Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа №4 на тему "Позиционные игры. Метод обратной индукции.

по дисциплине «Теория игр и исследование операций»

Вариант 12

Студент ИУ8-104 (Группа) Мильченко И. Д. (И. О. Фамилия) _____ (Подпись, дата)

Преподаватель Коннова Н. С. (И. О. Фамилия) _____ (Подпись, дата)

Москва, 2025 г.

ЦЕЛЬ РАБОТЫ

Изучить метод обратной индукции и его при- менение к решению конечных позиционных игр с полной информацией. Изучить свойства решений таких игр.

Задание

Найти решение конечношаговой позиционной игры с полной информацией. Для этого 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 их на дереве.

Данные о генерации дерева по заданному варианту представлены в таблице 1.

Таблица 1 – Значения по варианту

Номер варианта	Глубина дерева	Количество игроков	Количество стратегий	Диапазон выигрышей
12	7	2	2, 3	$[-5,\ 25]$

ХОД РАБОТЫ

В проекте был написан класс для решения позиционных игр методом обратной индукции. Также был реализован алгоритм построения деревьев с большой глубиной при помощи языка для отрисовки графов Dot.

Пример запуска программы:

```
go run cmd/lw4/main.go
```

Решим позционную игру методом обратной индукции.

1 Метод обратной индукции

В динамических играх с полной и совершенной информацией удобно решать игру методом обратной индукции. В методе обратной индукции рассматриваются все последние вершины игры, в которых один из игроков делает выбор, исходя из его рациональности. Далее процесс повторяется для всех предшествующих вершин, пока не дойдет до начальной вершины.

Для решения сгенерируем дерево, согласно таблице 1. Так как дерево слишком большое, далее будут представлены его части, в которых берутся выигрышные стратегии от листовых узлов до корня.

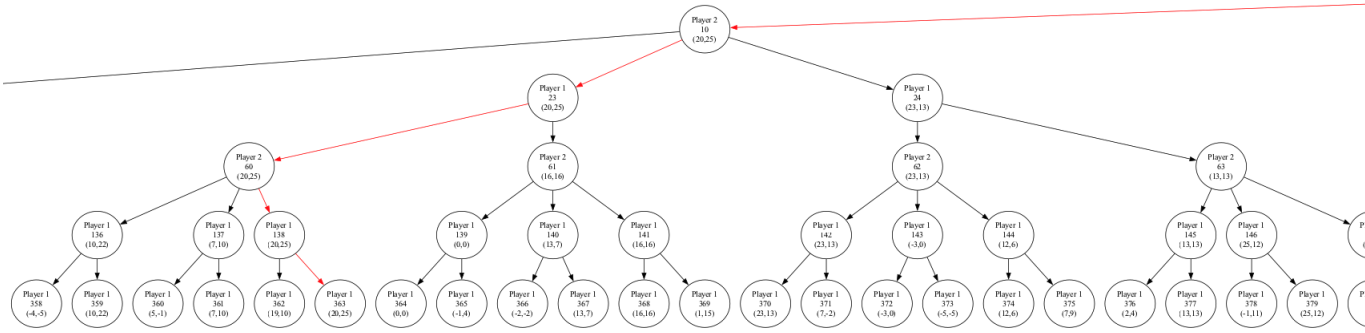

Рисунок 1 – Нижний уровень дерева

На рисунке 1 покрашены красным цветом ребра графа, по которому идет выигрышная стратегия. Например, между 362 и 363 узлом выбирается 363, так как при ходе первого игрока кортеж (20, 25) больше (19, 10).

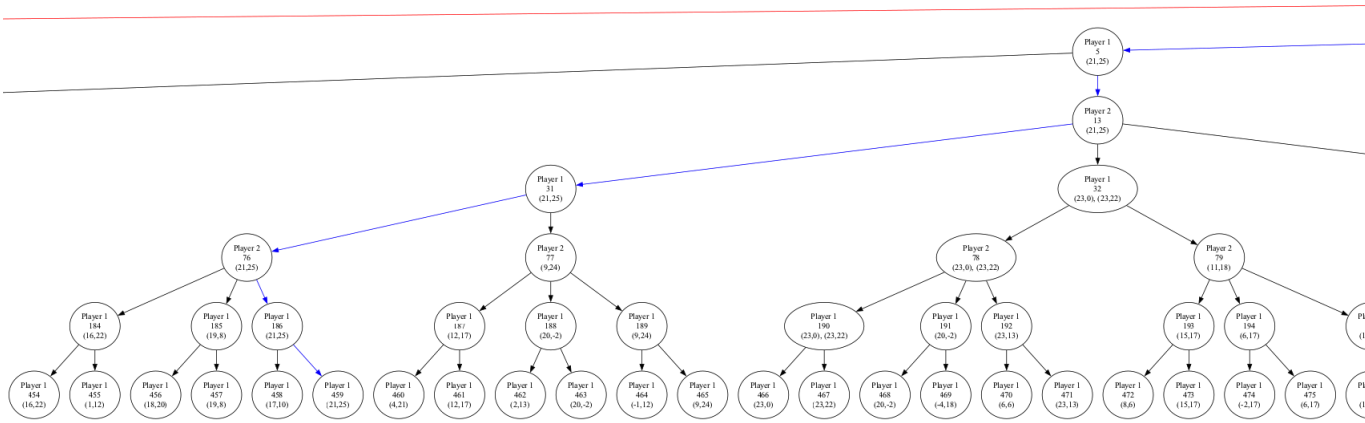

Рисунок 2 – Промежуточная индукция

На рисунке 2 покрашены синим цветом ребра гарафа другой равновероятной стратегии. Проталкивание кортежа вверх производится аналогичным образом.

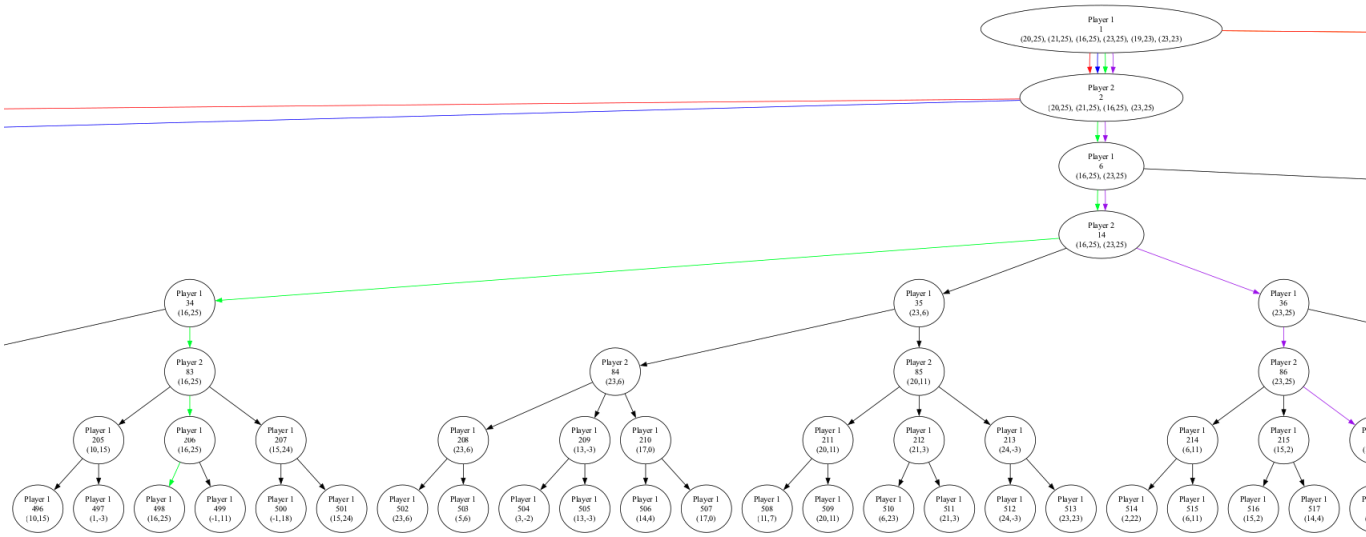

Рисунок 3 – Итоговая оптимальная стратегия

На рисунке 3 показаны, как игроки делают выбор на основе уже агрегированных выигрышей, полученных из нижележащих уровней. Таким образом в 14 узле выбираются все кортежи из 34 и 36 узлов, потому что при ходе второго игрока в узлах одинаковые элементы – 25.

Рисунок 4 – Второй дочерний узел корня

Итоговая оптимальная стратегия состоит из всех стратегий узла 2 (см. рисунок 3) и узла 3, который показан на рисунке 4 за счет того, что при ходе первого игрока максимальный элемент 23 находится как во 2, так и в 3 узлах.

Итого, все пути, разукрашенные в цвета, отличные от черного и дошедшие до корня, являются решениями игры. Выигрышими являются кортежи (20, 25), (21, 25), (16, 25), (23, 25), (19, 23), (23, 23).

2 Решение для дерева глубины 4

Для полноты картины решим игры для дерева глубины 4, которое представлено на рисунке. Рассуждения аналогичны тем, что приведены в предыдущем пункте.

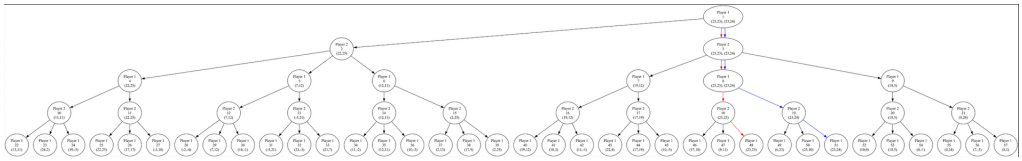

Рисунок 5 – Метод обратной индукции для дерева глубины 4

В этой игры выигрышами являются кортежи (23, 23), (23, 24).

ЗАКЛЮЧЕНИЕ

В данной работе был исследован метод обратной индукции на примере задач, представленных в виде двух деревьев решений с различной глубиной. Вначале рассмотрено дерево по варианту глубиной 7, что позволило детально проанализировать процесс принятия решений в отдельных ключевых узлах при наличии большого количества вариантов и высокой структурной сложности. Такой поэтапный разбор продемонстрировал, как, начиная с конечных результатов, можно последовательно «поднимать» оптимальные выигрыши на более высокие уровни дерева, обеспечивая выбор оптимальной стратегии на каждом этапе.

Далее, для полноты картины, было взято дерево с глубиной 4, чтобы отобразить работу метода на одной картинке сразу, где будут показаны выигрышные стратегии на каждом шаге одновременно.

ПРИЛОЖЕНИЕ А

Класс реализация позиционных игр

Листинг А.1 – game.go

```
package positionalgames

import (
  "errors"
  "math/rand"
  "slices"
)

type Game struct {
  r          *node
  curPlayer int
  playerNum int
  leafLvl    []*node
  bestStrat [][]*node
}

func NewGame(depth, playerNum int, strategyNum []int, winRange [2]
   int) (*Game, error) {
  if len(strategyNum) != playerNum {
    return nil, errors.New("length of startegies number should be
       the same as palyers number")
  }

  idCounter := 1

  g := &Game{
    r: &node{
      id: idCounter,
    },
    playerNum: playerNum,
    bestStrat: make([][]*node, 0),
  }

  idCounter++

  curLvl := []*node{g.r}
```

```
  for i := range depth {
    g.curPlayer = i % playerNum
    childrenLen := strategyNum[i%len(strategyNum)]

    newLvl := make([]*node, 0, len(curLvl))

    for _, child := range curLvl {
      child.children = make([]*node, childrenLen)
      child.playerNum = g.curPlayer

      for k := range child.children {
        child.children[k] = &node{
          id:     idCounter,
          parent: child,
        }

        idCounter++
      }

      newLvl = append(newLvl, child.children...)
    }

    curLvl = newLvl
  }

  for _, node := range curLvl {
    node.v = make([]tuple, 1)
    node.v[0] = make([]int, playerNum)

    for i := range node.v[0] {
      node.v[0][i] = rand.Intn(winRange[1]-winRange[0]+1) +
         winRange[0]
    }
  }

  g.leafLvl = curLvl

  return g, nil
}

func (g *Game) Solve() []tuple {
  curLvl := g.getLvl(g.leafLvl)
```

```
  for {
    for _, n := range curLvl {
      res := g.compare(n.children, g.curPlayer)
      for _, v := range res {
        n.v = append(n.v, v.v...)
      }
    }

    if len(curLvl) == 1 {
      break
    }

    g.curPlayer = (g.curPlayer + 1) % g.playerNum
    curLvl = g.getLvl(curLvl)
  }

  return g.r.v
}

func (g *Game) getLvl(children []*node) []*node {
  lvl := make([]*node, 0)

  for _, c := range children {
    if !slices.Contains(lvl, c.parent) {
      lvl = append(lvl, c.parent)
    }
  }

  return lvl
}

func (g *Game) compare(nodes []*node, player int) []*node {
  type val struct {
    v    tuple
    node *node
  }

  maxS := make([]val, 0, len(nodes))

  // From every node extract max tuple.
  for _, n := range nodes {
```

```
    maxS = append(maxS, val{
      v: slices.MaxFunc(n.v, func(a, b tuple) int {
        return a[player] - b[player]
      }),
      node: n,
    })
  }

  // Check for max tuple.
  m := slices.MaxFunc(maxS, func(a, b val) int {
    return a.v[player] - b.v[player]
  })

  res := make([]*node, 0)

  // Check that there is no tuple with max value.
  for _, v := range maxS {
    if v.v[player] == m.v[player] {
      res = append(res, v.node)
    }
  }

  return res
}
```

Листинг A.2 – node.go

```
package positionalgames

type node struct {
  id int

  v        []tuple
  parent   *node
  children []*node

  playerNum int

  isBestChildren []*node
}

type tuple []int
```

Листинг A.3 – solution.go

```
package positionalgames

type Solution struct {
  Evaluations [][]int
}
```

Листинг A.4 – visualize.go

```
package positionalgames

import (
  "fmt"
  "slices"
  "strconv"

  "github.com/awalterschulze/gographviz"
)

type colorEdges map[string][]string

func (g *Game) Dot() string {
  dotAst, _ := gographviz.ParseString("digraph G {}")
  graph := gographviz.NewGraph()
  _ = gographviz.Analyse(dotAst, graph)

  edges := make(colorEdges)

  colorsList := []string{"red", "blue", "green", "purple", "orange"
     }

  for i, payoff := range g.r.v {
    col := colorsList[i%len(colorsList)]
    highlightPath(g.r, payoff, col, edges)
  }

  addNodesAndEdges(graph, g.r, edges)
  return graph.String()
}

func highlightPath(n *node, payoff tuple, color string, edges
   colorEdges) {
```

```
  for _, child := range n.children {
    if containsPayoff(child.v, payoff) {
      key := fmt.Sprintf("%d->%d", n.id, child.id)
      edges[key] = append(edges[key], color)
      highlightPath(child, payoff, color, edges)
    }
  }
}

func addNodesAndEdges(gv *gographviz.Graph, n *node, edgeColors
   colorEdges) {
  label := fmt.Sprintf("Player %d\n%d", n.playerNum+1, n.id)
  if len(n.v) > 0 {
    label += "\\n("
    for i, payoff := range n.v {
      label += tupleToString(payoff)
      if i < len(n.v)-1 {
        label += "), ("
      }
    }
    label += ")"
  }
  quotedLabel := fmt.Sprintf("\"%s\"", label)
  nodeID := strconv.Itoa(n.id)

  gv.AddNode("G", nodeID, map[string]string{"label": quotedLabel})

  for _, child := range n.children {
    key := fmt.Sprintf("%d->%d", n.id, child.id)
    childID := strconv.Itoa(child.id)

    if colorList, ok := edgeColors[key]; !ok || len(colorList) == 0
       {
      gv.AddEdge(nodeID, childID, true, nil)
    } else {
      for _, col := range colorList {
        attrs := map[string]string{
          "color":    col,
          "penwidth": "1",
        }

        gv.AddEdge(nodeID, childID, true, attrs)
```

```
      }
    }

    addNodesAndEdges(gv, child, edgeColors)
  }
}

func containsPayoff(v []tuple, payoff tuple) bool {
  for _, x := range v {
    if slices.Equal(x, payoff) {
      return true
    }
  }
  return false
}

func tupleToString(t tuple) string {
  s := ""
  for i, val := range t {
    s += strconv.Itoa(val)
    if i < len(t)-1 {
      s += ","
    }
  }
  return s
}
```

ПРИЛОЖЕНИЕ Б

Точка входа в программу

Листинг Б.5 – main.go

```
package main

import (
  "fmt"
  "log"
  "os"

  positionalgames "github.com/themilchenko/game_theory/internal/
     positional_games"
)

var (
  depth       = 7
  players     = 2
  strategyNum = []int{2, 3}
  winRange    = [2]int{-5, 25}
)

func main() {
  g, err := positionalgames.NewGame(depth, players, strategyNum,
     winRange)
  if err != nil {
    log.Fatal(err)
  }

  fmt.Println(g.Solve())

  f, err := os.Create("./artifacts/lw4/tree.dot")
  if err != nil {
    log.Fatal(err)
  }
  defer f.Close()

  fmt.Fprintln(f, g.Dot())
}
```