**Министерство образования Российской Федерации**

# МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления Кафедра: Информационная безопасность (ИУ8)

# ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

**Лабораторная работа №4 на тему:**

«Позиционные игры. Метод обратной индукции» Вариант 9

|  |  |
| --- | --- |
| Преподаватель: | Коннова Н.С. |
| Студент: | Киселев В.А. |
| Группа: | ИУ8-104 |

Москва, 2025

# Цель работы

Изучить метод обратной индукции и его применение к решению конечных позиционных игр с полной информацией. Изучить свойства таких игр.

# Задание

1. Найти решение конечношаговой позиционной игры с полной информацией: сгенерировать и построить дерево случайной игры согласно варианту, используя метод обратной индукции, найти решение игры и путь (все пути, если он не единственный) к этому решению. Обозначить его (их) на дереве.

Исходные данные по варианту:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  варианта | Глубина  дерева | Количество  игроков | Количество  стратегий | Диапазон  выигрышей |
| 9 | 7 | 2 | 2, 2 | [0; 20] |

# Ход работы

Генерация дерева.

В связи с тем, что дерево оказалось довольно большим, оно было разделено на несколько снимков экрана, но считается за один рисунок.

Также к отчету прилагается .png файл с деревом.

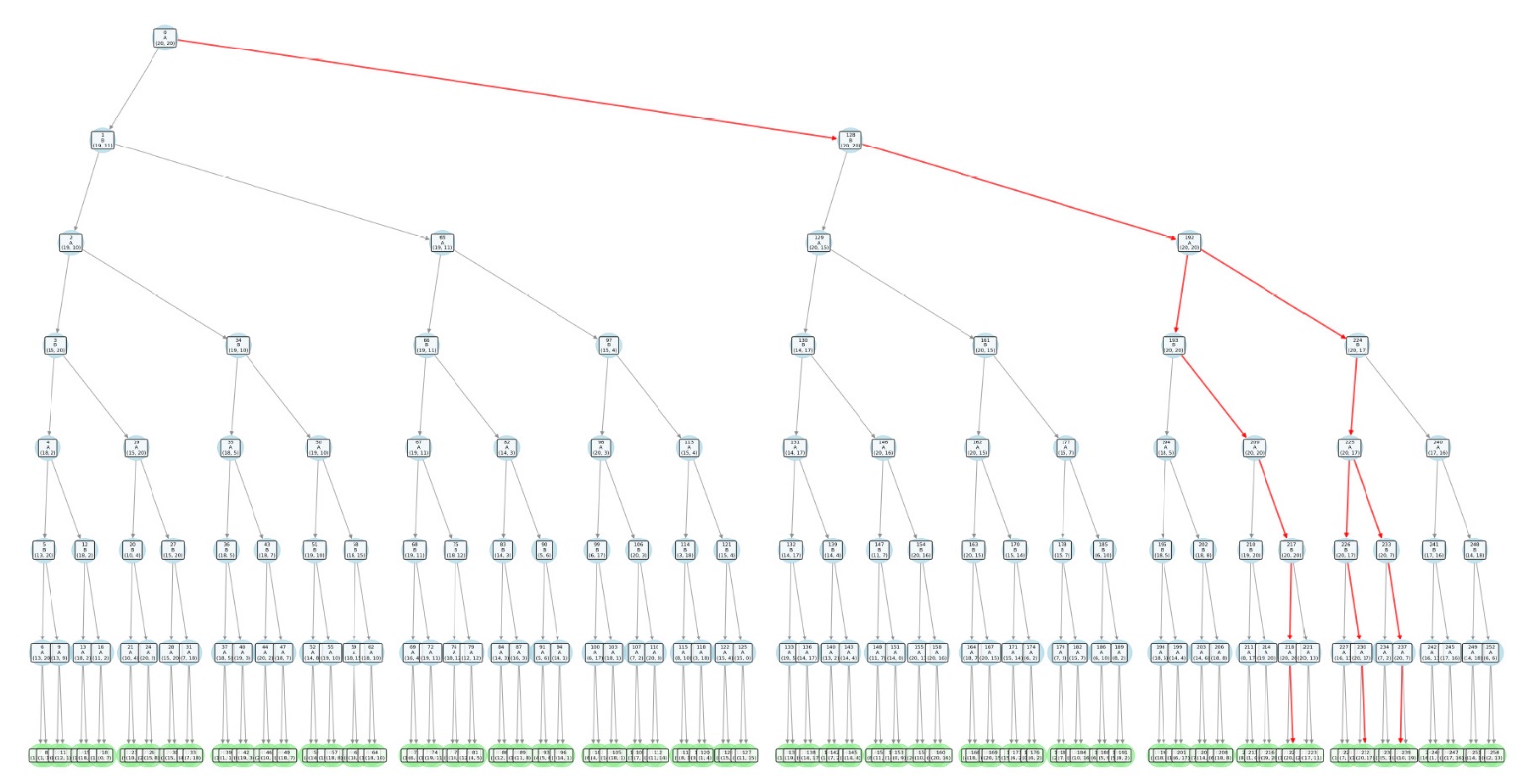


Рисунок 1 – случайно сгенерированное дерево игры

(Детальная версия дерева предоставлена в файле Дерево.pdf)

# Выводы

В ходе выполнения лабораторной работы был изучен метод обратной индукции и его применение для решения конечных позиционных игр с полной информацией.

Метод обратной индукции является эффективным инструментом для решения конечных игр с полной информацией, так как позволяет последовательно исключать заведомо неоптимальные ходы.

Всего узлов: 255

Количество терминальных узлов (листьев): 128

Оптимальные выигрыши в корневой вершине (игрок A, игрок B): (19, 16)

Найдено оптимальных путей: 3

Путь 1: [0, 1, 2, 3, 4, 5, 6, 7] -> Выигрыши в конечной вершине: (19, 16)

Путь 2: [0, 1, 65, 66, 82, 83, 84, 85] -> Выигрыши в конечной вершине: (18, 16)

Путь 3: [0, 128, 129, 161, 177, 178, 182, 183] -> Выигрыши в конечной вершине: (19, 16)

Приложение 1   
import random

from typing import List, Tuple, Optional

from reportlab.lib.pagesizes import A4, landscape

from reportlab.pdfgen import canvas

from reportlab.lib.utils import ImageReader

import matplotlib.pyplot as plt

import networkx as nx

from io import BytesIO

from reportlab.lib.units import inch

from reportlab.lib.colors import Color, red, blue, green

from reportlab.pdfbase import pdfmetrics

from reportlab.pdfbase.ttfonts import TTFont

import matplotlib

matplotlib.rcParams['font.family'] = 'DejaVu Sans'

matplotlib.rcParams['font.sans-serif'] = ['DejaVu Sans']

class GameNode:

    def \_\_init\_\_(self, node\_id: int, is\_terminal: bool = False, payoffs: Tuple[int, int] = None,

                 depth: int = 0, player: str = None):

        self.id = node\_id

        self.is\_terminal = is\_terminal

        self.payoffs = payoffs if payoffs else (0, 0)

        self.children = []

        self.depth = depth

        self.player = player

        self.optimum\_payoffs = None

        self.optimal\_children = []

        self.x = 0

        self.y = 0

def generate\_game\_tree(depth: int, num\_players: int, branches\_per\_player: List[int],

                      payoff\_range: Tuple[int, int], current\_depth: int = 0,

                      node\_counter: list = [0]) -> Optional[GameNode]:

    if current\_depth > depth:

        return None

    node\_id = node\_counter[0]

    node\_counter[0] += 1

    is\_terminal = (current\_depth == depth)

    current\_player\_index = current\_depth % num\_players

    current\_player = chr(65 + current\_player\_index)

    num\_branches = branches\_per\_player[current\_player\_index]

    node = GameNode(node\_id=node\_id, is\_terminal=is\_terminal,

                   depth=current\_depth, player=current\_player)

    if is\_terminal:

        payoff\_a = random.randint(payoff\_range[0], payoff\_range[1])

        payoff\_b = random.randint(payoff\_range[0], payoff\_range[1])

        node.payoffs = (payoff\_a, payoff\_b)

        node.optimum\_payoffs = (payoff\_a, payoff\_b)

    else:

        for \_ in range(num\_branches):

            child = generate\_game\_tree(depth, num\_players, branches\_per\_player,

                                      payoff\_range, current\_depth + 1, node\_counter)

            if child is not None:

                node.children.append(child)

    return node

def backward\_induction(node: GameNode):

    if node.is\_terminal:

        return

    for child in node.children:

        backward\_induction(child)

    optimizing\_player\_index = 0 if node.player == 'A' else 1

    best\_value = -10\*\*9

    best\_children = []

    for child in node.children:

        child\_payoff = child.optimum\_payoffs[optimizing\_player\_index]

        if child\_payoff > best\_value:

            best\_value = child\_payoff

            best\_children = [child]

        elif child\_payoff == best\_value:

            best\_children.append(child)

    if best\_children:

        node.optimal\_children = best\_children

        node.optimum\_payoffs = best\_children[0].optimum\_payoffs

def find\_optimal\_paths(root: GameNode) -> List[List[GameNode]]:

    all\_paths = []

    current\_path = []

    def dfs\_collect\_paths(current\_node):

        current\_path.append(current\_node)

        if current\_node.is\_terminal:

            all\_paths.append(current\_path[:])

        else:

            for optimal\_child in current\_node.optimal\_children:

                dfs\_collect\_paths(optimal\_child)

        current\_path.pop()

    dfs\_collect\_paths(root)

    return all\_paths

def calculate\_positions(node: GameNode, x\_spacing: float = 2.0, y\_spacing: float = 2.0, x\_offset: float = 0.0):

    if not node.children:

        node.x = x\_offset

        node.y = -node.depth \* y\_spacing

        return 1, x\_offset

    total\_width = 0

    current\_x = x\_offset

    for child in node.children:

        child\_width, new\_x = calculate\_positions(child, x\_spacing, y\_spacing, current\_x)

        total\_width += child\_width

        current\_x = new\_x + x\_spacing

    node.x = x\_offset + total\_width / 2

    node.y = -node.depth \* y\_spacing

    return total\_width, current\_x

def visualize\_tree(root: GameNode, optimal\_paths: List[List[GameNode]]):

    G = nx.DiGraph()

    pos = {}

    labels = {}

    node\_colors = []

    node\_sizes = []

    # Собираем все узлы в граф

    def add\_nodes\_edges(node):

        pos[node.id] = (node.x, node.y)

        if node.is\_terminal:

            labels[node.id] = f"{node.id}\n{node.payoffs}"

            node\_colors.append('lightgreen')

            node\_sizes.append(800)

        else:

            labels[node.id] = f"{node.id}\n{node.player}\n{node.optimum\_payoffs}"

            node\_colors.append('lightblue')

            node\_sizes.append(1000)

        for child in node.children:

            G.add\_edge(node.id, child.id)

            add\_nodes\_edges(child)

    add\_nodes\_edges(root)

    # Определяем цвет ребер (красный для оптимальных путей)

    edge\_colors = []

    edge\_widths = []

    for u, v in G.edges():

        is\_optimal = False

        for path in optimal\_paths:

            path\_ids = [n.id for n in path]

            for i in range(len(path\_ids) - 1):

                if path\_ids[i] == u and path\_ids[i+1] == v:

                    is\_optimal = True

                    break

            if is\_optimal:

                break

        edge\_colors.append('red' if is\_optimal else 'gray')

        edge\_widths.append(2.0 if is\_optimal else 1.0)

    # Создаем визуализацию

    plt.figure(figsize=(30, 15))

    nx.draw(G, pos, with\_labels=False, node\_color=node\_colors,

            edge\_color=edge\_colors, node\_size=node\_sizes, arrows=True,

            width=edge\_widths, alpha=0.7)

    # Рисуем метки с улучшенным форматированием

    for node\_id, (x, y) in pos.items():

        label = labels[node\_id]

        plt.text(x, y, label, fontsize=6, ha='center', va='center',

                 bbox=dict(boxstyle="round,pad=0.3", facecolor="white", alpha=0.8))

    plt.title("Дерево позиционной игры", fontsize=16)

    plt.axis('off')

    buf = BytesIO()

    plt.savefig(buf, format='png', dpi=300, bbox\_inches='tight')

    plt.close()

    buf.seek(0)

    return buf

def create\_pdf\_report(root: GameNode, optimal\_paths: List[List[GameNode]], filename: str = "game\_tree\_report.pdf"):

    try:

        pdfmetrics.registerFont(TTFont('DejaVuSans', 'DejaVuSans.ttf'))

        font\_name = 'DejaVuSans'

    except:

        font\_name = 'Helvetica'

    # Вычисляем позиции для визуализации

    calculate\_positions(root, x\_spacing=3.0, y\_spacing=2.0)

    # Создаем визуализацию дерева

    tree\_image = visualize\_tree(root, optimal\_paths)

    # Создаем PDF

    c = canvas.Canvas(filename, pagesize=landscape(A4))

    width, height = landscape(A4)

    # Добавляем заголовок

    c.setFont(font\_name, 16)

    c.drawString(50, height - 50, "Отчет по дереву позиционной игры")

    # Добавляем информацию о параметрах

    c.setFont(font\_name, 12)

    c.drawString(50, height - 80, f"Глубина дерева: 7")

    c.drawString(50, height - 100, f"Количество игроков: 2 (A и B)")

    c.drawString(50, height - 120, f"Количество стратегий: 2 для каждого игрока")

    c.drawString(50, height - 140, f"Диапазон выигрышей: [0, 20]")

    c.drawString(50, height - 160, f"Оптимальные выигрыши в корне: {root.optimum\_payoffs}")

    c.drawString(50, height - 180, f"Количество оптимальных путей: {len(optimal\_paths)}")

    # Добавляем изображение дерева

    img = ImageReader(tree\_image)

    img\_width, img\_height = img.getSize()

    aspect = img\_height / img\_width

    display\_width = width - 100

    display\_height = display\_width \* aspect

    if display\_height > height - 250:

        display\_height = height - 250

        display\_width = display\_height / aspect

    c.drawImage(img, (width - display\_width) / 2, height - 250 - display\_height,

                width=display\_width, height=display\_height)

    # Добавляем информацию об оптимальных путях

    y\_position = height - 250 - display\_height - 30

    c.setFont(font\_name, 12)

    c.drawString(50, y\_position, "Оптимальные пути:")

    c.setFont(font\_name, 10)

    y\_position -= 20

    for i, path in enumerate(optimal\_paths):

        if y\_position < 50:

            c.showPage()

            y\_position = height - 50

            c.setFont(font\_name, 10)

        path\_ids = [f"{n.id}" for n in path]

        terminal\_payoffs = path[-1].payoffs

        path\_str = " → ".join(path\_ids)

        c.drawString(70, y\_position, f"Путь {i+1}: {path\_str} → Выигрыши: {terminal\_payoffs}")

        y\_position -= 15

    c.showPage()

    c.setFont(font\_name, 14)

    c.drawString(50, height - 50, "Легенда:")

    c.setFont(font\_name, 12)

    c.drawString(50, height - 80, "Синие узлы: нетерминальные вершины (с указанием игрока и оптимальных выигрышей)")

    c.drawString(50, height - 100, "Зеленые узлы: терминальные вершины (с указанием выигрышей)")

    c.drawString(50, height - 120, "Красные ребра: оптимальные пути")

    c.drawString(50, height - 140, "Серые ребра: неоптимальные пути")

    # Сохраняем PDF

    c.save()

def main():

    DEPTH = 7

    NUM\_PLAYERS = 2

    BRANCHES\_PER\_PLAYER = [2, 2]

    PAYOFF\_RANGE = (0, 20)

    node\_counter = [0]

    root\_node = generate\_game\_tree(

        depth=DEPTH,

        num\_players=NUM\_PLAYERS,

        branches\_per\_player=BRANCHES\_PER\_PLAYER,

        payoff\_range=PAYOFF\_RANGE,

        current\_depth=0,

        node\_counter=node\_counter

    )

    print(f"Всего узлов: {node\_counter[0]}")

    print(f"Количество терминальных узлов: {2\*\*DEPTH}")

    backward\_induction(root\_node)

    print(f"\nОптимальные выигрыши в корневой вершине (игрок A, игрок B): {root\_node.optimum\_payoffs}")

    print("\nПоиск всех оптимальных путей...")

    optimal\_paths = find\_optimal\_paths(root\_node)

    print(f"Найдено оптимальных путей: {len(optimal\_paths)}")

    create\_pdf\_report(root\_node, optimal\_paths, "game\_tree\_report.pdf")

    print("PDF-отчет создан: game\_tree\_report.pdf")

if \_\_name\_\_ == "\_\_main\_\_":

    main()