|  |  |
| --- | --- |
| Санкт-Петербургский политехнический университет Петра Великого  Институт металлургии, машиностроения и транспорта  Высшая школа автоматизации и робототехники | |
| Курсовая работа | |
| Дисциплина: Программирование на языках высокого уровня  Тема: Метод секущих плоскостей, Метод Гомори | |
|  | |
| Выполнил студент гр. 3331506/70401 | Патрушева А.И. |
| Преподаватель | Ананьевский М.С. |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.   |  | | --- | | Санкт-Петербург  2020 | | |

Содержание

[1. Формулировка задачи, которую решает алгоритм 3](#_Toc38874821)

[2. Словесное описание алгоритма 4](#_Toc38874822)

[3. Реализация алгоритма 4](#_Toc38874823)

[4. Анализ алгоритма 18](#_Toc38874824)

[5. Применение алгоритма 19](#_Toc38874825)

[6. Заключение 19](#_Toc38874826)

# Формулировка задачи, которую решает алгоритм

Линейное программирование – математическая дисциплина, посвященная теории и методам решения экстремальных задач на множествах n- мерного пространства, задаваемых системами линейными уравнений и неравенств.

Существует ряд алгоритмов, решающих задачу целочисленного линейного программирования точно. Один из классов таких алгоритмов — [методы секущих плоскостей](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%93%D0%BE%D0%BC%D0%BE%D1%80%D0%B8) (методы Гомори), которые работают путём решения ослабленной линейной задачи с последующим добавлением линейных ограничений, которые отсекают нецелочисленное решение задачи без отсечения целочисленных допустимых решений.

Большое практическое значение имеют методы решения задач линейного программирования, с помощью которых можно найти оптимальный план, координаты которого - целые числа. Задачи целочисленного программирования решаются именно такими методами.

Метод Гомори является универсальным методом решения задач целочисленного программирования, с помощью которого после конечного числа итераций можно найти оптимальный план или убедиться в том, что задача не имеет решений.

Рассмотрим следующую задачу ЦЛП:

1. *F = c1 x1+ c2 x2 + cn xn → max*

Тогда решить задачу оптимизации означает одно из:

* Найти такие
* Или показать, что множество *Х = Ø*

В данной работе будет реализован расширенный метод Гомори, который использует перед этапом симплекс-метода, метод искусственного базиса, для облегчения поиска первоначального опорного плана. Явно выделенным базисом будем называть вектора, где только одна координата вектора ненулевая и равна 1. Благодаря введению в базис искусственных переменных, содержащих М (достаточно большое положительное число >> 1), первыми выводиться будут именно они, что облегчает задачу.

# Словесное описание алгоритма

На каждом этапе решается соответствующая непрерывная задача линейного программирования, и если ее решение не является целочисленным, то на следующем этапе к ней добавляется дополнительное ограничение, так называемое правильное сечение. Р. Гомори предложил метод, гарантирующий решение следующей задачи за конечное число шагов.

1. Решаем исходную задачу (пункты 1-3) без условия 4 методом искусственного базиса. Если задача имеет целочисленное решение, то оно и будет решением задачи. В противном случае переходим к следующему пункту.
2. Выбирается одна из нецелых компонент решения (например, имеющая наибольшую дробную часть), и строится правильное сечение.

*Рассмотрим правило построения правильных сечений*:

Пусть решение исходной задачи без требования целочисленности симплекс-методом на последнем шаге дало решение, в котором базисные переменные , …, связаны с небазисными переменными, …, формулами:

В этом случае вектор является оптимальным решением. Пусть В этом случае в качестве правильного сечения можно взять неравенство:

1. Полученное неравенство преобразуется в равенство путем добавления новой неотрицательной переменной:

Это равенство присоединяется к задаче, рассмотренной в первом пункте.

1. Решаем задачу с добавленным в пункте 3 ограничением. Если ее решение оказывается целочисленным, то оно будет решением поставленной задачи. В противном случае переходим к пункту 2 и добавляем новое правильное сечение.

# Реализация алгоритма

Реализация алгоритма Гомори, а также используемых в нем симплекс метода и метода искусственного базиса, в виде листинга программы и текстового пользовательского интерфейсом приведен ниже:

#include <iostream>

#include <iomanip>

#include <vector>

#include <sstream>

#include <string>

#include <math.h>

/\*Обертка для вещественных чисел. Нужна для M-задачи.

\* Такое число представляет две компоненты - вещественную часть

\* и коэффициент при некотором числе M.

\*/

class number {

private:

// вещественная часть

double real;

// коэффициет при M

double coef;

// флаг "не число"

bool is\_nan;

public:

/\*\*

\* Конструктор.

\* r - вещественная компонента

\* c - коэффициент при M

\* num - если этот аргумент будет false, то число считается как NaN

\*/

number(double r = 0, double c = 0, bool num = true) {

real = r;

coef = c;

is\_nan = !num;

}

double get\_real() {

return real;

}

double get\_coef() {

return coef;

}

/\*\*

\* Методы перегрузки арифметических операторов.

\*/

number operator+(number num) {

number res;

res.real = real + num.real;

res.coef = coef + num.coef;

return res;

}

void operator+=(number num) {

real += num.real;

coef += num.coef;

}

number operator-(number num) {

number res;

res.real = real - num.real;

res.coef = coef - num.coef;

return res;

}

void operator-=(number num) {

real -= num.real;

coef -= num.coef;

}

number operator\*(number num) {

number res;

res.real = real \* num.real;

if (real != 0 && coef == 0 && num.coef != 0) {

res.coef = real \* num.coef;

}

else if (num.real != 0 && num.coef == 0 && coef != 0) {

res.coef = coef \* num.real;

}

return res;

}

void operator\*=(number num) {

real \*= num.real;

if (real != 0 && coef == 0 && num.coef != 0) {

coef = real \* num.coef;

}

else if (num.real != 0 && num.coef == 0 && coef != 0) {

coef = coef \* num.real;

}

}

number operator/(number num) {

number res;

res.real = real / num.real;

return res;

}

void operator/=(number num) {

real /= num.real;

}

bool operator!=(number num) {

return (real != num.real && coef != num.coef);

}

bool operator!=(double num) {

return (real != num);

}

bool operator<(number num) {

if (coef != num.coef) {

return (coef < num.coef);

}

return (real < num.real);

}

bool operator>(number num) {

if (coef != num.coef) {

return (coef > num.coef);

}

return (real > num.real);

}

bool operator==(number num) {

return (real == num.real && coef == num.coef);

}

bool operator>=(number num) {

return this->operator>(num) || this->operator==(num);

}

bool operator<=(number num) {

return this->operator<(num) || this->operator==(num);

}

/\*\*

\* Вывод в виде строки в формате

\* [R][+,-][C]M

\* где R - вещественная часть,

\* C - коэффициент при M.

\*/

std::string str() {

if (is\_nan) {

return std::string("NaN");

}

if (real == 0 && coef == 0) {

return std::string("0");

}

std::stringstream ss;

ss << std::setprecision(2);

if (real != 0) {

ss << real;

}

if (coef != 0) {

if (real != 0 && coef > 0) {

ss << '+';

}

else if (coef < 0) {

ss << '-';

}

if (coef != 1 && coef != -1) {

ss << abs(coef);

}

ss << 'M';

}

return ss.str();

}

/\*\*

\* Возвращает дробную часть вещественной компоненты

\*/

number fract() {

return (real > 0) ? number(real - floor(real)) : number(abs(floor(real)) + (real));

}

/\*\*

\* Проверяет является ли число (почти) целым

\*/

bool is\_integer() {

return this->fract() < 0.00001;

}

/\*\*

\* Модуль числа

\*/

number absolute() {

return number(fabs(real));

}

};

/\* Несколько тайпдефов

\*/

typedef std::vector<number> vector\_nums;

typedef std::vector<int> vector\_ints;

typedef std::vector<std::vector<number> > matrix\_nums;

/\* Несколько функций для вывода в консоль

\*/

void print\_line() {

std::cout << std::endl;

for (int i = 0; i < 80; ++i) {

std::cout << (char)0xCD;

}

std::cout << std::endl;

}

/\*\*

\* Печать строки

\*/

void print\_str(std::string msg) {

std::cout << std::endl << msg << std::endl;

}

/\*\*

\* Печатает число

\*/

void print\_num(std::string msg, number n) {

std::cout << std::endl << msg << n.str() << std::endl;

}

/\*\*

\* Печатает вещественное число

\*/

void print\_num(std::string msg, double n) {

std::cout << std::endl << msg << n << std::endl;

}

/\*\*

\* Печатает целое число

\*/

void print\_num(std::string msg, int n) {

std::cout << std::endl << msg << n << std::endl;

}

/\*\*

\* Печает вектор

\*/

void print\_vector(std::string msg, vector\_nums v) {

unsigned int i, j;

unsigned int width = 8;

std::cout << std::endl << msg << std::endl << (char)0xDA;

for (i = 0; i < v.size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < v.size() - 1) {

std::cout << (char)0xC2;

}

}

std::cout << (char)0xBF << std::endl << (char)0xB3;

for (i = 0; i < v.size(); ++i) {

std::cout << std::setw(width - 1) << v[i].str() << (char)0xB3;

}

std::cout << std::endl << (char)0xC0;

for (i = 0; i < v.size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < v.size() - 1) {

std::cout << (char)0xC1;

}

}

std::cout << (char)0xD9 << std::endl;

}

/\*\*

\* Печатает вектор целых чисел

\*/

void print\_vector(std::string msg, std::vector<int> v) {

std::cout << std::endl << msg << std::endl << (char)0xDA;

unsigned int i, j;

unsigned int width = 8;

for (i = 0; i < v.size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < v.size() - 1) {

std::cout << (char)0xC2;

}

}

std::cout << (char)0xBF << std::endl << (char)0xB3;

for (i = 0; i < v.size(); ++i) {

std::cout << std::setw(width - 1) << v[i] << (char)0xB3;

}

std::cout << std::endl << (char)0xC0;

for (i = 0; i < v.size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < v.size() - 1) {

std::cout << (char)0xC1;

}

}

std::cout << (char)0xD9 << std::endl;

}

/\*\*

\* Печатает матрицу

\*/

void print\_matrix(std::string msg, matrix\_nums m) {

unsigned int i, j, k;

unsigned int width = 8;

std::cout << std::endl << msg << std::endl << (char)0xDA;

for (i = 0; i < m[0].size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < m[0].size() - 1) {

std::cout << (char)0xC2;

}

}

std::cout << (char)0xBF << std::endl;

for (i = 0; i < m.size(); ++i) {

std::cout << (char)0xB3;

for (j = 0; j < m[i].size(); ++j) {

std::cout << std::setw(width - 1) << std::right << m[i][j].str() << (char)0xB3;

}

std::cout << std::endl;

if (i < m.size() - 1) {

std::cout << (char)0xC3;

for (j = 0; j < m[i].size(); ++j) {

for (k = 0; k < width - 1; ++k) {

std::cout << (char)0xC4;

}

if (j < m[i].size() - 1) {

std::cout << (char)0xC5;

}

else {

std::cout << (char)0xB4;

}

}

std::cout << std::endl;

}

}

std::cout << (char)0xC0;

for (i = 0; i < m[0].size(); ++i) {

for (j = 0; j < width - 1; ++j) {

std::cout << (char)0xC4;

}

if (i < m[0].size() - 1) {

std::cout << (char)0xC1;

}

}

std::cout << (char)0xD9 << std::endl;

std::cout << std::endl;

}

// максимальное количество итераций метода

#define DEBUG\_MAX\_ITER\_NUM 10

/\* Исключение \*/

class simplex\_error {

public:

std::string msg;

simplex\_error(std::string m) {

msg = std::string(m);

}

};

/\*структура таблицы симплекс-метода \*/

struct opt\_table {

vector\_nums cib;

vector\_ints bp;

vector\_nums br;

vector\_nums c;

matrix\_nums a;

vector\_nums z;

vector\_nums delta;

};

/\*Проверка целочисленности компонент вектора \*/

bool vector\_is\_integer(vector\_nums v) {

for (unsigned int i = 0; i < v.size(); ++i) {

if (!v[i].is\_integer()) {

return false;

}

}

return true;

}

/\*Симплекс метод поиска максимума\*/

opt\_table simplex\_max(opt\_table task, bool negative = false, bool int\_break = false) {

int i, j;

int n = task.c.size();

int m = task.bp.size();

vector\_nums cib(m);

vector\_nums z(n);

vector\_nums delta(n);

vector\_nums min(m);

matrix\_nums tmp\_a(m, vector\_nums(n));

vector\_nums tmp\_br(m);

for (i = 0; i < m; ++i) {

if (task.bp[i] == -1) {

continue;

}

cib[i] = task.c[task.bp[i]];

}

for (int k = 0; k < DEBUG\_MAX\_ITER\_NUM; ++k) {

print\_line();

// определяем разрешающий столбец

number deltaExtr;

if (negative) {

deltaExtr = 1000000;

}

else {

deltaExtr = delta[0];

}

int r = -1;

for (j = 0; j < n; ++j) {

z[j] = number();

for (i = 0; i < m; ++i) {

if (task.bp[i] == -1) {

continue;

}

z[j] += cib[i] \* task.a[i][j];

}

delta[j] = task.c[j] - z[j];

// наименьшая по модулю отрицательная оценка

if (negative) {

if (delta[j] < number(0) && delta[j].absolute() < deltaExtr){

deltaExtr = delta[j].absolute();

r = j;

}

// наибольшая оценка

}

else {

if (deltaExtr < delta[j]) {

deltaExtr = delta[j];

r = j;

}

}

}

print\_vector("Coefs of variables (c)", task.c); //Коэффициенты при М (переменных) (c)

print\_vector("Coefs of basic variables (cib)", cib);//Коэффициенты основных переменных (cib)

print\_vector("Basic variables numbers (bp)", task.bp);//Количество основных переменных (bp)

print\_vector("Basic variables values (br)", task.br);//Значения основных переменных (br)

print\_matrix("Coefs of system (a)", task.a); //Коэффициенты системы(а)

print\_vector("Values (z)", z); //Значения(z)

print\_vector("Relative valuations (delta)", delta); //Относительные оценки (Дельта)

print\_num("Permitted column number (r): ", r); //Номер разрешающего столбца (r)

if (r == -1) {

break;

}

// определяем разрешающую строку

number min\_row = number(100000);

int s = -1;

for (i = 0; i < m; ++i) {

if (task.bp[i] == -1) {

s = i;

break;

}

if (task.a[i][r] != 0) {

min[i] = task.br[i] / task.a[i][r];

}

else {

min[i] = number(0, 0, false);

continue;

}

if (min[i] < 0) {

min[i] = number(0, 0, false);

continue;

}

if (min\_row > min[i]) {

min\_row = min[i];

s = i;

}

}

//print\_vector("Relations x/a (br[i]/a[i][r], min): ", min);

if (s == -1) {

throw simplex\_error("Permitted row not found");//разрешающая строка не найдена

}

print\_num("Permitted row number (s): ", s); //Номер разрешающей строки

number element = task.a[s][r];//Разрешающий элемент

print\_num("Permitted value (element): ", element);

for (i = 0; i < m; ++i) {

for (j = 0; j < n; ++j) {

tmp\_a[i][j] = task.a[i][j];

}

tmp\_br[i] = task.br[i];

}

// вносим переменную в базис

task.bp[s] = r;

cib[s] = task.c[r];

for (j = 0; j < n; ++j) {

task.a[s][j] /= element;

}

task.br[s] /= element;

for (i = 0; i < m; ++i) {

if (i == s) {

continue;

}

number air = tmp\_a[i][r];

for (j = 0; j < n; ++j) {

task.a[i][j] -= (air \* tmp\_a[s][j]) / element;

}

task.br[i] -= (air \* tmp\_br[s]) / element;

}

if (int\_break && vector\_is\_integer(task.br)) {

break;

}

}

print\_line();

print\_vector("Result: ", task.br);

opt\_table tab;

tab.cib = cib;

tab.bp = task.bp;

tab.br = task.br;

tab.c = task.c;

tab.a = task.a;

tab.z = z;

tab.delta = delta;

return tab;

}

/\*Сравнение по модулю 1\*/

bool cmp\_mod\_one(number a, number b) {

return (a - b).is\_integer();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Проверка значения элемента в массиве\*/

bool in\_vector(int a, vector\_ints v) {

for (int i = 0; i < v.size(); ++i) {

if (a == v[i]) {

return true;

}

}

return false;

}

/\*Метод Гомори\*/

opt\_table gomory(opt\_table tab) {

tab = simplex\_max(tab, false, false);

int i, j;

// общее количество переменных

int n = tab.c.size();

// количество базисных переменных

int m = tab.bp.size();

for (int k = 0; k < DEBUG\_MAX\_ITER\_NUM; ++k) {

print\_line();

std::cout << "Gomory iteration #" << k << std::endl;

print\_line();

// условие выхода

if (vector\_is\_integer(tab.br)) {

return tab;

}

// максимальная дробная часть

number max\_fract = 0;

// номер строки с максимальной дробной частью

int r = -1;

// поиск переменной с максимальной дробной частью

for (i = 0; i < m; ++i) {

number fract = tab.br[i].fract();

if (fract >= max\_fract) {

max\_fract = fract;

r = i;

}

}

// составляем новую таблицу, добавляя один столбец и одну строку

++n;

++m;

opt\_table ext\_tab;

// вектор коэффициентов всех переменных

ext\_tab.c = vector\_nums(n);

for (j = 0; j < n - 1; ++j) {

ext\_tab.c[j] = tab.c[j];

}

ext\_tab.c[n - 1] = 0;

// матрица a

ext\_tab.a = matrix\_nums(m, vector\_nums(n));

for (i = 0; i < m - 1; ++i) {

for (j = 0; j < n - 1; ++j) {

ext\_tab.a[i][j] = tab.a[i][j];

}

ext\_tab.a[i][n - 1] = 0;

}

for (j = 0; j < n - 1; ++j) {

if (in\_vector(j, tab.bp)) {

ext\_tab.a[m - 1][j] = 0;

}

else {

ext\_tab.a[m - 1][j] = tab.a[r][j].fract();

}

}

ext\_tab.a[m - 1][n - 1] = -1;

// вектор-решение

ext\_tab.bp = vector\_ints(m);

ext\_tab.br = vector\_nums(m);

for (i = 0; i < m - 1; ++i) {

ext\_tab.bp[i] = tab.bp[i];

ext\_tab.br[i] = tab.br[i];

}

ext\_tab.bp[m - 1] = -1;

ext\_tab.br[m - 1] = tab.br[r].fract();

tab = simplex\_max(ext\_tab, true, true);

}

return tab;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char\* argv[]) {

int n = 6; //число переменных; 4 для первого примера

int m = 3; //число уравнений; 2 для первого примера

/\*vector\_nums c(n);

c[0] = number(1);

c[1] = number(-1);

c[2] = number(0);

c[3] = number(0);

c[4] = number(0, -1);

matrix\_nums a(m, vector\_nums(n));

a[0][0] = number(-1);

a[0][1] = number(2);

a[0][2] = number(-1);

a[0][3] = number(0);

a[0][4] = number(1);

a[1][0] = number(3);

a[1][1] = number(2);

a[1][2] = number(0);

a[1][3] = number(1);

a[1][4] = number(0);

vector\_ints bp(m);

bp[0] = 4;

bp[1] = 3;

vector\_nums br(m);

br[0] = number(4);

br[1] = number(14);

opt\_table task;

task.a = a;

task.bp = bp;

task.br = br;

task.c = c;\*/

/\*opt\_table task;

task.c = vector\_nums(n);

task.c[0] = number(2);

task.c[1] = number(1);

task.c[2] = number(0);

task.a = matrix\_nums(m, vector\_nums(n));

task.a[0][0] = number(15);

task.a[0][1] = number(30);

task.a[0][2] = number(1);

task.bp = vector\_ints(m);

task.bp[0] = 2;

task.br = vector\_nums(m);

task.br[0] = number(96);\*/

opt\_table task;

task.c = vector\_nums(n);

task.c[0] = number(1);

task.c[1] = number(0);

task.c[2] = number(0);

task.c[3] = number(0);

task.c[4] = number(0);

task.c[5] = number(0, -1);

task.a = matrix\_nums(m, vector\_nums(n));

task.a[0][0] = number(1);

task.a[0][1] = number(-2);

task.a[0][2] = number(1);

task.a[0][3] = number(0);

task.a[0][4] = number(0);

task.a[0][5] = number(0);

task.a[1][0] = number(1);

task.a[1][1] = number(-1);

task.a[1][2] = number(0);

task.a[1][3] = number(-1);

task.a[1][4] = number(0);

task.a[1][5] = number(1);

task.a[2][0] = number(1);

task.a[2][1] = number(1);

task.a[2][2] = number(0);

task.a[2][3] = number(0);

task.a[2][4] = number(1);

task.a[2][5] = number(0);

task.bp = vector\_ints(m);

task.bp[0] = 2;

task.bp[1] = 5;

task.bp[2] = 4;

task.br = vector\_nums(m);

task.br[0] = number(4);

task.br[1] = number(-1);

task.br[2] = number(2);

//запись времени в момент начала работы алгоритма

unsigned int start\_time = clock();

gomory(task);

unsigned int end\_time = clock(); //конец работы

unsigned int search\_time = end\_time - start\_time;

std::cout << '\n' << "time is " << search\_time << '\n'; //выводим время работы

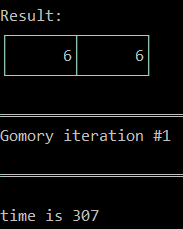
return 0;

}

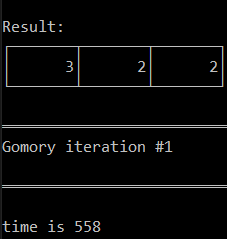
# Анализ алгоритма

Для анализа метода возьмем две задачи целочисленного линейного программирования и сравним скорость их решения программой, приведенной выше. В первой задаче необходимо найти оптимальный план, состоящий из 2 значений, а во второй из 3. Дополнительно рассмотрим случай, когда метод Гомори не применялся, так как метод искусственного базиса дал целочисленный оптимальный план. Время в мс решения каждой из задач представлены ниже.

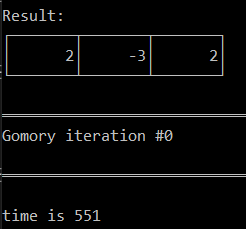
Оптимальный план из 2 значений (одна итерация метода Гомори):



Оптимальный план из 3 значений (одна итерация метода Гомори):



Оптимальный план из 3 значений (ноль итераций метода Гомори):



На самом деле, метод Гомори это всего лишь "надстройка" над обычным симплекс-методом, поэтому время его выполнения также зависит от скорости симплекс- метода, а в нашем случае, еще и от скорости метода искусственного базиса.

# Применение алгоритма

Задача целочисленного программирования — это задача математической оптимизации. Где оптимизация – это поиск экстремума некой функции. Метод Гомори используется в производственном планировании (максимизация дохода без выхода за границы имеющихся ресурсов), планировании (обслуживание и расписание работы транспортной сети), сетях передачи данных (обеспечение предопределенных требований за минимальную цену), сотовых сетях (требуется распределение допустимых частот по антеннам).

# Заключение

В алгоритме Гомори предполагается целочисленность всех переменных. В то же время имеется достаточно много практических задач, в которых целочисленными являются лишь некоторые из переменных. Кроме того, недостатком является чувствительность к ошибкам вычислений при подсчете дробных частей, при округлении чисел и т.д.

По сравнению с методами ветвей и границ, методы отсекающих плоскостей являются более технологичными при программировании, т.к. не требуют дополнительного объема оперативной памяти неопределенного размера для хранения дерева решений.