

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №7

**Алгоритми та структури даних – 1. Основи алгоритмізації**

*Тема: «Динамічне програмування. Жадібні алгоритми»*

Команда №6

|  |  |  |
| --- | --- | --- |
| Виконали  студенти групи ІТ-03: |  | Перевірив: |
|  |  |  |
| Чабан А.Є. |  |  |
| Філянін Н.С. |  |  |
| Хамад І.А. |  | ас. Цимбал С. І. |

Київ 2020

Мета роботи: ознайомитися та навчитися застосовувати такі методи

оптимізації як динамічне програмування та жадібні алгоритми.

Хід роботи:

Завдання:

У вас є необмежена кількість купюр усіх номіналів (1, 2,

5, 10, 20, 50, 100, 200 та 500 грн). Вам необхідно зробити P

подарунків. В магазині є декілька екземплярів кожного типу

подарунку, кожен з яких характеризується якістю Qi (1 ≤ Q ≤

10) та ціною Ci (1 ≤ C ≤ 10000), де і – порядковий номер типу.

Знайти:

1) Мінімальну кількість купюр для купівлі

найдешевших подарунків P середньої якості (середня якість

визначається як 5 ≤ Q < 7)

2) Знайти максимальну кількість подарунків P високої

якості (7 ≤ Q < 9), які можна купити на обмежену суму

грошей M, таку що 100 ≤ M < 50000.

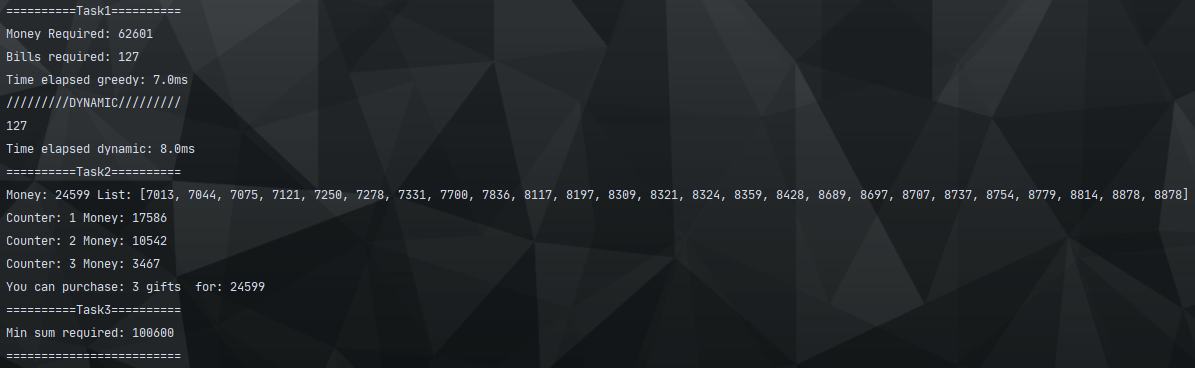
3) Мінімальну суму грошей, необхідну для купівлі

найдешевших подарунків P найвищої якості (9 ≤ Q ≤ 10).

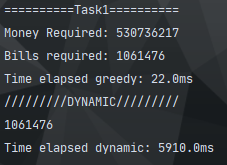
[\* Код лаби на GitHub \*](https://github.com/it03team6/asdlabs/tree/master/asd7)

Всі завдання можливо (і буде більш оптимально) виконувати використовуючи жадібні алгоритми. Далі наглядно буде видно на прикладі великих обсягів данних.

Навіть на не великих обсягах вхідних данних вже видно що в конкретно данній задачі динамічне програмування програє по швидкості виконання:



А взявши вже 100 тис подарунків — явно видно що жадібні алгоритми виграють зі значним відривом:



Проте для жадібних алгоритмів неможливо буде розв’язати задачу якщо купюри будуть наприклад номіналів 5 та 2, а необхідна сума грошей - 6. А динамічна реалізація впорається з цим завданням.

p.s Для прискорення виконання ми вдосконалили Heap Sort, зробивши сортування лише до необхідного нам елементу.

Код основного завдання:

**package** ua.kpi.fict.acts.it03.asd7;

**import** **java.util.ArrayList**;

**import** **java.util.Collections**;

**import** **java.util.List**;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

System.out.println("==========Task1==========");

task1(**12**);

System.out.println("==========Task2==========");

task2(**24599**);

System.out.println("==========Task3==========");

task3(**11**);

System.out.println("=========================");

}

**private** **static** **void** **task1**(**int** gifts)

{

**int**[] giftsPrices = **new** **int**[**100**];

**final** **int** min = **5000**;

**final** **int** max = **6999**;

**for** (**int** i = **0**; i < giftsPrices.length; i++)

{

giftsPrices[i] = randIntInRange(min,max);

}

**int**[] firstCopy = giftsPrices.clone();

**long** m = System.currentTimeMillis();

System.out.println("Bills required: " + greedy(gifts,giftsPrices));

System.out.println("Time elapsed greedy: " + (**double**)(System.currentTimeMillis() - m) + "ms");

System.out.println("/////////DYNAMIC/////////");

m = System.currentTimeMillis();

System.out.println(dinGreedy(gifts,firstCopy));

System.out.println("Time elapsed dynamic: " + (**double**)(System.currentTimeMillis() - m) + "ms");

}

**private** **static** **void** **task2**(**int** money) {

**int**[] giftsPrices = **new** **int**[**25**];

**final** **int** min = **7000**;

**final** **int** max = **8999**;

**for** (**int** i = **0**; i < giftsPrices.length; i++) {

giftsPrices[i] = randIntInRange(min, max);

}

System.out.println("You can purchase: " + maxGifts(money, giftsPrices) +" gifts " + " for: " + money);

}

**private** **static** **void** **task3**(**int** gifts) {

**int**[] giftsPrices = **new** **int**[**25**];

**final** **int** min = **9000**;

**final** **int** max = **10**\_000;

**for** (**int** i = **0**; i < giftsPrices.length; i++) {

giftsPrices[i] = randIntInRange(min, max);

}

System.out.println("Min sum required: " + minCheapest(gifts, giftsPrices));

}

**private** **static** **int** **greedy**(**int** gifts, **int**[] mediumGiftsPrices) {

**int**[] bills = {**500**, **200**, **100**, **50**, **20**, **10**, **5**, **2**, **1**};

**int** reqMoney = minCheapest(gifts, mediumGiftsPrices);

**int** counter = **0**;

System.out.println("Money Required: " + reqMoney);

**for**(**int** x : bills)

{

counter += reqMoney / x;

reqMoney %= x;

}

**return** counter;

}

**private** **static** **int** **maxGifts**(**int** money, **int**[] giftsPrice) {

List<Integer> giftList = **new** ArrayList<>(giftsPrice.length);

**for** (**int** i : giftsPrice)

{

giftList.add(i);

}

**int** giftCounter = **0**;

Collections.sort(giftList);

**if** (money < Collections.min(giftList))

**return** **0**;

System.out.println("Money: " + money + " List: " + giftList);

**do** {

money = money - Collections.min(giftList);

giftCounter++;

giftList.remove(Collections.min(giftList));

System.out.println("Counter: "+giftCounter + " Money: " + money);

//System.out.println(giftList);

} **while** (money > Collections.min(giftList));

**return** giftCounter;

}

**private** **static** **int** **minCheapest**(**int** gifts, **int**[] prices)

{

HeapSort hs = **new** HeapSort();

**int** reqMoney = **0**;

hs.sort(prices, gifts);

// System.out.println("Prices: " + Arrays.toString(prices));

**for** (**int** i = prices.length-**1**; i >= prices.length-gifts; i--)

reqMoney += prices[i];

**return** reqMoney;

}

**private** **static** **int** **dinGreedy**(**int** gifts, **int**[] mediumGiftsPrices)

{

**int** money = minCheapest(gifts, mediumGiftsPrices);

//int[] bills = {2, 5};

**int**[] bills = {**500**, **200**, **100**, **50**, **20**, **10**, **5**, **2**, **1**};

**int**[] mem = **new** **int**[money+**1**];

**int** memRes;

**for** (**int** i = **1**; i < money+**1**; i++)

mem[i] = Integer.MAX\_VALUE;

**for** (**int** i = **1**; i < money + **1**; i++) // O(money)

{

**for** (**int** j = **0**; j < bills.length; j++)

{

**if** (bills[j]<=i) {

memRes = mem[i - bills[j]];

**if** (memRes != Integer.MAX\_VALUE) {

mem[i] = Math.min(memRes + **1**, mem[i]);

}

}

}

}

**return** mem[money];

}

**public** **static** **int** **randIntInRange**(**int** min, **int** max)

{

max -= min;

**return** (**int**) (Math.random() \* ++max) + min;

}

}

Код додатково створеного сортування:

**package** ua.kpi.fict.acts.it03.asd7;

**class** **HeapSort** { //O(count\*logN)

**static** **void** **sort**(**int**[] array, **int** count)

{

**for** (**int** i = array.length/**2** -**1**;i >= **0**; i--)

{

heap(array, array.length, i);

}

**for** (**int** i = array.length-**1**; i>**0**; i--)

{

**int** tmp = array[**0**];

array[**0**] = array[i];

array[i] = tmp;

**if** (i <= array.length - count)

**break**;

heap(array,i,**0**);

}

}

**private** **static** **void** **heap**(**int**[] array, **int** heapSize, **int** idx)

{

**int** largest = idx;

**int** left = **2**\*idx + **1**;

**int** right = **2**\*idx+**2**;

**if** (left < heapSize && array[left] < array[largest])

largest = left;

**if** (right < heapSize && array[right] < array[largest])

largest = right;

**if** (largest != idx)

{

**int** swap = array[idx];

array[idx] = array[largest];

array[largest] = swap;

heap(array, heapSize, largest);

}

}

}

Висновки: Виконуючи цю лабораторну роботу ми освоїли використання жадібних алгоритмів та динамічного програмування на практиці.