Синтез и анализ на алгоритми

Упражнение №4

Студент: Иван Ивайлов Милев

ФК №: 121218022

Грипа: 52Б

# Задача 32 Алчни(Greedy) алгоритми:

Сложността на този алгоритъма съставен за решаването на проблема за най-малък брой монети е линейна. Този алгоритъм е от така наречените алчни алгоритми, защото се опитва да реши „целия“ проблем с моментните данни. В този случай ако приемем, че трябва да платим 135 на първата стъпка ще опитаме да платим с монети от 50, което не е възможно, но можем да върнем 100 . Остават 35 не можем повече да работим с 50 затова отиваме на следващата стъпка 20, тук можем да дадем само 20 и остават 15. Вече сме на монета от 10, остават 5 и с 5 ст. Ще дадем цялата сума и няма да се наложи да използваме монети от 1 ст, както и да правим изчисления за тях. Тези алгоритми се наричат алчни, защото на всяка стъпка се опитват да решат проблема ако е възможно без да имаме повече стъпки.

# Задача 33. Рекурсивния вариант на 32:

Има я решена в прикачения cpp файл.

# Задача 34. Изчисляване на най-голям общ делител на две естествени числа чрез рекурсия. Задача за Ханойските кули. Разделяй и владей

Идеята на алгоритмите от типа Разделяй и Владей е да разделят проблема на различни подпроблеми, така че да са достатъчно малки и изолирани, за да могат да се реализират самостоятелно.

В задачата за най-голям общ делител това са задачите за намиране на минимум от 2 числа и за най-малък общ делител.

В задачата за Ханойските кули в отделяме преместването на пул от това как ще меси пуловете.

# Задача 36/37. Изчисляване на числото на Фибоначи по въведен номер в редицата на Фибоначи чрез рекурсия.

При неоптимизираната имплементация на алгоритъма за получаване на число от реда на Фибуначи сложността му е O(2n), което е далеч от хубава(ниска) сложност. Затова се използва подход от динамичното програмиране – меморизация. С подхода меморизация запазваме вече сметнатите числа на фибуначи, за да предотвратил повторното им смятане. Например ако искаме да сметне 4тото число на фибуначи без меморизация ще изглежда така:

fib(3) + fib(2)

fib(2) + fib(1) ------ fib(1) + fib(0)

fib(1) + fib(0) ----- fib(0) ----- fib(0)

Както се вижда в случая за числото 0 се смята 4 пъти, първото число се смята 3 пъти, второто 2 пъти. След въвеждането на меморизация, т.е. записване на вече сметнатите числа ще всяко n-то число на фибуначи ще бъде сметнато точно веднъж, което ще доведе до много по-малко работа със стека(извикване на функции) и на процесора, защото няма да има нужда от толкова пресмятания. За жалост тази имплементация би ползвала повече памет от обикновената, но това не е толкова голям минус, защото процесорното време е по-скъпо от паметта.

Тук отново се използва използва паметта, за да не се налага пресмятане повече от веднъж за даден капацитет, тъй като с намаляването на капацитета, вече може да сме намерили оптималният предмет. Сложността на този алгоритъм е O(N\*W1\*W2)

# Задача 46. Да се разгледат и сравнят различни алгоритми за сортиране

## Метод на мехурчето:

Метода на мехурчето работи по такъв начин, че най-малкия/най-големия елемент от масива отива съответно най-отгоре/най-отдолу с други думи изплува. Този алгоритъм е изключително прост за реализация, но за жалост не е много ефективен неговата сложност е квадратична(за всеки елемент се обхожда масива от край до край), което значи, че с увеличаването на масива, който сортираме той не се справя особено добре.

## Сортиране чрез вмъкване

Идеята на сортирането чрез вмъкване е да се итерира през всички елементи като всеки от тях се вмъкне на мястото, в което не нарушава реда(нарастващ/намаляващ) в елементите пред себе си. Така този алгоритъм е много лесен за имплементация за жалост отново не достатъчно бърз, но по-оптимизиран от сортиране по метода на мехурчето и метода чрез селекция. Плюс на този алгоритъм е, че може да се ползва за сортиране на непрестанно идващи елементи. Отново сложността му е квадратична

## Сортиране чрез селектиране

Сортирането чрез селекция сортира чрез търсене(селектиране) на най-малкия/големия елемент в оставащия масив и така подрежда всички елементи по големина. Отново сложността му е квадратична.

## Бърза сортировка

Този вид сортировка е голям скок в скоростта за сортиране, реално в най-лошия случай сложността му е квадратично, но това е много рядко състояние, а в повечето случаи се справя с линейна или O(nlog(n)) сложност. Този алгоритъм спада и към разделяй и владей алгоритмите, защото избира елемент, който да разделя масива на 2 части, лявата с по-малки, дясна с по-големи елементи от него, това се повтаря до сортирането на масива.

## Сортировка на Шел

Сортировката на Шел е обединение на метода на мехурчето и на вмъкването, като използва сравнение на елементи в различни краища на масива така позволява на елементи, които са далеч от правилната си позиция да бъдат преместени лесно. Тази дистанция между сравняваните елементи се намалява докато стигне 1(съседни). Отново в най-лошия сценарий алгоритъма ще бъде с квадратична сложност, но е възможно да сортира и в n logn.

## Пирамидална сортировка

Пирамидалната сортировка прилича на сортирането чрез селектиране, подобно на този метод пирамидалната сортировка разделя масива на сортирана и несортирана част. За разлика от сортирането чрез селекци в случая несортираната част на масива не се обикаля линейно и така успява да сортира списък със сложност O(nlog(n))

## Сортиране чрез сливане

Този метод използва разделянето на масива на двойки, като всяка двойка бива сортирана и след това всеки 2 двойки биват сортирани помежду си и така до построяването на целия масив отново представител на алгоритми от вида на разделяй и владей. Този алгоритъм има сложност от O(nlog(n)).

## Сравнение на различните алгоритми с произволни данни и ненапълно неподредени данни.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Size | Bubble sort time in microseconds | Insertion sort time in microseconds | Selection sort time in microseconds | Quick sort time in microseconds | Shell sort time in microseconds | Heap sort time in microseconds | Merge sort time in microseconds |
| 10000 | 252153 | 24494 | 138823 | 525 | 31 | 1457 | 1457 |
| 20000 | 1012973 | 89391 | 499309 | 1106 | 62 | 3021 | 2849 |
| 30000 | 2230065 | 212553 | 1122092 | 1593 | 90 | 4354 | 4353 |
| 40000 | 3895251 | 355739 | 1853656 | 2232 | 126 | 6335 | 6246 |
| 50000 | 5859414 | 557333 | 2860597 | 2958 | 163 | 7412 | 7581 |
| 60000 | 8571739 | 805273 | 4222052 | 3587 | 195 | 9188 | 8872 |
| 70000 | 11584001 | 1112685 | 5619523 | 4287 | 237 | 10797 | 10886 |
| 80000 | 15109439 | 1443147 | 7341319 | 5493 | 505 | 12029 | 12557 |
| 90000 | 19177380 | 1799072 | 9327543 | 5554 | 397 | 13705 | 13558 |
| 100000 | 24020627 | 2240034 | 12146843 | 6804 | 329 | 20027 | 16905 |
| 110000 | 29734946 | 2777776 | 14074231 | 7409 | 465 | 18118 | 19618 |
| 120000 | 34319849 | 3238585 | 16955582 | 7439 | 797 | 31450 | 19436 |
| 130000 | 40485326 | 3829070 | 19603148 | 8373 | 406 | 20534 | 20476 |
| 140000 | 47984941 | 4523933 | 22742957 | 9333 | 464 | 22221 | 21410 |
| 150000 | 59731823 | 5385173 | 27561810 | 9720 | 465 | 25198 | 24818 |
| 160000 | 64230829 | 5970331 | 30799121 | 10552 | 521 | 27191 | 26485 |
| 170000 | 69376631 | 6298440 | 32394636 | 10647 | 561 | 26595 | 26995 |
| 180000 | 74591434 | 7067209 | 36180848 | 11688 | 583 | 28839 | 28574 |
| 190000 | 83164883 | 7863641 | 40315692 | 11955 | 605 | 30562 | 31150 |
| 200000 | 92088206 | 8758215 | 45183971 | 12661 | 650 | 31449 | 31250 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Size | Bubble sort time in microseconds | Insertion sort time in microseconds | Selection sort time in microseconds | Quick sort time in microseconds | Shell sort time in microseconds | Heap sort time in microseconds | Merge sort time in microseconds |
| 10000 | 131249 | 34 | 114790 | 576 | 31 | 1332 | 1548 |
| 20000 | 522907 | 65 | 450139 | 1110 | 61 | 2786 | 2926 |
| 30000 | 1137590 | 106 | 1006590 | 1689 | 92 | 4273 | 4276 |
| 40000 | 2011099 | 141 | 1785465 | 2216 | 126 | 5748 | 6199 |
| 50000 | 3145308 | 176 | 2791257 | 2785 | 187 | 7364 | 7710 |
| 60000 | 4523493 | 211 | 4014210 | 3355 | 195 | 9043 | 8827 |
| 70000 | 6166202 | 247 | 5450501 | 3943 | 227 | 10348 | 10580 |
| 80000 | 8038780 | 327 | 7140278 | 4525 | 259 | 12336 | 12067 |
| 90000 | 10174211 | 317 | 9025443 | 5121 | 299 | 13552 | 13541 |
| 100000 | 12559205 | 350 | 11154484 | 5620 | 347 | 15273 | 15266 |
| 110000 | 15185131 | 387 | 13498641 | 6868 | 360 | 16769 | 17414 |
| 120000 | 18092101 | 415 | 17607268 | 6967 | 391 | 19573 | 22982 |
| 130000 | 23433823 | 457 | 21100831 | 7398 | 428 | 19937 | 20074 |
| 140000 | 24693952 | 504 | 21942690 | 8217 | 469 | 21812 | 21871 |
| 150000 | 28251497 | 530 | 25113811 | 8814 | 545 | 23324 | 23565 |
| 160000 | 37978461 | 562 | 30429317 | 9886 | 720 | 26790 | 25954 |
| 170000 | 42841257 | 603 | 37780691 | 10643 | 563 | 30879 | 30226 |
| 180000 | 46265735 | 628 | 37367478 | 10663 | 638 | 28923 | 28079 |
| 190000 | 45590949 | 679 | 40365699 | 10941 | 599 | 30507 | 29850 |
| 200000 | 50264958 | 724 | 44682493 | 11841 | 653 | 32274 | 30918 |

От данните положени горе се вижда как сложността на алгоритми директно влия на изпълнението им с увеличаване на размера на масивите и как някои алгоритми печелят изключително много от не „много“ разреден масив.

# Задача 47. Списък.

Работата със списък, като показания е с линейна сложност, защото операцията за намиране е с линейна сложност, която би била най-честно използваната операция. Операцията за добавяне на елемента на конкретна позиция също е с линейна сложност. Добавянето в началото на списъка е с константна сложност, защото пазим референция към началото (главата) на списъка. Операцията триене също както добавяне е с линейна сложност поради нуждата от обхождане на списъка до намиране на елемента, който търсим за триене.

# Задача 48. Решението е в прикачения cpp файл.