­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №2

Прийняття рішень в умовах повної інформації.   
Задача про упакування в контейнери

з курсу «МАТЕМАТИЧНІ МЕТОДИ ДОСЛІДЖЕННЯ ОПЕРАЦІЙ»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

## Мета роботи

Ознайомитись з методами прийняття рішень в умовах повної інформації на прикладі задачі про упакування в контейнери та дослідити особливості їх використання.

## Короткі теоретичні відомості

Задача про упакування в контейнери відноситься до NP–важких комбінаторних задач. Завдання полягає в упаковці об'єктів зумовленої форми в кінцеве число контейнерів зумовленої форми таким чином, щоб кількість використаних контейнерів була найменшою. Іноді розглядають зворотну задачу, щоб кількість або обсяг об'єктів, які упаковують, були найбільшими.

Існує безліч різновидів цієї задачі (двовимірна упаковка, лінійна упаковка, упаковка по вазі, упаковка по вартості і т.і.), які можуть застосовуватися в різних областях, наприклад, в задачі оптимального заповнення контейнерів, завантаження вантажівок з обмеженням по вазі, створення резервних копій на змінних накопичувачах і т.і.

Оскільки задача є NP-важкою, часто використовують алгоритми з евристичним та метаевристичним методом вирішення для отримання оптимальних результатів. Також активно використовуються методи штучного інтелекту, як, наприклад, нейронні мережі.

## Завдання

У контейнери вантажопідйомністю 100 потрібно розкласти вантажі за варіантом («Додаток А»).

4.1 Скласти програму, що реалізує алгоритми NFA, FFA, WFA, BFA без впорядкування і з впорядкуванням та обраховує кількість порівнянь (обчислювальну складність алгоритму), враховуючи витрати на впорядкування.

Розрахувати:

4.2 Мінімально можливу кількість контейнерів для 20-ти вантажів («Додаток А»), за допомогою (2.2) – 3 значення окремо для 1-го, 2-го та 3-го рядка варіанту та для 60 вантажів для 1-3 рядка варіанту сумісно.

4.3 Кількість контейнерів та обчислювальну складність для 20-ти вантажів, за допомогою алгоритмів NFA, FFA, WFA, BFA без впорядкування окремо для 1-го, 2-го та 3-го рядка варіанту (12 значень).

4.4 Кількість контейнерів та обчислювальну складність для 20-ти вантажів, за допомогою алгоритмів NFA, FFA, WFA, BFA з впорядкуванням окремо для 1-го, 2-го та 3-го рядка варіанту (12 значень).

4.5 Кількість контейнерів та обчислювальну складність для 60-ти вантажів, за допомогою алгоритмів NFA, FFA, WFA, BFA без впорядкування для 1-3 рядка варіанту сумісно (4 значення).

4.6 Кількість контейнерів та обчислювальну складність для 60-и вантажів, за допомогою алгоритмів NFA, FFA, WFA, BFA з впорядкуванням для 1-3 рядка варіанту сумісно (4 значення).

4.7 Результати п. 4.2–4.6 за кількістю контейнерів та за обчислювальною складністю звести у таблицю.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | С | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 14 | 1 | 47 | 24 | 82 | 35 | 32 | 04 | 54 | 43 | 98 | 86 | 40 | 78 | 59 | 62 | 62 | 83 | 41 | 48 | 23 | 24 |
| 2 | 72 | 22 | 54 | 35 | 21 | 57 | 65 | 47 | 71 | 76 | 69 | 18 | 01 | 03 | 53 | 33 | 07 | 59 | 28 | 06 |
| 3 | 97 | 20 | 84 | 08 | 34 | 98 | 91 | 76 | 98 | 15 | 52 | 71 | 89 | 59 | 06 | 10 | 16 | 24 | 09 | 39 |

## Рішення

Лістінг l2\_2.py

from collections import deque

import math

from utils import \*

class Container(object):

def \_\_init\_\_(self, c):

self.\_weights = []

self.\_indexes = []

self.\_c = c

def add(self, index, weight):

if (not self.can\_fit(weight)):

raise "There is no anough space in container"

self.\_weights.append(weight)

self.\_indexes.append(index)

def add\_if\_can\_fit(self, index, weight):

if (self.can\_fit(weight)):

self.add(index, weight)

return True

return False

def filled(self):

return sum(self.\_weights)

def can\_fit(self, weight):

return self.filled() + weight <= self.\_c

def weights(self):

return self.\_weights

def indexes(self):

return self.\_indexes

def \_\_repr\_\_(self):

return str(self.\_weights)

def compare\_by\_filled(a, b):

return cmp(a.filled(), b.filled())

class Containers(list):

def \_\_init\_\_(self, c, weights):

self.\_c = c

self.\_weights = weights

self.append\_empty()

def append\_empty(self):

self.append(Container(self.\_c))

self.last = self[-1]

def \_repr\_html\_(self):

table = IPyTable(["Weights"] + self.\_weights)

for container\_index, container in enumerate(self):

container\_indexes = container.indexes()

table.append(["C #" + str(container\_index)]

+ [(weight if weight\_index in container\_indexes else "&nbsp;")

for weight\_index, weight in enumerate(self.\_weights)])

return table.to\_html()

def \_sort\_complexity(items):

n = len(items)

return n + math.log(n)

def \_sorted\_transformer(c, weights, alg):

(containers, o) = alg(c, sorted(weights, reverse=True))

return (containers, o + \_sort\_complexity(weights))

def nfa(c, weights):

""" Next Fit Algorithm """

containers = Containers(c, weights)

for i, weight in enumerate(weights):

if (not containers.last.add\_if\_can\_fit(i, weight)):

containers.append\_empty()

containers.last.add(i, weight)

return (containers, len(weights))

def nfa\_sorted(c, weights):

return \_sorted\_transformer(c, weights, nfa)

def ffa(c, weights):

""" First Fit Algorithm """

containers = Containers(c, weights)

o = 0

for i, weight in enumerate(weights):

o += 1

if (not containers.last.add\_if\_can\_fit(i, weight)):

containers\_queue = deque(containers)

while containers\_queue:

potential\_conatiner = containers\_queue.popleft()

o += 1

if (potential\_conatiner.add\_if\_can\_fit(i, weight)):

break

else:

containers.append\_empty()

containers.last.add(i, weight)

return (containers, o)

def ffa\_sorted(c, weights):

return \_sorted\_transformer(c, weights, ffa)

def wfa(c, weights):

""" Worst Fit Algorithm """

containers = Containers(c, weights)

o = 0

for i, weight in enumerate(weights):

o += 1

if (not containers.last.add\_if\_can\_fit(i, weight)):

o += \_sort\_complexity(containers)

filled\_containers = sorted(containers, Container.compare\_by\_filled)

potential\_conatiner = filled\_containers[0]

o += 1

if (not potential\_conatiner.add\_if\_can\_fit(i, weight)):

containers.append\_empty()

containers.last.add(i, weight)

return (containers, o)

def wfa\_sorted(c, weights):

return \_sorted\_transformer(c, weights, wfa)

def bfa(c, weights):

""" Best Fit Algorithm """

containers = Containers(c, weights)

o = 0

for i, weight in enumerate(weights):

o += 1

if (not containers.last.add\_if\_can\_fit(i, weight)):

o += \_sort\_complexity(containers)

filled\_containers = sorted(containers, Container.compare\_by\_filled)

filled\_containers\_queue = deque(filled\_containers)

while filled\_containers\_queue:

potential\_conatiner = filled\_containers\_queue.pop()

o += 1

if (potential\_conatiner.add\_if\_can\_fit(i, weight)):

break

else:

containers.append\_empty()

containers.last.add(i, weight)

return (containers, o)

def bfa\_sorted(c, weights):

return \_sorted\_transformer(c, weights, bfa)

def min\_analitical(c, weights):

return sum(weights)/c

## Результат

Див l2\_2.html

## Висновки

Ознайомився з методами прийняття рішень в умовах повної інформації на прикладі задачі про упакування в контейнери та дослідив особливості їх використання.