имеет значение 5;

После добавления дополнительных ограничений выходными данными программы является искомое единственное решение:

1 solutions:

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

## Заключение

По результату работы было изучено программное средство работы с бинарными решающими диаграммами (BDD) – библиотека BuDDy. Она оказалась достаточно удобной в работе за счет перегрузки логических операторов для класса bdd и предоставляет удобный инструментарий для абстрагированного решения задач, и от пользователя требуется только придумать удовлетворяющую требуемым ограничениям и условиям абстрактную логическую модель.

В ходе работы было получено практическое подтверждение того, что BDD являются эффективным средством при решении задач большой размерности. При традиционном представлении структур данных такие задачи решать невозможно из-за большого числа переменных, а также из-за сложности реализации операций над этими данными. BDD упрощают создание ограничений, необходимых для поиска единственного решения.

## Приложение

### Код программы

```
#pragma comment(lib, "bdd.lib")
#include "bdd.h"
#include <fstream>

using namespace std;

ofstream out;

#define N 9 //Число объектов
#define M 4 // Число свойств
#define Log_N 4 //Число булевых переменных для кодирования 1 свойства
#define N_VAR N*M*Log_N // число булевых переменных
#define K 13 // сумма свойств объектов
#define N_COLUMN 3//число столбиков в таблице

void fun(char* varset, int size); //функция, используемая для вывода решений
void print(); // Печать в файл

//Ограничение типа 1
bdd o1(const bdd p[N][N], const int num, const int value)
{
    return p[num][value];
}

//Ограничение типа 2
bdd o2(const bdd p1[N][N], const int value1, const bdd p2[N][N], const int value2)
{
    bdd temp = bddtrue;
    for (int i = 0; i < N; i++)
    {
        temp &= !(p1[i][value1] ^ p2[i][value2]);
    }
    return temp;
}

//Ограничение типа 3
bdd o3_1(const bdd current[N][N], const int cvalue, bdd next[N][N], const int
nvalue)//расписываем для соседа слева снизу
```

```

{
    bdd temp = bddtrue;

    for (int i = 0; i < N; i++)
    {
        if (i % N_COLUMN == N_COLUMN - 1) //крайний правый столбец не может быть левым
нижним соседом
            temp &= !next[i][nvalue];
        if (i % N_COLUMN == 0) //крайний левый столбец не может иметь левого нижнего соседа
            temp &= !current[i][cvalue];
    }
    for (int i = N - N_COLUMN; i < N; i++)
        temp &= !current[i][cvalue]; //последняя строка не может иметь левого нижнего соседа
    for (int i = 0; i < N_COLUMN; i++)
        temp &= !next[i][nvalue]; //1 строка не может быть чьим-то левым нижним соседом
    for (int i = 1; i < N - N_COLUMN; i++)
        temp &= !(current[i][cvalue] ^ next[i + N_COLUMN - 1][nvalue]);
    return temp;
}

bdd o3_2(const bdd current[N][N], const int cvalue, bdd next[N][N], const int
nvalue) //расписываем для соседа слева сверху
{
    bdd temp = bddtrue;

    for (int i = 0; i < N; i++)
    {
        if (i % N_COLUMN == 0) //крайний левый столбец не может быть правым нижним соседом
            temp &= !next[i][nvalue];
        if (i % N_COLUMN == N_COLUMN - 1) //крайний правый столбец не может иметь правого
нижнего соседа
            temp &= !current[i][cvalue];
    }
    for (int i = N - N_COLUMN; i < N; i++)
        temp &= !next[i][nvalue]; //последняя строка не может быть чьим-то правым нижним
соседом
    for (int i = 0; i < N_COLUMN; i++)
        temp &= !current[i][cvalue]; //1 строка не может иметь правого нижнего соседа
    for (int i = 0; i < N - N_COLUMN - 1; i++)
        temp &= !(current[i][cvalue] ^ next[i - N_COLUMN + 1][nvalue]);
    return temp;
}

//ограничение типа 4
bdd o4(const bdd current[N][N], const int cvalue, bdd next[N][N], const int nvalue)
{
    bdd temp1 = bddtrue;
    bdd temp2 = bddtrue;
    temp1 &= o3_1(current, cvalue, next, nvalue);
    temp2 &= o3_2(current, cvalue, next, nvalue);
    return temp1 | temp2;
}

int main()
{
    bdd_init(1000000, 100000); //Выделяем память для 1000000 строк таблицы и КЭШ размером
100000
    bdd_setvarnum(N_VAR); //задаем количество булевых переменных
    bdd p[M][N][N]; //Имеем M свойств у N объектов, каждое из которых принимает N значений
    unsigned I = 0;
    for (unsigned i = 0; i < N; i++)

```

```

{
    for (unsigned j = 0; j < N; j++)
    {
        for (unsigned k = 0; k < M; k++)
        {
            p[k][i][j] = bddtrue;
            for (unsigned m = 0; m < Log_N; m++)
            {
                p[k][i][j] &= ((j >> m) & 1) ? bdd_ithvar(I + Log_N * k + m) :
bdd_nithvar(I + Log_N * k + m);
            }
        }
        I += Log_N * M;
    }
}

```

bdd task = bddtrue; //Решение. Изначально true

//ограничение 6(по умолчанию)

```

for (unsigned i = 0; i < N; i++)
{
    bdd temp[M];
    for (int m = 0; m < M; m++)
        temp[m] = bddfalsе;
    for (unsigned j = 0; j < N; j++)
        for (int m = 0; m < M; m++)
            temp[m] |= p[m][i][j];
    for (int m = 0; m < M; m++)
        task &= temp[m];
}

```

//3 ограничения типа 1

```

task &= o1(p[0], 2, 4); // свойство 0 у объекта 2 имеет значение 4
task &= o1(p[1], 5, 0);
task &= o1(p[2], 7, 3);

```

```

task &= o1(p[3], 8, 5);
task &= o1(p[2], 1, 0);
task &= o1(p[1], 8, 5);
task &= o1(p[0], 0, 7);
task &= o1(p[0], 6, 3);
task &= o1(p[1], 2, 7);
task &= o1(p[2], 0, 5);

```

//5 ограничений типа 2

```

task &= o2(p[3], 5, p[0], 8); // объект со значением 5 в свойстве 3 имеет в свойстве 0
значение 8
task &= o2(p[2], 3, p[1], 1);
task &= o2(p[1], 0, p[2], 6);
task &= o2(p[2], 2, p[1], 8);
task &= o2(p[3], 8, p[1], 4);

```

//4 ограничения типа 3

```

task &= o3_1(p[3], 6, p[2], 7); //объект со значением 7 в свойстве 2 стоит слева снизу от
объекта со значением 6 в свойстве 3
task &= o3_1(p[2], 0, p[3], 3);

```

```
task &= o3_2(p[0], 5, p[3], 1);
task &= o3_2(p[1], 8, p[1], 2);
```

```
//4 ограничения типа 4
task &= o4(p[4], 3, p[8], 1);
task &= o4(p[1], 8, p[3], 4); //объект со значением 8 в свойстве 1 стоит слева снизу или
слева сверху от объекта со значением 4 в свойстве 3
task &= o4(p[2], 0, p[3], 3);
task &= o4(p[3], 3, p[0], 2);
```

```
// ограничение типа 7: сумма свойств нечётных объектов не должна быть больше K
```

```
bdd temp = bddtrue;
for (int i = 0; i < N; i++)
{
    if (i % N_COLUMN == N_COLUMN - 1 && i < N - N_COLUMN) { // нижняя крайняя строка
(сосед только сверху)

        for (int j1 = 0; j1 < N; j1++)
        {
            for (int j2 = 0; j2 < N; j2++)
            {
                for (int j3 = 0; j3 < N; j3++)
                {
                    for (int j4 = 0; j4 < N; j4++)
                    {
                        if (j1 + j2 + j3 + j4 > K)
                        {
                            temp &= !(p[0][i + N_COLUMN - 1][j1] &
p[1][i + N_COLUMN - 1][j2] & p[2][i + N_COLUMN - 1][j3] & p[3][i + N_COLUMN - 1][j4]);
                        }
                    }
                }
            }
        }

    else if (i % N_COLUMN == 0 && i > N_COLUMN) // верхняя крайняя строка (сосед только
снизу)
    {
        for (int j1 = 0; j1 < N; j1++)
        {
            for (int j2 = 0; j2 < N; j2++)
            {
                for (int j3 = 0; j3 < N; j3++)
                {
                    for (int j4 = 0; j4 < N; j4++)
                    {
                        if (j1 + j2 + j3 + j4 > K)
                        {
                            temp &= !(p[0][i - N_COLUMN + 1][j1] &
p[1][i - N_COLUMN + 1][j2] & p[2][i - N_COLUMN + 1][j3] & p[3][i - N_COLUMN + 1][j4]);
                        }
                    }
                }
            }
        }
    }
}
```



```

    }
    }
}

else // не крайние строки (сосед снизу и сверху)
{
    for (int j1 = 0; j1 < N; j1++)
    {
        for (int j2 = 0; j2 < N; j2++)
        {
            for (int j3 = 0; j3 < N; j3++)
            {
                for (int j4 = 0; j4 < N; j4++)
                {
                    if (j1 + j2 + j3 + j4 > K)
                    {
                        if (i < N - N_COLUMN)
                        {
                            temp &= !(p[0][i + N_COLUMN - 1][j1]
& p[1][i + N_COLUMN - 1][j2] & p[2][i + N_COLUMN - 1][j3] & p[3][i + N_COLUMN - 1][j4]);
                        }
                        if (i > N_COLUMN)
                        {
                            temp &= !(p[0][i - N_COLUMN + 1][j1]
& p[1][i - N_COLUMN + 1][j2] & p[2][i - N_COLUMN + 1][j3] & p[3][i - N_COLUMN + 1][j4]);
                        }
                    }
                }
            }
        }
    }

    task &= temp; // вставка условия в общее решение
}

//ограничение 5(по умолчанию)
for (unsigned m = 0; m < M; m++)
{
    for (unsigned j = 0; j < N; j++)
        for (unsigned i = 0; i < N - 1; i++)
            for (unsigned k = i + 1; k < N; k++)
                task &= p[m][i][j] >> !p[m][k][j];
}
out.open("out.txt");
unsigned satcount = (unsigned)bdd_satcount(task); // количество решений
out << satcount << " solutions:\n" << endl;
if (satcount) bdd_allsat(task, fun);
out.close();
bdd_done();
}
char var[N_VAR];
void print() {
    for (unsigned i = 0; i < N; i++) {
        out << i << ": ";
        for (unsigned j = 0; j < M; j++) {
            int J = i * M * Log_N + j * Log_N;
            int num = 0;
            for (unsigned k = 0; k < Log_N; k++) num += (unsigned)
                (var[J + k] << k);
            out << num << ' ';
        }
        out << endl;
    }
    out << endl;
}

```

```

}
void build(char* varset, unsigned n, unsigned I) {
    if (I == n - 1) {
        if (varset[I] >= 0) {
            var[I] = varset[I];
            print();
            return;
        }
        var[I] = 0;
        print();
        var[I] = 1;
        print();
        return;
    }
    if (varset[I] >= 0) {
        var[I] = varset[I];
        build(varset, n, I + 1);
        return;
    }
    var[I] = 0;
    build(varset, n, I + 1);
    var[I] = 1;
    build(varset, n, I + 1);
}
void fun(char* varset, int size) {
    build(varset, size, 0);
}

```