**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 3

|  |  |
| --- | --- |
| Преподаватель: | Коннова Н.С. |
| Студент: | Валиев В.С. |
| 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:

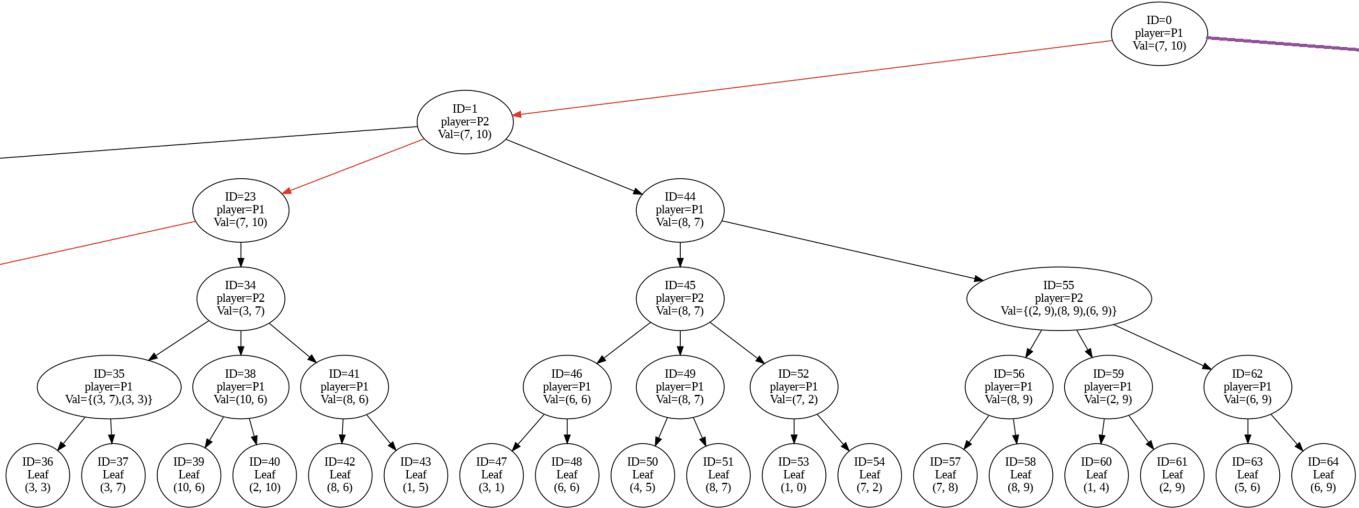
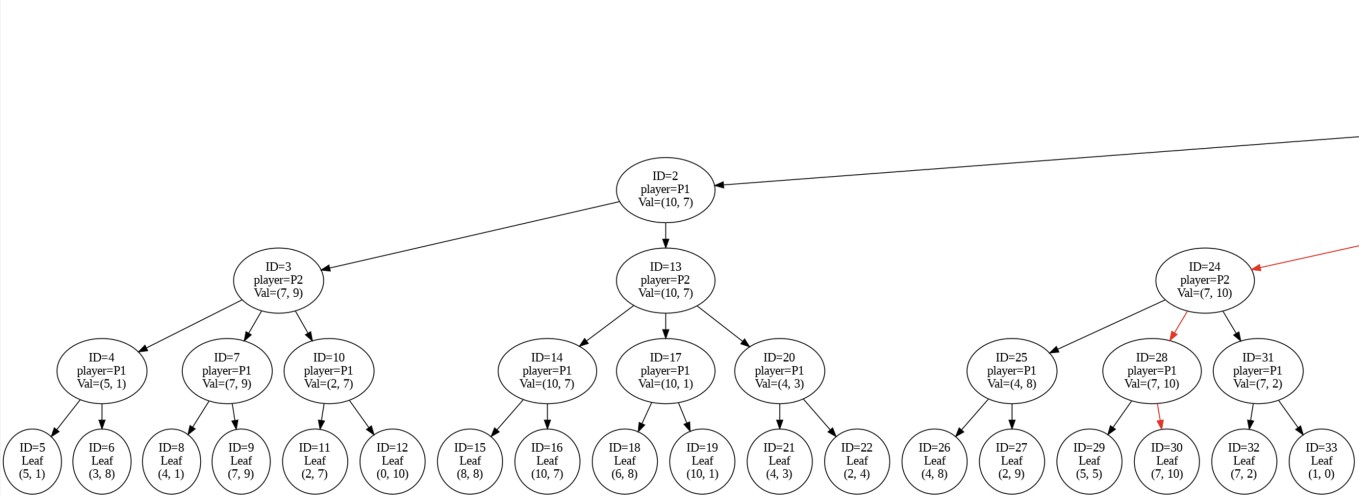
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  варианта | Глубина  дерева | Количество  игроков | Количество  стратегий | Диапазон  выигрышей |
| 3 | 6 | 2 | 2, 3 | [0; 10] |

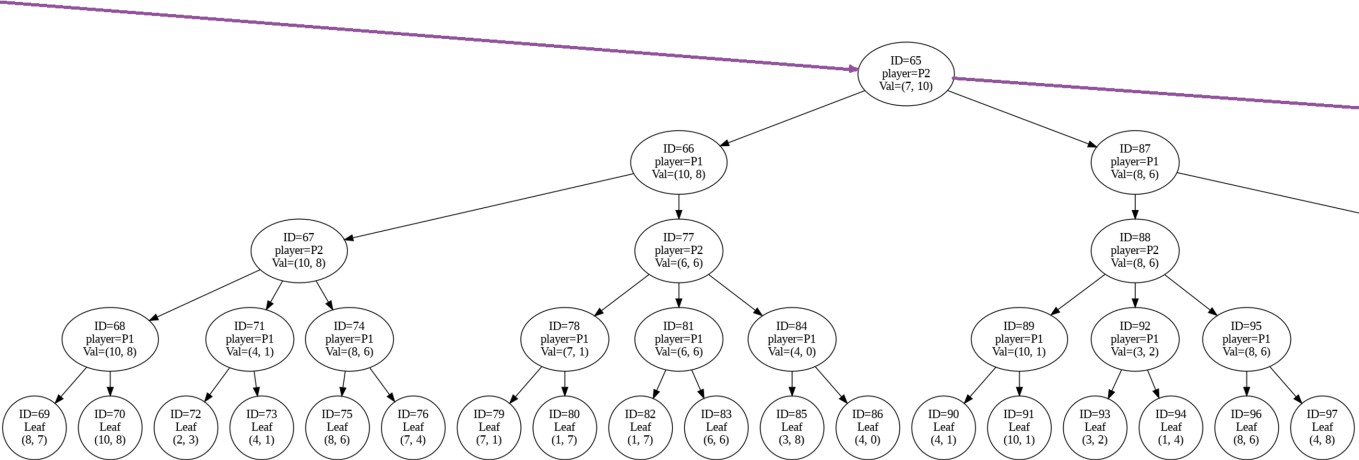
# Ход работы

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.





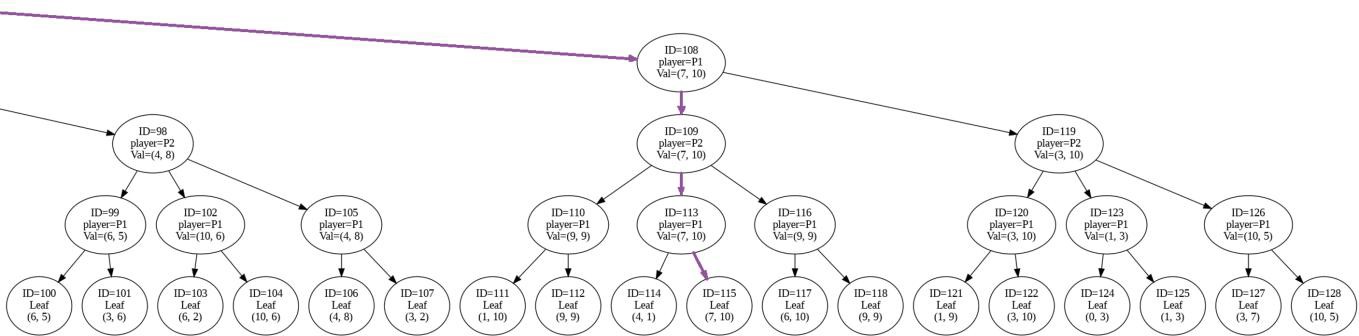


Рисунок 1 – случайно сгенерированное дерево игры по частям (слева направо)

# 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.

21.05.2025, 01:51 LR4.ipynb - Colab

!pip install anytree graphviz

Requirement already satisfied: anytree in /usr/local/lib/python3.11/dist-packages (2.13.0) Requirement already satisfied: graphviz in /usr/local/lib/python3.ll/dist-packages (0.20.3)

import random

from anytree import Node, NodeMixin, RenderTree from anytree.exporter import DotExporter

from graphviz import Digraph

class Node:

\_id\_counter = 0

def \_init\_(self, level): self.id= Node.\_id\_counter Node.\_id\_counter += 1

self.level= level self.is\_terminal = False self.player= None self.children=[]

self.payoff= None self.best\_value [] self.best\_moves = []

class GameTree:

def \_init\_(self, max\_level=S, players=None, strategies=None, payoff\_range=(0,10)): if players is None:

players= ["Pl","P2"] if strategies is None:

strategies {"Pl":2,"P2":3}

self.max\_level max\_level self.players= players self.strategies= strategies self.payoff\_range = payoff\_range

self.player\_colors = { "Pl":"red",

"P2":"red",

self.root= None

def get\_player\_for\_level(self, level):

return self.players[(level - 1) % len(self.players)]

def build\_tree\_fixed(self, level=l): node= Node(level)

if level== self.max\_level + 1: node.is\_terminal = True

ul = random.randint(self.payoff\_range[0], self.payoff\_range[l]) u2 = random.randint(self.payoff\_range[0], self.payoff\_range[l]) node.payoff= (ul,u2)

node.best\_value = [node.payoff] return node

else:

player= self.get\_player\_for\_level(level) node.player= player

bf= self.strategies[player]

for\_ in range(bf):

child= self.build\_tree\_fixed(level+l) node.children.append(child)

return node

def collect\_nodes\_by\_level(self, node, dct): dct.setdefault(node.level, []).append(node) for c in node.children:

self.collect\_nodes\_by\_level(c, dct)

def iterative\_backward\_induction(self): levels={} self.collect\_nodes\_by\_level(self.root, levels)

chosen\_edges = set()

for lvl in range(self.max\_level, 0, -1): if lvl not in levels:

21.05.2025, 01:51

**con-c1nue**

LR4.ipynb - Colab

for node in levels[lvl]: if node.is\_terminal:

continue else:

player= node.player

p\_index = self.players.index(player)

max\_val = node.children[0].best\_value[0][p\_index] for c in node.children:

val= c.best\_value[0][p\_index] if val>max\_val:

max\_val = val best\_ids = [] best\_vals = []

for i,c in enumerate(node.children): cv = c.best\_value[0]

if cv[p\_index] == max\_val: best\_ids.append(i) best\_vals.append(cv)

best\_vals = list(set(best\_vals)) node.best\_value best\_vals node.best\_moves = best\_ids

# Ao6asnReM s chosen\_edges for i in best\_ids:

chosen\_edges.add((node.id, node.children[i].id, player)) self.visualize\_tree\_all(step=f"level\_{lvl}", colored\_edges=chosen\_edges)

return chosen\_edges

def visualize\_tree\_all(self, step="final", colored\_edges=None): dot= Digraph(comment=f"Game Tree Step:{step}")

def label\_for(n: Node): if n.is\_terminal:

return f"ID={n.id}\nleaf\n{n.payoff}" else:

if len(n.best\_value)==l:

val\_str str(n.best\_value[0]) else:

val\_str "{"+",".join(str(v) for v in n.best\_value)+"}" return f"ID={n.id}\nplayer={n.player}\nVal={val\_str}"

def add\_nodes(n: Node): dot.node(str(n.id), label\_for(n)) for c inn.children:

add\_nodes(c)

def add\_edges(n: Node): if not n.is\_terminal:

for i,c in enumerate(n.children): triple= (n.id, c.id, n.player) color= "black"

if colored\_edges and triple in colored\_edges: color= self.player\_colors.get(n.player,"black")

dot.edge(str(n.id), str(c.id), color=color) for c inn.children:

add\_edges(c)

add\_nodes(self.root) add\_edges(self.root)

fname = f"game\_tree\_{step}" dot.render(fname, format="png", cleanup=True)

def visualize\_tree\_final(self, step="final\_all\_paths"): dot= Digraph(comment=f"Game Tree Step:{step}")

dot.graph\_attr["rankdir"] dot.graph\_attr["nodesep"] dot.graph\_attr["ranksep"]

"TB"

"0.2"

"0.4"

colored\_edges = set()

def gather\_best\_moves(n: Node): if not n.is\_terminal:

for i in n.best\_moves: child= n.children[i]

colored\_edges.add((n.id, child.id, n.player))

gather\_best\_moves(child)

gather\_best\_moves(self.root)

21.05.2025, 01:51 LR4.ipynb - Colab

def label\_for(n: Node): if n.is\_terminal:

return f"ID={n.id}\nleaf\n{n.payoff}" else:

if len(n.best\_value)==l:

val\_str str(n.best\_value[0]) else:

val\_str "{"+",".join(str(v) for v in n.best\_value)+"}" return f"ID={n.id}\nplayer={n.player}\nVal={val\_str}"

def add\_nodes(n: Node): dot.node(str(n.id), label\_for(n)) for c inn.children:

add\_nodes(c)

def add\_edges(n: Node): if not n.is\_terminal:

for i,child in enumerate(n.children): triple= (n.id, child.id, n.player) color= "black"

if triple in colored\_edges:

color= self.player\_colors.get(n.player,"black") dot.edge(str(n.id), str(child.id), color=color)

for c inn.children: add\_edges(c)

add\_nodes(self.root) add\_edges(self.root)

fname = f"game\_tree\_{step}" dot.render(fname, format="png", cleanup=True)

game= GameTree()

game.root= game.build\_tree\_fixed() game.iterative\_backward\_induction() game.visualize\_tree\_final(step="final")