**ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ**

**Мэдээлэл холбооны технологийн сургууль**



**БИЕ ДААЛТЫН АЖЛЫН**

**ТАЙЛАН**

**Алгоритмын шинжилгээ ба зохиомж (F.CS301)**

**2024-2025 оны хичээлийн жилийн намар**

**Хичээл заасан багш:** Д. Батмөнх

**Бие даалтын ажил гүйцэтгэсэн:** А. Эрдэнэбаяр (B221960012)

**Лабораторийн цаг:** 2-2

**Улаанбаатар хот**

**2024**

1. Divide-and-Conquer

Энэ нь асуудлыг жижиг дэд асуудал болгон хувааж, дэд бодлого бүрийг бие даан шийдэж, дараа нь тэдгээрийн шийдлүүдийг нэгтгэж анхны асуудлыг шийддэг алгоритмын дизайны парадигм (загвар) юм. Энэ арга нь гурван үндсэн алхмаас бүрдэнэ.

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

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

1. Dynamic

Динамик программчлал (DP) нь асуудлыг жижиг дэд асуудал болгон хувааж, дэд бодлого бүрийг нэг удаа шийдэж, шийдлийг нь хадгалах замаар шийдвэрлэхэд ашигладаг арга юм. Хадгалсан шийдлүүдийг дараа нь илүү том дэд асуудлуудыг үр дүнтэй шийдвэрлэхийн тулд дахин ашиглаж болно.

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]

1. Greedy

Шунахай алгоритм нь асуудлыг шийдвэрлэх арга бөгөөд алхам тутамдаа хамгийн сайн хувилбарыг сонгох замаар шийдвэр гаргадаг бөгөөд эдгээр сонголтууд нь оновчтой шийдэлд хүргэнэ гэж үздэг. Өөрөөр хэлбэл, greedy нь эдгээр сонголтын ирээдүйн үр дагаврыг заавал тооцохгүйгээр хамгийн сайн ерөнхий үр дүнд хүргэнэ гэсэн хүлээлттэй, үе шат бүрд орон нутгийн оновчтой сонголтыг хийдэг.

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))

1. 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)