ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ

Мэдээлэл, Холбооны Технологийн Сургууль

БИЕ ДААЛТ-2

Алгоритмын шинжилгээ ба зохиомж(F.CS301)

2024-2025 оны хичээлийн жилийн намар

Хичээл заасан багш:..... Д. Батмөнх

Хийсэн оюутан:..... Д. Туяа /B200900019/

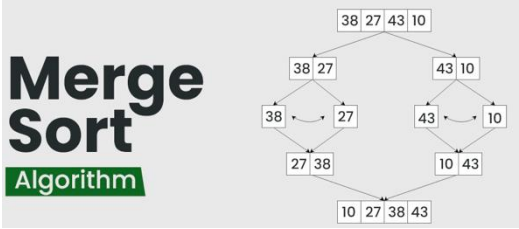
Лабораторын цаг: 2-1 пар

Улаанбаатар хот

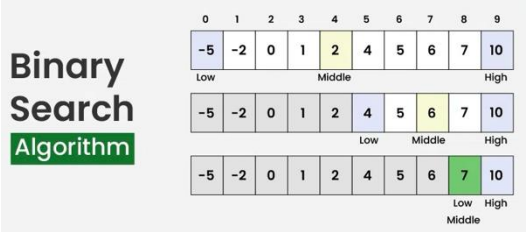
2024 он

Энэхүү бие даалтын ажлын хүрээнд оюутан дараах сэдвүүдийг судалж өөрийн ойлгосноо жишээ бодлогоор тайлбарлан бичнэ.

1. Divide-and-Conquer
    - Уг алгоритм нь асуудлыг жижиг хэсгүүдэд хувааж(DIVIDE) тус бүрийг рекурсивээр шийдвэрлээд(CONQUER) хариуг нь нэгтгэж асуудлыг шийддэг арга юм. Нэгтгэх үе шат нь хамгийн чухал.



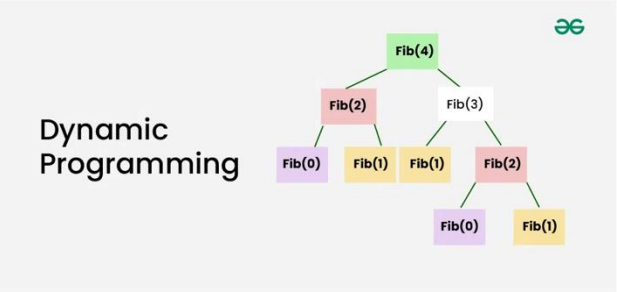
    - Жишээ алгоритм:
        - Merge sort (Нэгтгэн эрэмбэлэх) Том жагсаалтыг хуваан жижиг жагсаалтуудыг эрэмбэлж нэгтгэнэ.
        - Binary search (2-тын хайлт) Том жагсаалтыг хуваан хүссэн элементүүдийг дэд жагсаалтуудад хайх.




```
def binary_search(arr, left, right, target):
    if left > right:
        return -1  # Target not found
    mid = (left + right) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] > target:
        return binary_search(arr, left, mid - 1, target)
    else:
        return binary_search(arr, mid + 1, right, target)

# Жишээ ашиглалт:
arr = [2, 3, 5, 7, 11, 13, 17]
target = 7
result = binary_search(arr, 0, len(arr) - 1, target)
print(f"Target {target} found at index: {result}")
```

2. Dynamic Programming

- 

Divide-and-Conquer ийн сайжруулсан хувилбар гэж үздэг. Нарийн төвөгтэй асуудлуудыг энгийн дэд асуудал болгон задлаж тэдгээрийн оновчтой шийдлийг хүснэгт эсвэл массив хэлбэрээр хадгалана. Хадгалсан шийдлүүдийг ашигласнаар тооцоолох хугацааг багасгаж чаддаг.
- Жишээ алгоритм1:

Фибоначчийн дараалал: Дарааллын ямар ч гишүүн нь өмнөх 2 гишүүний нийлбэр байдаг. (0,1,1,2,3,5,8,13,...)

F(n) = F(n-1) + F(n-2)

F (0) =0, F (1) =1

Memoization (Top-Down): Урьдчилан тооцоолсон утгыг хадгалж дахин ашиглана.

Tabulation (Bottom-Up): Бүх утгыг эхнээс нь тооцоолж хүснэгтэд хадгална.

Tabulation аргаар тооцоолбол:

Хүснэгт (DP массив) үүсгэж, F (0) болон F (1)-ийг хадгална.

Хүснэгтийн 2 дахь индексээс эхлэн F (i) = F(i-1) +F(i-2)-ийг тооцоолно.

Хүснэгтийн хамгийн сүүлийн элемент нь F(n) утгыг илэрхийлнэ.

```
def fib_tab(n):
    if n <= 1:
        return n
    dp = [0] * (n + 1)
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
print(fib_tab(10))
```

Жишээ 2:
Үүргэвчтэй цүнхний бодлого

N Багтаамжтай цүнхэнд аливаа зүйлийг оруулан, хамгийн их үнэ цэнэтэй хослолыг олох.

3. Greedy Algorithms – Ye шат бүрт хамгийн оновчтой сонголтыг хийдэг алгоритм. Урт хугацааны үр дүнг тооцохгүйгээр зөвхөн тухайн нөхцөл байдалд хамгийн сайн гэсэн сонголтоо хийдэг.

Жишээ алгоритмууд:

1. Huffman Encoding
2. Dijkstra’s Shortest Path Algorithm
3. Kruskal’s Algorithm (Мөрний холбоос олох)
4. Fractional Knapsack Problem

Жишээ бодлого: Fractional Knapsack Problem

Асуудал:

Багтаамжтай цүнхэнд аливаа зүйлсийг хэсэгчлэн (fractional) оруулан, хамгийн их үнэ цэнэтэй байхаар сонгоно.

Алгоритм:

1. Зүйлсийг үнэ цэнэ/жин харьцаагаар эрэмбэлэх.
2. Цүнхэнд багтах хэмжээгээр сонгох, хэрэв жин хэтэрвэл хэсэгчлэн авна.

```
def fractional_knapsack(values, weights, capacity):
    items = sorted(zip(values, weights), key=lambda x: x[0]/x[1], reverse=True)
    total_value = 0
    for value, weight in items:
        if capacity >= weight:
            total_value += value
            capacity -= weight
        else:
            total_value += value * (capacity / weight)
            break
    return total_value
```

```
# Жишээ ашиглалт
values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print(fractional_knapsack(values, weights, capacity))
```
Үр дүн: 240.0

1. Recursion болон Divide and Conquer нь Хоорондоо холбоотой ойлголт боловч хэрэглээ, зорилго, асуудал шийдвэрлэх аргачлалаараа ялгаатай.

Recursion

- Өөрийгөө дахин дуудаг функц ба асуудлыг задлаж хуваахгүйгээр эцсийн нөхцөлд хүрээд зогсдог.
```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Divide and Conquer

- Асуудлыг жижиг хэсгүүдэд хувааж, тус бүрд шийдэж хариуг нэгтгэнэ. Рекурсив функцийг үндсэн аргачлал болгон ашигладаг ч divide, conquer, combine гэсэн үе шатуудтай.

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

        merge_sort(left)
        merge_sort(right)

        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
```

```
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            j += 1
        k += 1

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

    while j < len(right):
        arr[k] = right[j]
        j += 1
        k += 1
```

Шинж чанар	Recursion	Divide and Conquer
Зорилго	Өөрийгөө дуудаж асуудлыг шийдвэрлэх.	Асуудлыг жижиг хэсгүүдэд хувааж шийдээд нэгтгэх.
Алгоритм	Энгийн рекурсив асуудал шийдэлт.	Хуваах, шийдвэрлэх, нэгтгэх гэсэн үе шаттай.
Програмчлалын хэв маяг	Базын нөхцөл ба рекурсив нөхцөл ашиглана.	Рекурсын тусламжтай асуудлыг хуваах ба нэгтгэнэ.
Жишээ алгоритм	Fibonacci, Factorial	Merge Sort, Quick Sort
Давуу тал	Ойлгоход хялбар, богино код.	Илүү үр ашигтай, том хэмжээний өгөгдөлд тохиромжтой.
Сул тал	Илүү их санах ой зарцуулдаг.	Ойлгоход илүү төвөгтэй байж магадгүй.

2. Divide and Conquer vs Dynamic Programming

Divide and Conquer:

Асуудлыг жижиглэж шийддэг боловч дэд асуудлуудын үр дүнг дахин ашиглахгүй

```
# Divide and Conquer: Recursive Fibonacci
def fibonacci_dc(n):
    if n <= 1:
        return n
    return fibonacci_dc(n - 1) + fibonacci_dc(n - 2)
```

```
# Example usage
n = 10
print("Divide and Conquer:", fibonacci_dc(n))  # Output: 55
```

Dynamic Programming:

Давтагдсан дэд асуудлуудын үр дүнг хадгалж ашигладаг тул илүү үр ашигтай.

```
# Dynamic Programming: Bottom-Up Fibonacci
def fibonacci_dp(n):
    if n <= 1:
        return n
    dp = [0] * (n + 1)
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]

# Example usage
n = 10
print("Dynamic Programming:", fibonacci_dp(n))  # Output: 55
```

Шинж чанар	Divide and Conquer	Dynamic Programming
Зорилго	Асуудлыг дэд хэсгүүдэд хувааж, тус бүрийг шийдвэрлэнэ.	Дэд асуудлуудын давхардсан тооцооллыг хадгалж ашиглана.
Шийдэл	Дэд хэсгүүдийг тус бүрд нь шийдээд хариуг нэгтгэнэ.	Урьдчилан тооцоолсон үр дүнг санах ойд хадгална.
Давталт	Давхардсан тооцоолол гарч болно.	Давхардсан тооцооллыг багасгана.
Хэрэглээ	Merge Sort, Quick Sort, Binary Search	Fibonacci, Shortest Path (Dijkstra, Floyd-Warshall)
Санах ой	Дэд асуудал тус бүрийг шийдэхдээ дахин санах ой ашиглана.	Хадгалагдсан үр дүнг ашиглаж санах ойн хэрэглээг бууруулна.
Алгоритмын бүтэц	Рекурсив аргаар ажиллана.	Memoization (Top-Down) болон Tabulation (Bottom-Up).
Хугацааны нарийн түвшин	Илүү их хугацаа шаарддаг байж болно.	Илүү хурдан бөгөөд оновчтой.

3. Greedy vs Dynamic programming

Асуудлыг шийдвэрлэхэд ашиглаж болох 2 өөр алгоритмын арга.

Greedy нь ихэвчлэн энгийн хэрэгжүүлэхэд хялбар , үр дүнтэй байдаг ч үргэлж хамгийн сайн шийдвэрт хүргэж чаддаггүй. Тухайн мөчид л хамгийн сайн гэсэн шийдвэрээ гаргадаг нь урт

хугацааны үр дүнээ хардаггүй гэсэн үг юм. (Хамгийн бага хүрээтэй мод хамгийн богино замтай алгоритмууд.)

```
def fractional_knapsack(values, weights, capacity):
    ratio = [(values[i] / weights[i], values[i], weights[i]) for i in range(len(values))]
    ratio.sort(reverse=True)  # Өндөр үнэ/жинтэй зүйлсээс эхлэнэ.

    total_value = 0
    for r, v, w in ratio:
        if capacity >= w:
            total_value += v
            capacity -= w
        else:
            total_value += r * capacity
            break

    return total_value

values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print(fractional_knapsack(values, weights, capacity))  # Гаралт: 240.0
```

Динамик програмчлал нь Асуудлыг жижиг дэд асуудал болгон задлаад, дэд асуудал бүрийг зөвхөн нэг удаа шийдэж, тухайн шийдлээ хадгалдаг. Өөрөөр хэлбэл илүү их хугацаа, санах ой шаардаж дэд асуудлуудын үр дүнг ашиглаж том асуудлын оновчтой шийдлийг олдог гэсэн үг юм. (Фибоначчийн дараалал, хамгийн урт нийтлэг дэд дараалал.

```
def knapsack(values, weights, capacity):
    n = len(values)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print(knapsack(values, weights, capacity))  # Гаралт: 220
```