ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл холбооны технологийн сургууль

БИЕ ДААЛТ II

2024-2025 оны хичээлийн жилийн намар F.CSM301

Багш: Д. Батмөнх
Бие даалтын ажил гүйцэтгэсэн: B221960009 Баяртаа Н.

1. Divide-and-Conquer

Divide-and-Conquer нь асуудлыг жижиг дэд асуудлуудад хуваан шийдвэрлэх арга юм.

Энэ арга нь:

1. Асуудлыг дэд асуудалд хуваана.
2. Дэд асуудал бүрийг рекурсив аргаар шийдвэрлэх замаар ялгана.
3. Анхны асуудлын шийдлийг олохын тулд үр дүнг нэгтгэнэ.

Merge Sort-д массив нь хоёр хагаст хуваагдана. Хагас бүрийг рекурсив байдлаар эрэмбэлж, дараа нь эрэмбэлсэн талыг нэгтгэнэ.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    return merge(left_half, right_half)
def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

2. Dynamic programming

Динамик программчлал нь том асуудлыг жижиг дэд асуудлуудад хуваах бөгөөд дэд асуудал бүрийн шийдлийг санах ойд хадгалж, дахин тооцоолох шаардлагагүй болгодог. Энэ арга нь асуудлыг рекурсив аргаар шийдэж, нэмэлт тооцооллоос зайлсхийхийн тулд дэд бодлого бүрийн үр дүнг хадгалдаг. Дэд асуудал дахин тулгарах үед түүнийг дахин тооцоолохын оронд хадгалсан шийдлийг гаргаж авдаг.

Жишээ: Фибоначчийн дарааллын тооцоололд Фибоначчийн тоо бүрийг дахин тооцоолохын оронд өмнө нь тооцсон үр дүнг хадгалах замаар тооцоолж болно.

```
class Solution(object):
    def climbingStairs(self, n):
        if n == 1:
            return 1
        if n == 2:
            return 2

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

        return dp[n]
```

3. Greedy algorithm

> Энэ алгоритм нь өгөгдсөн үед хамгийн сайн санагдах шийдлийг сонгодог. Энэ нь локал оптимум шийдэл олох боловч глобал оптимум шийдэл байх баталгаагүй.

3.1 Greedy алгоритмын гол шинж чанарууд:

- Алхам бүрд хамгийн боломжит шийдвэрийг гаргадаг.
- Асуудлыг жижиг дэд асуудал болгон хувааж, тус бүрийг бие даан шийдвэрлэх замаар ерөнхий оновчтой шийдэлд хувь нэмрээ оруулах боломжтой.
- Нэгэнт шийдвэр гаргасан бол түүнийг хэзээ ч өөрчлөхгүй. Алгоритм нь өмнөх сонголтуудыг дахин авч үзэхгүй.

```
def min_coins(amount):
    coins = [25, 10, 5, 1]
    coin_count = 0
    for coin in coins:
        while amount >= coin:
            amount -= coin
            coin_count += 1
    return coin_count
amount = 63
print(min_coins(amount))
```

4. Recursion

Рекурсив нь тухайн функц нь асуудлыг шийдэхийн тулд өөрийгөө дууддаг программчлал юм. Энэ нь асуудлыг илүү жижиг, удирдах боломжтой дэд асуудал болгон хуваадаг бөгөөд эдгээр дэд асуудлын шийдлийг нэгтгэж анхны асуудлыг шийддэг.

Рекурсив функцэд хоёр үндсэн хэсэг байдаг:

1. Функц өөрөө дуудагдахаа больсон нөхцөл. Энэ нь хязгааргүй дахин давтагдахаас сэргийлнэ.
2. Функц нь шинэ аргументуудаар өөрийгөө дууддаг хэсэг бөгөөд ихэвчлэн үндсэн тохиолдолд ойртдог.

Жишээ нь: тооны факториалыг тооцоолохдоо рекурсивыг ашиглан ихэвчлэн хийдэг.

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

1. Memoization (Дээрээс Доошоо хандах): Энэ хандлага нь асуудлыг дахин давтан тооцоолохоос зайлс хийхийн тулд дэд асуудлуудын хариуг хүснэгт (эсвэл толь) дотор хадгалж, асуудлыг рекурсивээр шийдэх арга юм. Хэрэв дэд асуудлын хариу аль хэдийн тооцоологдсон бол хадгалсан хариуг дахин ашиглана.
2. Tabulation (Доороос Дээшээ хандах): Энэ хандлага нь дэд асуудлуудын шийдэлүүд хамгийн бага дэд асуудлаас эхлэн хүснэгтэд дүүргэн, үе шаттайгаар өсгөж тооцоолдог. Энэ нь ихэвчлэн бага санах ойг шаарддаг ба рекурсив дуудах процессыг зайлс хийдэг.
3. Recursion vs Divide-and-Conquer Recursion нь функц өөрийгөө дуудах замаар асуудлыг шийдвэрлэх арга юм. Divide-and- Conquer нь рекурсийг ашигладаг боловч асуудлыг тодорхой загвараар (хуваах шийдвэрлэх нэгтгэх) шийдвэрлэдэг.
4. Divide-and-Conquer vs Dynamic Programming
Divide-and-Conquer нь асуудлыг хуваан шийдвэрлэдэг бол Dynamic Programming нь давтагдах дэд асуудлуудын шийдлийг санах ойд хадгалж, дахин тооцоолохоос зайлсхийдэг.
5. Dynamic Programming vs Greedy
Dynamic Programming нь бүх боломжит шийдлүүдийг харгалзан үзэж, хамгийн сайн шийдлийг олдог бол Greedy алгоритм нь тухайн үед хамгийн сайн санагдах шийдлийг сонгодог.