Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Кафедра вычислительной математики и механики

**Лабораторная работа № 4**

**по теме: «Модель принятия коллективных решений»**

Выполнил

студент группы ИСТ-19-2б

Репин М.Е.

Проверил

ассистент кафедры ВММБ

Нетбай Г.В.

Пермь, 2022

# Задача

Реализовать модели принятия коллективных решений. Из предоставленных вариантов были выбраны два метода:

1. Относительного большинства,
2. Модель Борда.

# Метод относительного большинства: теория

Для реализации метода относительного большинства требуется следовать следующим положениям:

1. Каждый избиратель отдает свой голос одному кандидату – наилучшему в его индивидуальном предпочтении,
2. Коллективная ценность кандидата измеряется числом полученных им голосов,
3. Победителем признается кандидат, имеющий наибольшую коллективную ценность, то есть получивший наибольшее число голосов

# Метод относительного большинства: алгоритм работы

Для реализации метода необходимо знать N – количество голосующих, это число мы спрашиваем у пользователя.

Далее, программа задаёт вопрос о предпочтении пользователя N раз.

По завершению ответа на вопросы выводится статистика, за какой вариант проголосовали чаще всего.

# Метод относительного большинства: примеры запуска программы

|  |  |
| --- | --- |
|  |  |

# Модель Борда: теория

В данной модели коллективного принятия решения выбирающие отдают свой голос не единственному подходящему, по их мнению, кандидату, а расставляют приоритеты выбора. То есть голосующий расставляет все варианты в соответствии со своими предпочтениями.

После этого производится подсчёт баллов для каждого варианта следующим образом, если было всего три варианта. Каждому варианту даётся три балла за каждый голос, поставивший этот вариант на первое место, два балла за каждый голос, поставивший на второе место, и один балл за каждый голос, поставивший на третье место.

В итоге, у каждого варианта имеется своё количество набранных баллов, в соответствии с которыми принимается решение о наиболее подходящем кандидате.

# Модель Борда: алгоритм работы

Сначала программа просит перечислить кандидатов через запятую.

После этого программа выводит на экран все возможные комбинации расстановки кандидатов по приоритету с вопрос, сколько голосов отдали своё предпочтение данной расстановке.

По завершению ввода голосов для каждого варианта считаются баллы в соответствии с формулой, описанной в теории выше. Пользователю выводится наиболее подходящий вариант, список остальных кандидатов, и общая сумма баллов.

# Модель Борда: примеры запуска программы

|  |  |
| --- | --- |
|  |  |

# Листинг кода main.py

import sys

import pathlib

import funcs

folder = str(pathlib.Path(\_\_file\_\_).parent.resolve())

match sys.argv[1].lower():

case '-obvious':

if len(sys.argv) == 3:

votes = funcs.ObviousWinnerReadsJsonFile(folder+"/"+sys.argv[2])

else:

votes = funcs.ObviousWinnerReadsUserInput()

funcs.PrintVerdict(votes)

case '-borda':

if len(sys.argv) == 3:

table = funcs.BordaReadsJsonFile(folder+"/"+sys.argv[2])

else:

table = funcs.BordaReadsUserInput()

points = funcs.BordaCountsPoints(table)

funcs.PrintVerdict(points)

case \_:

funcs.PrintHelp(); print(); exit()

print() *# last blank line*

# Листинг кода funcs.py

import json

from itertools import permutations

**def** ObviousWinnerReadsJsonFile(filePathVotes: str) -> dict[str, int]:

*# reading options*

with open(filePathVotes, "r", encoding='utf-8') as reading:

raw\_data = json.load(reading)

*# collecting options*

options = []

for option in raw\_data.values():

if option not in options:

options += [option]

*# initiating votes dict*

votes = {option: 0 for option in options}

for option in raw\_data.values():

votes[option] += 1

*# sorting*

votes = dict(sorted(votes.items(), key=**lambda** item: item[1], reverse=True))

return votes

**def** ObviousWinnerReadsUserInput() -> dict[str, int]:

chosen = []

*# listening to user input*

n = int(input("\nNumber of voters: "))

for \_ in range(n):

user\_input = input("What do you choose: ")

chosen += [user\_input]

*# initiating votes*

votes = {}

for option in chosen:

if option not in votes:

votes[option] = 0

*# counting votes*

for option in chosen:

votes[option] += 1

votes = dict(sorted(votes.items(), key=**lambda** item: item[1], reverse=True))

return votes

**def** BordaReadsJsonFile(filePathVotes: str) -> list[dict[str, any]]:

'''

Returns: table = [{"options": [one, two, three], "votes": 5}, etc...]

'''

with open(filePathVotes, "r", encoding='utf-8') as reading:

table = json.load(reading)

return table

**def** BordaReadsUserInput() -> list[dict[str, any]]:

'''

Returns: table = [{"options": [one, two, three], "votes": 5}, etc...]

'''

*# getting possible options*

user\_input = input("\nGive me a list of options (comma separated): ")

options = [option for option in user\_input.split(', ')]

*# generating possible combinations*

combinations = list(permutations(options))

*# initinating dict of votes*

votes = []

for options in combinations:

prompt = **f**"How many people agree with {options}: "

value = int(input(prompt))

votes += [value]

*# assemblying table*

table = []

for i in range(len(votes)):

line = {"options": combinations[i], "votes": votes[i]}

table += [line]

return table

**def** BordaCountsPoints(table: list[dict[str, any]]) -> dict[str, int]:

options = table[0]["options"]

*# calculating points*

points = {option: 0 for option in options}

for option in options:

for line in table:

value = (len(options) - line["options"].index(option)) \* line["votes"]

points[option] += value

*# sorting*

points = dict(sorted(points.items(), key=**lambda** item: item[1], reverse=True))

return points

**def** PrintVerdict(score: dict[str, int]) -> None:

*# defining core parameters*

options = \_options(score)

top\_options = \_top\_options(score)

*# greeting message*

if len(top\_options) == 0:

raise Exception

elif len(top\_options) == 1:

print("\nTop option: ")

else:

print(**f**"\nTop options: ")

*# printing top options*

count = 1

for option in top\_options:

print(**f**"{count}. {option} with score {score[option]}")

count += 1

if len(score) - count == -1:

return

*# in case there are unnamed options left*

print("\nThe remaining options are: ")

for option, votes in score.items():

if option not in top\_options:

print(**f**"{count}. {option} with score {votes}")

count += 1

return

**def** PrintHelp():

print("""\nProgram takes parameters, try running:

- python main.py -obvious

- python main.py -obvious voted.json

- python main.py -borda

- python main.py -borda voted3.json""")

**def** \_options(score: dict[str, int]) -> list[str]:

*# collecting possible options*

options = list(score.keys())

return options

**def** \_top\_options(score: dict[str, int]) -> list[str]:

*# finding highest score (votes, points)*

highest\_score = 0

for \_, value in score.items():

if value > highest\_score:

highest\_score = value

*# collecting options with highest score*

top\_options = []

for option, score in score.items():

if score == highest\_score:

top\_options += [option]

return top\_options