Решим задачу динамического программирования методом ветвей и границ.

--------------------------------------------------------------------------------

С помощью симплекс-метода

Задача

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи имеем следующее решение

x₁ = 5.091, x₂ = 0, x₃ = 3.364

F(x) = 2.5 \* 5.091 + 4 \* 0 + 4.2 \* 3.364 = 26.855

Получили дробные решения.

Возьмем переменную x₁ = 5.0909091.

Разобьём задачу на две подзадачи, где x₁ ≤ 5 и x₁ ≥ 6.

--------------------------------------------------------------------------------

Задача 1

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≤ 5,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 1 имеем следующее решение

x₁ = 5, x₂ = 0.083, x₃ = 3.333

F(x) = 2.5 \* 5 + 4 \* 0.083 + 4.2 \* 3.333 = 26.833

Получили дробные решения.

Возьмем переменную x₂ = 0.083333333.

Разобьём задачу на две подзадачи, где x₂ ≤ 0 и x₂ ≥ 1.

--------------------------------------------------------------------------------

Задача 2

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≥ 6,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 2 имеем следующее решение

x₁ = 6, x₂ = 0, x₃ = 2

F(x) = 2.5 \* 6 + 4 \* 0 + 4.2 \* 2 = 23.400

--------------------------------------------------------------------------------

Задача 1-1

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≤ 5,

| x₂ = 0,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 1-1 имеем следующее решение

x₁ = 5, x₂ = 0, x₃ = 3.400

F(x) = 2.5 \* 5 + 4 \* 0 + 4.2 \* 3.400 = 26.780

Получили дробные решения.

Возьмем переменную x₃ = 3.4.

Разобьём задачу на две подзадачи, где x₃ ≤ 3 и x₃ ≥ 4.

--------------------------------------------------------------------------------

Задача 1-2

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≤ 5,

| x₂ ≥ 1,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 1-2 имеем следующее решение

x₁ = 4, x₂ = 1, x₃ = 3

F(x) = 2.5 \* 4 + 4 \* 1 + 4.2 \* 3 = 26.600

Так как решение лучше 2, то теперь лучшая задача 1-2

--------------------------------------------------------------------------------

Задача 1-1-1

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≤ 5,

| x₂ = 0,

| x₃ ≤ 3,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 1-1-1 имеем следующее решение

x₁ = 5, x₂ = 0, x₃ = 3

F(x) = 2.5 \* 5 + 4 \* 0 + 4.2 \* 3 = 25.100

Так как решение не лучше 1-2, то не будем разбивать задачу 1-1-1

--------------------------------------------------------------------------------

Задача 1-1-2

Z = 2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃ → max,

┌ 3 \* x₁ + 4 \* x₂ + 2 \* x₃ ≤ 22,

| 1 \* x₁ + 3 \* x₂ + 5 \* x₃ ≤ 31,

| 2 \* x₁ + 4 \* x₂ + 5 \* x₃ ≤ 27,

| x₁ ≤ 5,

| x₂ = 0,

| x₃ ≥ 4,

└ x₁, x₂, x₃ ∈ Z≥0

Для задачи 1-1-2 имеем следующее решение

x₁ = 3.500, x₂ = 0, x₃ = 4

F(x) = 2.5 \* 3.500 + 4 \* 0 + 4.2 \* 4 = 25.550

Так как решение не лучше 1-2, то не будем разбивать задачу 1-1-2

--------------------------------------------------------------------------------

Получили оптимальное решение

x₁ = 4, x₂ = 1, x₃ = 3

F(x) = 2.5 \* 4 + 4 \* 1 + 4.2 \* 3 = 26.600

Изображение выглядит как текст, снимок экрана, круг, дизайн

Автоматически созданное описание

Листинг программ

from typing import Iterable, Dict, Optional, Tuple

from pulp import LpMaximize, LpProblem, LpVariable, PULP\_CBC\_CMD

def lower\_index(index: int) -> str:

indexes = '₀₁₂₃₄₅₆₇₈₉'

return f'x{indexes[index]}'

def custom\_eval(expression: str, vars: Dict[str, float]) -> float:

for i in range(1, 1 + len(vars)):

expression = expression.replace(lower\_index(i), f"x{i}")

return eval(expression, None, vars)

def LPMaxSolution(

x\_count: int,

target\_function: str,

expressions: Iterable[str],

is\_int: bool = False

) -> Optional[Tuple[Dict[str, float], float]]:

model = LpProblem(name='', sense=LpMaximize)

vars = {}

for i in range(x\_count):

name\_var = f"x{i + 1}"

if is\_int:

var = LpVariable(name=name\_var, lowBound=0, cat="Integer")

else:

var = LpVariable(name=name\_var, lowBound=0)

vars[name\_var] = var

for i, expression in enumerate(expressions, 1):

model += custom\_eval(expression, vars), f'Constraint\_{i}'

model += custom\_eval(target\_function, vars)

status = model.solve(PULP\_CBC\_CMD(msg=False))

if status != 1:

return

result = {var.name: var.value() for var in model.variables()}

return result, model.objective.value()

--------------------------------------------------------------------------------------------------------

from copy import deepcopy

from typing import Iterable, Optional, List, Union

from solver import LPMaxSolution

def lower\_index(index: int) -> str:

indexes = '₀₁₂₃₄₅₆₇₈₉'

return f'x{indexes[index]}'

class Task:

def \_\_init\_\_(self,

x\_count: int,

target\_function: str,

expressions: Iterable[str],

indexes\_parts: Optional[List[int]] = None

):

self.x\_count = x\_count

self.target\_function = target\_function

self.expressions = expressions

self.x: List[Optional[Union[float, int]]] = [None] \* x\_count

self.solution: Optional[float] = None

self.status = False

self.indexes\_parts = indexes\_parts

if self.indexes\_parts is None:

self.indexes\_parts = []

self.parent: Optional['Task'] = None

self.new\_constraint: Optional[str] = None

def add\_constraint(self, constraint: str) -> None:

if constraint.endswith('<= 0'):

constraint = constraint.replace('<= 0', '== 0')

self.expressions.append(constraint)

self.new\_constraint = constraint

@property

def name(self) -> str:

return '-'.join(map(str, self.indexes\_parts))

def get\_subtask(self, index: int) -> 'Task':

obj = Task(self.x\_count, self.target\_function, deepcopy(self.expressions), deepcopy(self.indexes\_parts))

obj.indexes\_parts.append(index)

obj.parent = self

return obj

def solve(self):

result = LPMaxSolution(self.x\_count, self.target\_function, self.expressions)

if result:

self.status = True

self.solution = result[1]

if round(self.solution) == self.solution:

self.solution = int(self.solution)

for key, value in result[0].items():

index = int(key[1:]) - 1

if round(value) == value:

value = int(value)

self.x[index] = value

def is\_not\_int(self) -> int:

for i in range(self.x\_count):

if not isinstance(self.x[i], int):

return i + 1

return 0

@staticmethod

def \_\_x\_to\_str(x: Union[int, float]) -> str:

if isinstance(x, int):

return str(x)

return f'{x:.3f}'

def print\_сonstraints(self) -> None:

print()

print(' ' \* 10, 'Z =', self.target\_function, '→ max,')

for i, expression in enumerate(self.expressions):

print(' ' \* 10, '|' if i else '┌', end=' ')

expression = expression.replace('<=', '≤').replace('>=', '≥').replace('==', '=')

print(expression, ',', sep='')

print(' ' \* 10, '└', ', '.join(lower\_index(i + 1) for i in range(self.x\_count)), '∈ Z≥0')

print('')

def print\_x(self) -> None:

result = []

for i, x in enumerate(self.x, 1):

result.append(f'{lower\_index(i)} = {self.\_\_x\_to\_str(x)}')

print(', '.join(result))

def print\_solution(self) -> None:

result = 'F(x) = ' + self.target\_function + f' = '

if isinstance(self.solution, int):

result += str(self.solution)

else:

result += f'{self.solution:.3f}'

for i, x in enumerate(self.x, 1):

x\_str = self.\_\_x\_to\_str(x)

if x < 0:

x\_str = f'({x\_str})'

result = result.replace(lower\_index(i), x\_str)

print(result)

---------------------------------------------------------------------------------------------------------

from sys import exit

from math import ceil, floor

from collections import deque

from typing import Optional

from task import Task, lower\_index

LIMIT\_DEPTH = 3

def branch\_and\_bound\_method(task: Task) -> None:

global best\_solution, best\_task

print('-' \* 80)

print('Задача', task.name)

task.print\_сonstraints()

task.solve()

if not task.status:

print(f'Задача {task.name} не имеет решения.')

return

print(f'Для задачи {task.name} имеем следующее решение')

task.print\_x()

task.print\_solution()

index\_x = task.is\_not\_int()

if task.solution <= best\_solution:

if best\_task:

print(f'Так как решение не лучше {best\_task.name}, то не будем разбивать задачу {task.name}')

return

if not index\_x:

if best\_task:

print(f'Так как решение лучше {best\_task.name}, то теперь лучшая задача {task.name}')

best\_solution = task.solution

best\_task = task

return

else:

current\_value = task.x[index\_x - 1]

upper\_bound = int(ceil(current\_value))

lower\_bound = int(floor(current\_value))

print(f'''Получили дробные решения.

Возьмем переменную {lower\_index(index\_x)} = {task.x[index\_x - 1]}.

Разобьём задачу на две подзадачи, где {lower\_index(index\_x)} ≤ {lower\_bound} и {lower\_index(index\_x)} ≥ {upper\_bound}.''')

task1 = task.get\_subtask(1)

task1.add\_constraint(f'{lower\_index(index\_x)} <= {lower\_bound}')

task2 = task.get\_subtask(2)

task2.add\_constraint(f'{lower\_index(index\_x)} >= {upper\_bound}')

tasks.append(task1)

tasks.append(task2)

task = Task(

3,

'2.5 \* x₁ + 4 \* x₂ + 4.2 \* x₃',

[

'3 \* x₁ + 4 \* x₂ + 2 \* x₃ <= 22',

'1 \* x₁ + 3 \* x₂ + 5 \* x₃ <= 31',

'2 \* x₁ + 4 \* x₂ + 5 \* x₃ <= 27',

]

)

tasks = deque([task])

print('Решим задачу динамического программирования методом ветвей и границ.')

best\_solution = float('-inf')

best\_task: Optional[Task] = None

step = 0

while tasks:

current\_task = tasks.popleft()

branch\_and\_bound\_method(current\_task)

step += 1

if step > 2 \*\* (LIMIT\_DEPTH + 1):

break

if not best\_task:

print('Задача не имеет решения.')

exit(0)

print('-' \* 80)

print('Получили оптимальное решение')

best\_task.print\_x()

best\_task.print\_solution()