



**Министерство образования Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
Московский государственный технологический университет  
«СТАНКИН»**

Кафедра прикладной математики  
Учебный курс «Методы оптимизации»

Отчет по лабораторной работе №4  
Вариант №15

Студента группы

ИДБ-18-09  
*Полс А.Д.*

Проверил

*Коробов Н.А.*

Оценка

г. Москва

# Лабораторная работа №4

Выполнил студент группы ИДБ-18-09 Полс А.Д.

## Вариант №15

**Цель работы:** Изучить метод ветвей и границ для решения задачи о коммивояжёре дискретного программирования и применить его на практическом примере

```
In [1]: # необходимые зависимости

import string
import warnings

import numpy as np
import pandas as pd

from typing import Tuple, List
from ast import literal_eval as make_tuple
from IPython.display import display, Markdown
```

```
In [2]: warnings.filterwarnings('ignore', r'All-NaN slice encountered')
```

```
In [3]: # форматированный вывод

def fprint(text):
    if text is str:
        display(Markdown(text))
    else:
        display(Markdown(str(text)))
```

## Матрица варианта

```
In [4]: matrix = np.array(
    [
        [np.nan, 6, 3, 1, 6],
        [4, np.nan, 3, 5, 3],
        [9, 3, np.nan, 4, 4],
        [2, 6, 2, np.nan, 7],
        [3, 1, 1, 9, np.nan]
    ],
    dtype=float
)

graph = pd.DataFrame(
    matrix,
    columns=list(string.ascii_uppercase[:matrix.shape[0]]),
    index=list(string.ascii_uppercase[:matrix.shape[0]])
)

graph
```

Out [4] :

	A	B	C	D	E
A	NaN	6.0	3.0	1.0	6.0
B	4.0	NaN	3.0	5.0	3.0
C	9.0	3.0	NaN	4.0	4.0
D	2.0	6.0	2.0	NaN	7.0
E	3.0	1.0	1.0	9.0	NaN

## Метод ветвей и границ

### Класс дерева

```
In [5]: class Node:
    def __init__(self, frame: pd.DataFrame = None, weight: float = None, path: str = 'root', parent = None):
        self.is_leaf = True
        self.left = None
        self.right = None
        self.frame = frame
        self.parent = parent
        self.weight = weight
        self.path = path
        self.is_forgotten = False

    def grow_left(self, frame: pd.DataFrame, weight: float, path: str) -> None:
        self.left = Node(frame, weight, path, self)
        self._check_growth()

    def grow_right(self, frame: pd.DataFrame, weight: float, path: str) -> None:
        self.right = Node(frame, weight, path, self)
        self._check_growth()

    def _check_growth(self) -> None:
        if self.left is not None and self.right is not None:
            self.is_leaf = False

    def find_least_leaf(self):
        if self.is_leaf:

            return self
        elif self.left is not None and self.right is not None:
            left = self.left.find_least_leaf()
            right = self.right.find_least_leaf()
            if left.weight < right.weight:
                right.is_forgotten = True
                return left
            else:
                left.is_forgotten = True
                return right
        else:
            return self

    def backward(self):
        total_path = list()

        cursor = self
        while True:
            total_path.append(cursor.path)
            if cursor.parent is not None:
                cursor = cursor.parent
            else:
                return total_path
```

## Класс алгоритма решения

In [6]:

```
class BranchAndBoundSolver:

    def __init__(self, initial_frame: pd.DataFrame):
        self.root = Node(
            *BranchAndBoundSolver.reduction(initial_frame)
        )

    def solve(self):

        while self.root.find_least_leaf().frame.to_numpy().shape != (2, 2):
            leaf = self.root.find_least_leaf()

            max_element_value, max_element_position, pos_names = BranchAndBoundSolver.max_element_analysis(
                leaf.frame)

            if leaf.is_forgotten:
                leaf.frame, _ = BranchAndBoundSolver.ban_element(leaf.frame, pos_names)
                max_element_value, max_element_position, pos_names = BranchAndBoundSolver.max_element_analysis(
                    leaf.frame)

            leaf.grow_left(*BranchAndBoundSolver.remove_path(leaf.frame, max_element_position, leaf.weight, pos_names))
            leaf.grow_right(leaf.frame.copy(), leaf.weight + max_element_value, f'skip: {pos_names}')

            last_leaf = self.root.find_least_leaf()
            last_frame = last_leaf.frame.copy(deep=True)

            pre, last = BranchAndBoundSolver.special_max_element_analyses(last_frame)

            self.path = list(reversed(last_leaf.backward()))
            self.path.append(f'add: {pre}')
            self.path.append(f'add: {last}')

        return self.path, last_leaf.weight

    @staticmethod
    def remove_path(frame: pd.DataFrame, max_element_position, last_penalty, pos_names):

        local_frame = frame.copy(deep=True)
        i, j = max_element_position
        begin, end = pos_names

        try:
            local_frame.loc[begin][end] = np.nan
        except:
            pass

        try:
            local_frame.loc[end][begin] = np.nan
        except:
            pass
```

```

local_matrix = local_frame.to_numpy().copy()

local_matrix = np.delete(local_matrix, i, axis=0)
local_matrix = np.delete(local_matrix, j, axis=1)

columns = local_frame.columns[local_frame.columns != end]
index = local_frame.index[local_frame.index != begin]

new_frame = pd.DataFrame(local_matrix.copy(), columns=columns,
                         index=index)

reduced_new_frame, penalty = BranchAndBoundSolver.reduction(new_frame)

return reduced_new_frame.copy(deep=True), penalty + last_penalty, f'add: {pos_names}'

@staticmethod
def special_max_element_analysis(frame: pd.DataFrame):
    local_matrix = frame.to_numpy().copy()
    new_matrix = np.zeros(shape=local_matrix.shape)

    for i in np.arange(local_matrix.shape[0]):
        for j in np.arange(local_matrix.shape[1]):
            stash = local_matrix[i, j]
            local_matrix[i, j] = np.nan
            axis0_min = np.nanmin(local_matrix[i])
            axis1_min = np.nanmin(local_matrix.T[j])
            new_matrix[i, j] = axis0_min + axis1_min
            local_matrix[i, j] = stash

    max_value = np.max(new_matrix)

    first_value_pos = np.argwhere(np.isnan(new_matrix)).T[0]
    second_value_pos = np.argwhere(np.isnan(new_matrix)).T[1]

    first_max_value_indecies = tuple(first_value_pos)
    second_max_value_indecies = tuple(second_value_pos)

    local_frame = pd.DataFrame(new_matrix, columns=frame.columns.copy(),
                               index=frame.index.copy())

    first_begin = local_frame.index[first_max_value_indecies[0]]
    first_end = local_frame.columns[first_max_value_indecies[1]]

    second_begin = local_frame.index[second_max_value_indecies[0]]
    second_end = local_frame.columns[second_max_value_indecies[1]]

    first_pos_names = (first_begin, first_end)
    second_pos_names = (second_begin, second_end)

return first_pos_names, second_pos_names

@staticmethod
def max_element_analysis(frame: pd.DataFrame):
    local_matrix = frame.to_numpy().copy()
    new_matrix = np.zeros(shape=local_matrix.shape)

```

```

        for i in np.arange(local_matrix.shape[0]):
            for j in np.arange(local_matrix.shape[1]):
                stash = local_matrix[i, j]
                local_matrix[i, j] = np.nan

                axis0_min = np.nanmin(local_matrix[i])
                axis1_min = np.nanmin(local_matrix.T[j])

                new_matrix[i, j] = axis0_min + axis1_min

                local_matrix[i, j] = stash

        max_value = np.nanmax(new_matrix)

        max_value_position = np.array(
            np.where(new_matrix == max_value)
        ).T[0]

        max_value_indecies = tuple(max_value_position)

        local_frame = pd.DataFrame(new_matrix, columns=frame.columns.copy(), index=frame.index.copy())

        begin = local_frame.index[max_value_indecies[0]]
        end = local_frame.columns[max_value_indecies[1]]

        pos_names = (begin, end)

    return max_value, max_value_position, pos_names

@staticmethod
def reduction(frame: pd.DataFrame):
    local_matrix = frame.to_numpy().copy()
    weight = 0

    axis1_mins = np.nanmin(local_matrix, axis=1).reshape(-1, 1)
    weight += axis1_mins.sum()
    local_matrix -= axis1_mins

    axis0_mins = np.nanmin(local_matrix, axis=0)
    weight += axis0_mins.sum()
    local_matrix -= axis0_mins

    new_frame = pd.DataFrame(local_matrix, columns=frame.columns, index=frame.index)

    return new_frame, weight

@staticmethod
def ban_element(frame: pd.DataFrame, pos_names):
    local_frame = frame.copy(deep=True)
    local_frame.loc[pos_names[0]][pos_names[1]] = np.nan

    return BranchAndBoundSolver.reduction(local_frame)

```

## Результат решения

```
In [7]: graph
```

```
Out[7]:
```

	A	B	C	D	E
A	NaN	6.0	3.0	1.0	6.0
B	4.0	NaN	3.0	5.0	3.0
C	9.0	3.0	NaN	4.0	4.0
D	2.0	6.0	2.0	NaN	7.0
E	3.0	1.0	1.0	9.0	NaN

```
In [8]: solver = BranchAndBoundSolver(graph)  
branch, weight = solver.solve()
```

## Ветвь дерева с решением

```
In [9]: branch
```

```
Out[9]: ['root',  
         "add: ('A', 'D')",  
         "add: ('D', 'C')",  
         "add: ('B', 'A')",  
         "add: ('C', 'E')",  
         "add: ('E', 'B')"]
```

## Конечная длина маршрута

```
In [10]: weight
```

```
Out[10]: 12.0
```

## Конечный маршрут

```
In [11]: def remove_prefix(text, prefix):
    if text.startswith(prefix):
        return text[len(prefix):]
    return text

verts = list(map(lambda t: make_tuple(remove_prefix(t, 'add: ')), branch[1:]))

chain = 'A'

while len(chain) < graph.shape[0]+1:
    for v in verts:
        if chain[-1] == v[0]:
            chain += v[1]

chain = chain[:7]; chain
```

```
Out[11]: 'ADCEBA'
```

```
In [12]: str_path = '$'

for symbol in chain:
    str_path += rf'{symbol} \rightarrow '

str_path += rf'{weight} $'
```

```
In [13]: fprint(str_path)
```

```
A → D → C → E → B → A → 12.0
```

```
In [ ]:
```

## Ручная проверка решения

<b>15)</b>	$\infty$	6	3	1	6
	4	$\infty$	3	5	3
	9	3	$\infty$	4	4
	2	6	2	$\infty$	7
	3	1	1	9	$\infty$

Рис. 1 Исходная матрица

	‡ A	‡ B	‡ C	‡ D	‡ E
A	nan	6	3	1	6
B	4	nan	3	5	3
C	9	3	nan	4	4
D	2	6	2	nan	7
E	3	1	1	9	nan

Рис. 2 Исходная матрица

Выполним редукцию: из каждой клетки вычтем сумму минимальных элементов в строке, затем с полученной матрицей делаем аналогичное, но по столбцам:

	‡ A	‡ B	‡ C	‡ D	‡ E
A	nan	5	2	0	5
B	1	nan	0	2	0
C	6	0	nan	1	1
D	0	4	0	nan	5
E	2	0	0	8	nan

Рис. 3 Редуцированная изначальная матрица

Найдём изначальный вес корня: для этого сложим все минимальные элементы в строках и столбцах:

$$W_{\text{корня}} = S_{\text{строк}} + S_{\text{столбцов}} = 10$$

Для каждого нулевого элемента найдём сумму минимальных по строке и столбу:

	÷ A	÷ B	÷ C	÷ D	÷ E
A	0	0	0	3	0
B	0	0	0	0	1
C	0	1	0	0	0
D	1	0	0	0	0
E	0	0	0	0	0

Рис. 4 Нулевые элементы матрицы с коэффициентами

Максимальный коэффициент со значением 3 находится на позиции A-D, значит штраф за непосещение данного пути 3.

Исключаем путь A-D, а также обратный ему D-A закрываем *nan*-ом, т.к. возвращаться в него мы не собираемся. После выполняем редуцирование по строкам и столбцам (получим штраф 1)

	÷ A	÷ B	÷ C	÷ E
B	0	nan	0	0
C	5	0	nan	1
D	nan	4	0	5
E	1	0	0	nan

Рис. 5 Матрица с «посещенным» A-D

Выполняем «вилку» и получаем следующее дерево:

Пропуск A-D: вес корня + штраф за непосещение =  $10 + 3 = 13$

Включение A-D: вес корня + штраф за посещение =  $10 + 1 = 11$

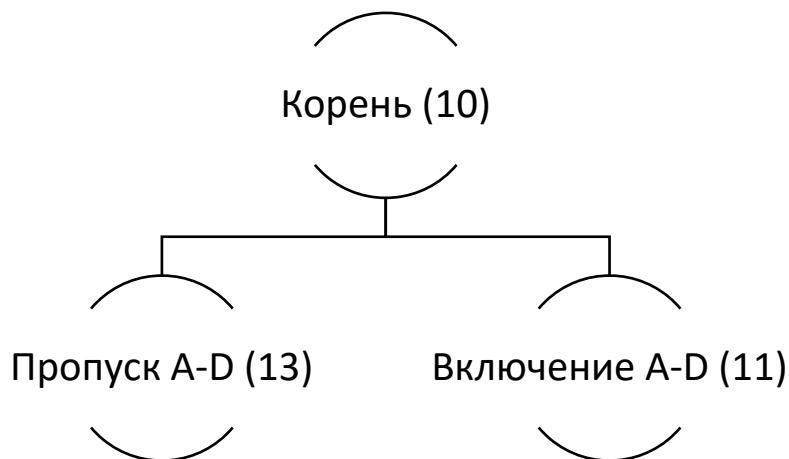


Рис. 6 Дерево поиска по окончании 1-ого тура

Минимальный вес достигается в правом «листке». С него начинаем второй тур.

	$\hat{A}$	$\hat{B}$	$\hat{C}$	$\hat{E}$
B	0	nan	0	0
C	5	0	nan	1
D	nan	4	0	5
E	1	0	0	nan

Рис. 7 Редуцированная матрица на момент начала второго тура

	$\hat{A}$	$\hat{B}$	$\hat{C}$	$\hat{E}$
B	1	0	0	1
C	0	1	0	0
D	0	0	4	0
E	0	0	0	0

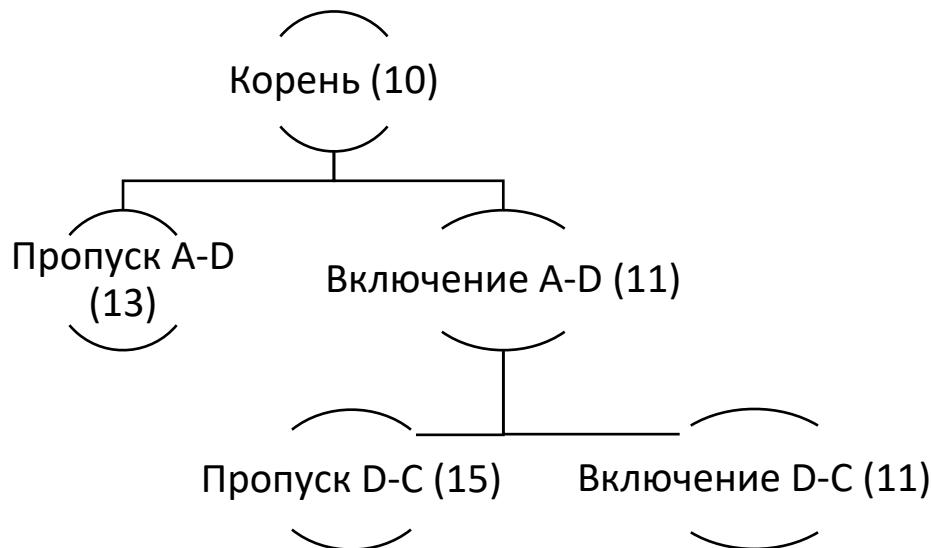
Рис. 8 Матрица коэффициентов нулевых элементов

Штраф за непосещение D-C: 4

	$\hat{A}$	$\hat{B}$	$\hat{E}$
B	0	nan	0
C	5	0	1
E	1	0	nan

Рис. 9 Матрица с посещенным D-C

Штраф за посещение D-C: 0 (т.к. уже редуцирование выполнено)



Минимальный вес достигается в правом «листке». С него начинаем третий тур.

	‡ A	‡ B	‡ E
B	0	nan	0
C	5	0	1
E	1	0	nan

Рис. 10 Редуцированная матрица на момент начала третьего тура

	‡ A	‡ B	‡ E
B	1	0	1
C	0	1	0
E	0	1	0

Рис. 11 Матрица коэффициентов нулевых элементов

Штраф за непосещение B-A: 1

Штраф за посещение B-A: 1

	‡ B	‡ E
C	nan	0
E	0	nan

Рис. 12 Матрица по окончании третьего тура

Т.к. размер матрицы 2x2 туры завершаются: два оставшихся нуля этой матрицы соответствуют двум последним ребрам, которые включаются в тур непосредственно, при этом стоимость тура не изменяется.

Итого получаем следующее дерево:

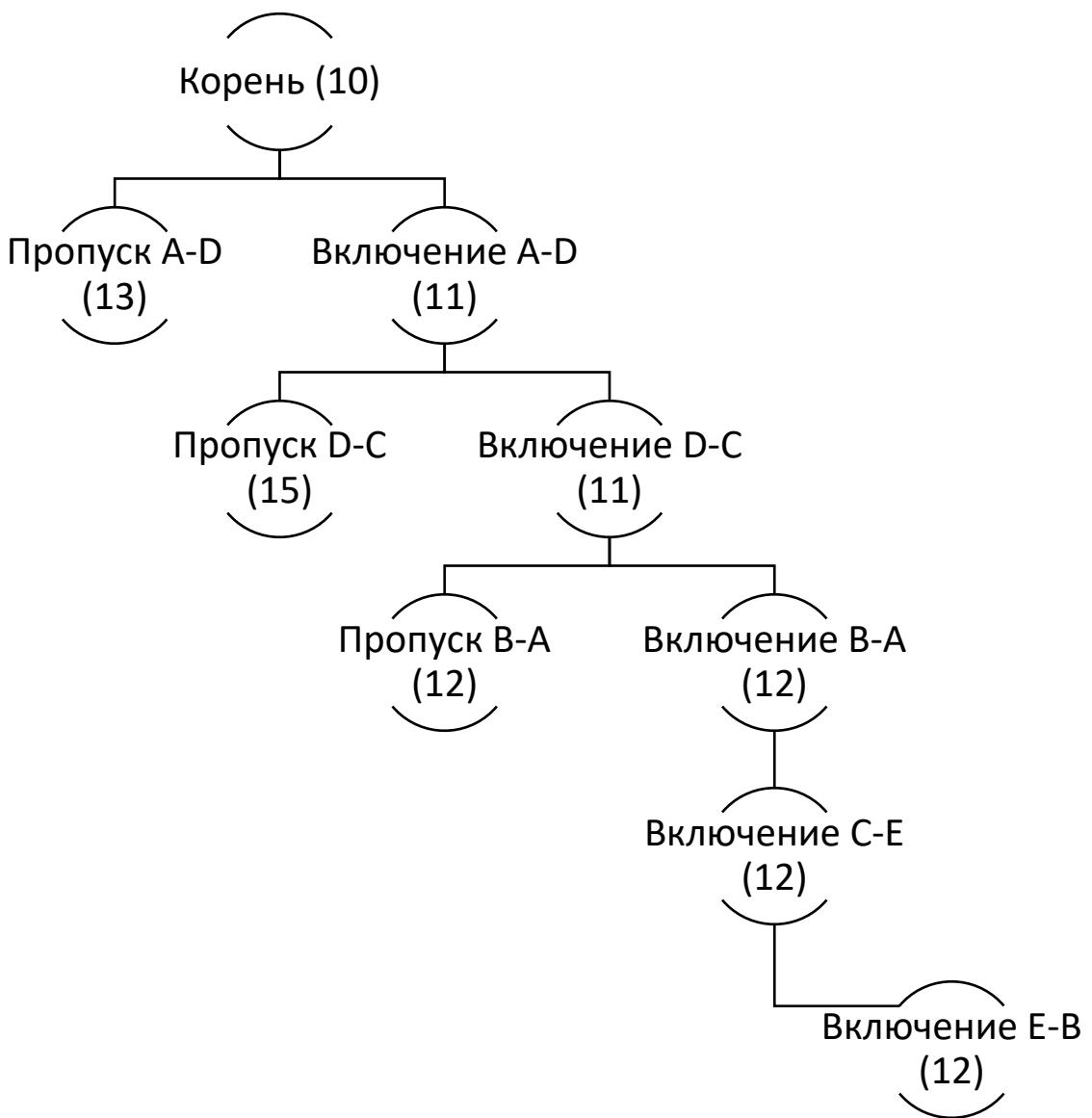


Рис. 13 Финальное дерево туров

Полученный маршрут с соединением всех включений:

$$C \rightarrow D \rightarrow A \rightarrow E \rightarrow B \rightarrow C = 12$$

Или в отсортированном варианта (начиная с A):

$$A \rightarrow E \rightarrow B \rightarrow C \rightarrow D \rightarrow A = 12$$

## Проверка онлайн решением

Проверка решением выполнялось с помощью онлайн-ресурса:  
<https://math.semestr.ru/kom/index.php>

В результате по дереву ветвлений гамильтонов цикл образуют ребра:  
(1,4), (4,3), (3,5), (5,2), (2,1),  
Длина маршрута равна  $F(M_k) = 1$

Финальный маршрут:  $1 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 1 = 12$

Что аналогично маршруту:  $A \rightarrow D \rightarrow C \rightarrow E \rightarrow B \rightarrow A = 12$

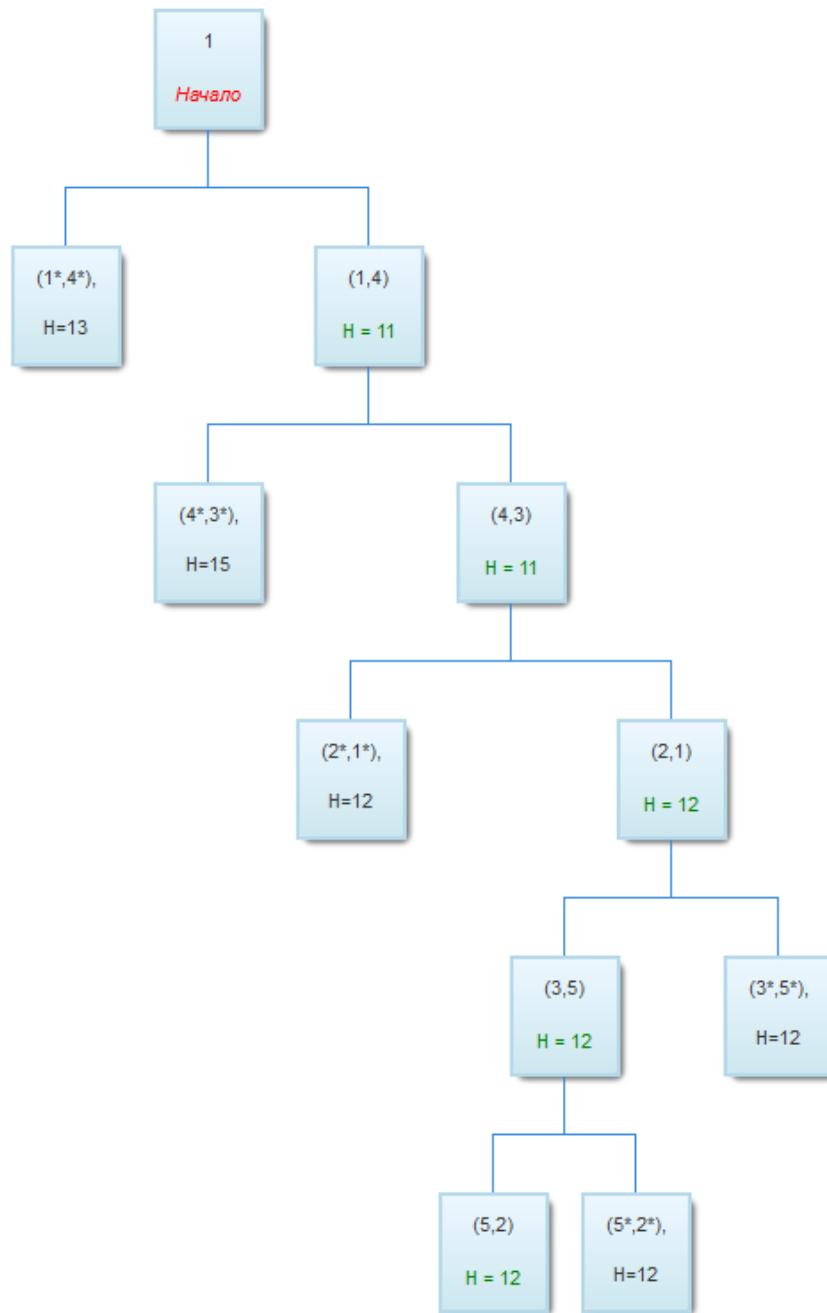


Рисунок 1 график дерева решения

## Выводы

1. Метод «Ветвей и границ» для задачи коммивояжёра рабочий и эффективный (точно эффективней перебора, поскольку на каждой итерации мы пытаемся оптимизировать выбор добавляемого элемента в тур).