ИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл холбооны технологийн сургууль

БИЕ ДААЛТЫН АЖЛЫН ТАЙЛАН

Алгоритм шинжилгээ ба зохиомж
(FCSM301) 2024-2025 оны хичээлийн
жилийн намар

Бие даалтын ажлын нэр:

Хичээл заасан багш: Д.Батмөнх
Бие даалтын ажил гүйцэтгэсэн: С.Тэмүүжин B221960002

Улаанбаатар хот. 2024 он

https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm
https://en.wikipedia.org/wiki/Greedy_algorithm
https://en.wikipedia.org/wiki/Dynamic_programming
https://www.geeksforgeeks.org/python-program-for-coin-change/
https://www.geeksforgeeks.org/divide-and-conquer/
https://onlinenotepad.org/notepad
https://www.programiz.com/dsa/divide-and-conquer
https://www.programiz.com/dsa/dynamic-programming

Divide-and-Conquer

"Divide-and-Conquer" гэдэг нь тухайн асуудлыг ижил буюу холбогдох төрлийн хоёр ба түүнээс дээш дэд бодлого болгон рекурсив байдлаар задалж, эдгээр дэд бодлогуудыг бие даан шийдэж, дараа нь тэдгээрийн шийдлүүдийг нэгтгэн анхны бодлогыг шийддэг алгоритмын юм.

Ерөнхий алхмууд нь:

1. **Divide**: Асуудлыг жижиг дэд асуудалд хуваа.
2. **Conque**r: Дэд бодлогуудыг рекурсив аргаар шийд. Хэрэв тэдгээр нь хангалттай жижиг бол тэдгээрийг шууд шийдээрэй.
3. **Combine**: Анхны асуудлын шийдлийг олохын тулд дэд асуудлын шийдлүүдийг нэгтгэнэ.

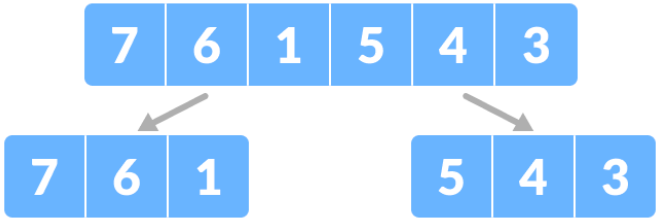
Merge Sort

Бүхэл тоонуудын эрэмбэлэгдээгүй массив өгөгдсөн бол массивыг өсөх дарааллаар эрэмбэл.

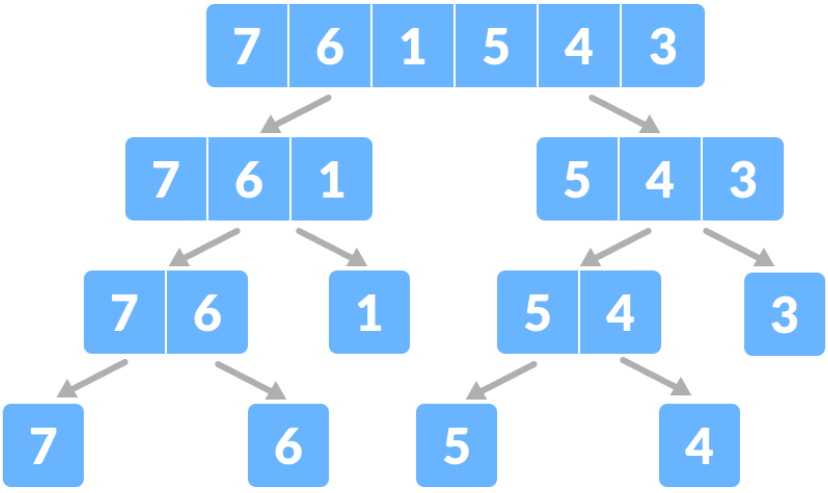
```
def merge_sort(arr):
        if len(arr) > 1:
            mid = len(arr) // 2
            L = arr[:mid]
            R = arr[mid:]

            merge_sort(L)
            merge_sort(R)

            i = j = k = 0

            while i < len(L) and j < len(R):
                if L[i] < R[j]:
                        arr[k] = L[i]
                        i += 1
                else:
                        arr[k] = R[j]
                        j += 1
                k += 1

            while i < len(L):
                arr[k] = L[i]
                i += 1
                k += 1

            while j < len(R):
                arr[k] = R[j]
```

```
j += 1
k += 1
```

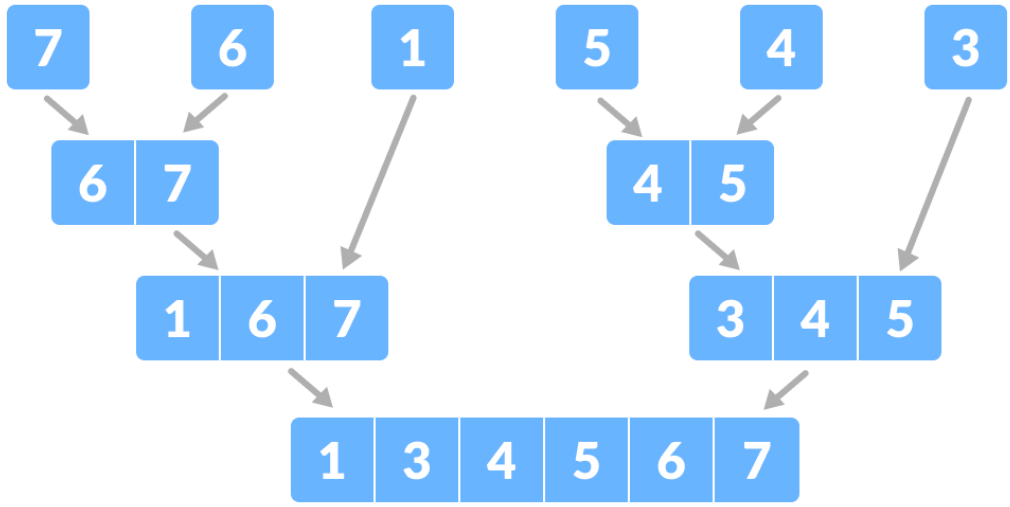
Массивыг хоёр хэсэгт хуваа**.**



Дахин хэлэхэд, тус тусын элементүүдийг авах хүртэл дэд хэсэг бүрийг рекурсив байдлаар хоёр хагас болгон хуваа.



Одоо тус тусын элементүүдийг эрэмбэлсэн байдлаар нэгтгэ. Энд, conquer, combine алхмууд зэрэгцэн явдаг.



Dynamic Programming

Dynamic Programming (DP) нь оновчлолын асуудлыг илүү энгийн дэд бодлого болгон задалж, илүүдэл тооцооллоос зайлсхийхийн тулд эдгээр дэд бодлогын үр дүнг хадгалах замаар шийдвэрлэх алгоритмын арга юм. Асуудал нь давхцаж буй дэд асуудлууд, оновчтой дэд бүтцийг харуулсан тохиолдолд энэ нь ялангуяа ашигтай байдаг.

There are two main approaches:

- **Top-Down (Memoization)**: Асуудлыг рекурсив аргаар шийдэж, шийдсэн дэд асуудлын үр дүнг хадгална.
- **Bottom-Up (Tabulation)**: Боломжтой бүх жижиг дэд асуудлуудыг эхлээд шийдэж, үр дүнг нь том дэд асуудлуудын шийдлийг бий болгоход ашиглана.

Fibonacci Numbers

Memoization:

```
memo = {}
def fib_dp(n, memo={}):
        if n in memo:
            return memo[n]
        if n <= 1:
            memo[n] = n
            return n
        memo[n] = fib_dp(n-1, memo) + fib_dp(n-2, memo)
        return memo[n]
```

Tabulation:

```
def fibonacci_tabulation(n):
        if n <= 2:
            return 1

        fib = [0] * (n + 1)
        fib[1], fib[2] = 1, 1
        for i in range(3, n + 1):
            fib[i] = fib[i - 1] + fib[i - 2]
        return fib[n]
```

F(0) = 0
F(1) = 1
F(2) = F(1) + F(0)
F(3) = F(2) + F(1)
F(4) = F(3) + F(2)

Илүүдэл тооцооллоос зайлсхийхийн тулд дэд асуудлын үр дүнг memo хадгална.

Chapter 3: Greedy Algorithms

Greedy Algorithm нь одоогийн байгаа хамгийн сайн хувилбарыг сонгох замаар асуудлыг шийдвэрлэх арга юм. Одоогийн хамгийн сайн үр дүн нь ерөнхий оновчтой үр дүнг авчрах эсэхээс санаа зовох хэрэггүй.

Сонголт буруу байсан ч алгоритм нь өмнөх шийдвэрийг хэзээ ч буцаадаггүй. Энэ нь дээрээс доош чиглэсэн байдлаар ажилладаг.

Энэ алгоритм нь бүх асуудалд хамгийн сайн үр дүнг өгөхгүй байж магадгүй юм. Энэ нь Global хамгийн сайн үр дүнд хүрэхийн тулд local хамгийн сайн сонголтыг үргэлж эрэлхийлдэг учраас тэр юм.

Coin Change Problem

```
def coin_change_greedy(amount, denominations):

        result = []
        for coin in denominations:
            count = amount // coin
            if count > 0:
                    result.append((coin, count))
                    amount -= coin * count

            if amount > 0:
                    print("The greedy algorithm could not find an exact solution.")
                    return None
            else:
                    return result
```

43
[25, 10, 5, 1]

[(25, 1), (10, 1), (5, 1), (1, 3)]

25×1=25
10×1=10
5×1=5
1×3=3

Recursion vs Divide-and-Conquer

Recursion гэдэг нь тухайн асуудлын шийдэл нь ижил асуудлын жижиг тохиолдлуудын шийдлүүдээс хамаардаг арга юм.

Тухайн асуудлын шийдэл нь ижил асуудлын жижиг тохиолдлуудын шийдлүүдээс хамаардаг арга юм.

Divide and Conquer гэдэг нь асуудлыг бие даасан дэд бодлогод хуваах, рекурсив аргаар шийдвэрлэх, үр дүнг нэгтгэх зэрэг recursive тодорхой хэлбэр юм.

Асуудлыг бие даасан дэд бодлогод хуваах, рекурсив аргаар шийдвэрлэх, үр дүнг нэгтгэх зэрэг recursive тодорхой хэлбэр.

Example:

- **Recursion**: Факториалуудыг тооцоолох n!=n×(n−1) × (n-2) × … × 1.

```
def factorial(n):
        if n <= 1:
            return 1
        return n * factorial(n - 1)
```

5!=5×4×3×2×1=120

- **Divide-and-Conquer**: Merge Sort, массивыг хагас болгон хувааж, бие даан эрэмбэлж, нэгтгэдэг.

Divide-and-Conquer vs Dynamic Programming

Хоёр алгоритм нь асуудлыг дэд асуудал болгон хуваадаг боловч дэд асуудлуудыг хэрхэн шийдэж, нэгтгэж байгаагаараа ялгаатай.

Divide-and-Conquer:

Бүх дэд асуудлууд нь тусдаа бие биетэйгээ харьцахгүй. Бүгдэнгийх нь хариултыг олоод бие биетэйгээ нэгтгээд анхны гол асуудлыг олно.

Dynamic Programming:

Sub problems overlap.Бүх дэд асуудлууд нь бие биетэйгээ давхцан. Үр дүн нь хадгалагдаж, хүчин чадал бага ашиглана

Dynamic Programming vs Greedy Algorithms

Хоёр алгоритм нь optimization асуудалд ашиглагддаг ч арга барилаараа өөр

.

Dynamic Programming:

Бүх боломжтой асуудлыг бодож хамгийн сайн үр дүнтэйг нь сонгоно. Үр дүнгээ хадгалж өөр дэд асуудалд ашиглана. Өмнөх сонголтоосоо шалтгаалж ирээдүйн хамгийн үр дүнтэйг сонгоно.

Хамгийн оновчтой үр дүнг гаргах ч илүү бодох тул удаан байж болно.

Greedy Algorithms:

Дэд асуудал дээрээ хамгийн сайн үр дүнг сонгоно. Өмнө гаргасан сонголтоо боддоггүй. Дэд асуудал нь optimization хийх нь гол асуудлын үр дүнг олоход л ашиглана.

Хурдан ч оновчтой үр дүнг гаргахгүй байж болно.

References

Footnotes

1. https://www.programiz.com/dsa/divide-and-conquer
2. https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm
3. https://en.wikipedia.org/wiki/Dynamic_programming
4. https://en.wikipedia.org/wiki/Greedy_algorithm