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

Бие даалтын ажлын нэр: **2-р ажил**

Хичээл заасан багш: **Д.Батмөнх**

Бие даалтын ажил гүйцэтгэсэн: **Э.Энх-Амар (B210900802)**

УЛААНБААТАР ХОТ 2024ОН

Хураангуй

1. Divide-and-Conquer
2. Dynamic Programming
3. Greedy Algorithms

Харьцуулалтууд;

1. Recursion vs Divide-and-Conquer
2. Divide-and-Conquer vs Dynamic Programming
3. Dynamic Programming vs Greedy

Тодорхойлолт;

Divide-and-Conquer (Хувааж, Ялах): Том асуудлыг жижиг хэсгүүдэд хувааж, шийдсэний дараа нийлүүлдэг

Dynamic Programming (Динамик Програмчлал): Дахин давтагдах дэд асуудлуудын шийдлийг хадгалж, хамгийн үр дүнтэй шийдлийг олох.

Greedy Algorithms (Хомхойлох аргаар шийдэх): Тухайн үед хамгийн сайн шийдлийг сонгодог.

Харьцуулалтууд:

Recursion vs Divide-and-Conquer: Давталт нь өөрийгөө дуудах бол хувааж, ялах нь асуудлыг хувааж шийддэг.

Divide-and-Conquer vs Dynamic Programming: Хувааж, ялах нь жижиг хэсгүүдэд хувааж шийддэг, харин динамик програмчлал нь дэд асуудлуудын шийдлийг хадгалж дахин тооцохгүйгээр шийддэг.

Dynamic Programming vs Greedy: Динамик програмчлал нь хамгийн тохиромжтой шийдлийг олдог бол Greedy нь тухайн үед хамгийн сайн шийдлийг сонгоно.

Divide-and-Conquer (Хувааж, Ялах аргаар шийдэх)

Жишээ: Merge Sort (Эрэмбэлэх)

Бодлого:

- Дараах 8 тооны жагсаалтыг эрэмбэлэх хэрэгтэй: [38, 27, 43, 3, 9, 82, 10, 5]

Алгоритм:

1. Эхлээд жагсаалтыг хуваана:
    - [38, 27, 43, 3] | [9, 82, 10, 5]
2. Эдгээр хэсгүүдийг дахин хуваана:
    - [38, 27] | [43, 3]
    - [9, 82] | [10, 5]
3. Хуваагдсан жагсаалтыг эрэмбэлнэ:
    - [27, 38] | [3, 43]
    - [9, 82] | [5, 10]
4. Эцсийн байдлаар эдгээр хэсгүүдийг нийлүүлнэ:
    - [3, 27, 38, 43]
    - [5, 9, 10, 82]
5. Төгсгөлийн эрэмбэлэгдсэн жагсаалт:
    - [3, 5, 9, 10, 27, 38, 43, 82]

Dynamic Programming (Динамик Програмчлал)

Жишээ: Fibonacci тоо

Бодлого:

- Фибоначийн 10 дахь тоог олоорой.

Алгоритм:

- Фибоначийн тоо нь дараах томьёогоор илэрхийлэгдэнэ:
    - $F(n) = F(n-1) + F(n-2)$
    - $F(0) = 0, F(1) = 1$

Тоолох алхам:

1. $F(0) = 0, F(1) = 1$
2. $F(2) = F(1) + F(0) = 1 + 0 = 1$
3. $F(3) = F(2) + F(1) = 1 + 1 = 2$
4. $F(4) = F(3) + F(2) = 2 + 1 = 3$
5. $F(5) = F(4) + F(3) = 3 + 2 = 5$
6. $F(6) = F(5) + F(4) = 5 + 3 = 8$
7. $F(7) = F(6) + F(5) = 8 + 5 = 13$
8. $F(8) = F(7) + F(6) = 13 + 8 = 21$
9. $F(9) = F(8) + F(7) = 21 + 13 = 34$
10. $F(10) = F(9) + F(8) = 34 + 21 = 55$

Хариулт: $F(10) = 55$

Greedy Algorithms (Хомхойлох аргаар шийдэх**)**

Жишээ: Coin Change (Зоосны асуудал)

Бодлого:

- 63 төгрөгийг хамгийн бага тооны зоосоор хуваарилах.
- Зоосны хэмжээ: 25, 10, 5, 1

Алгоритм (Greedy):

1. Эхлээд хамгийн том зоосоор хуваана:
    - 63 / 25 = 2 (25 төгрөгийг хоёр зоосоор хувааж, 50 төгрөг гарна)
2. Үлдсэн 63 - 50 = 13 төгрөг.
3. 13 / 10 = 1 (10 төгрөгийг нэг зоосоор хувааж, үлдсэн 3 төгрөг гарна)
4. 3 / 1 = 3 (1 төгрөгийг гурван зоосоор хувааж, үлдсэн зүйлгүй болно)

Эцсийн шийдэл:

- 2 зоос 25, 1 зоос 10, 3 зоос 1
- 63 төгрөгийг хамгийн бага тооны зоосоор хуваарилсан: 2 x 25 + 1 x 10 + 3 x 1 = 63

Харьцуулалтууд;

1. Recursion vs Divide-and-Conquer

- **Recursion**:.
  - Жишээ: Factorial тооцох.

```
python

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Энэ жишээнд, **factorial(n)** тооцоолохдоо өөрийгөө дуудах замаар жижиг дэд асуудлуудыг шийдэж, эцсийн үр дүнд хүрнэ.

- **Divide-and-Conquer**:
  - Жишээ: Merge Sort (Жагсаалтыг эрэмбэлэх).

```
python
def merge_sort(arr):
    if Len(arr) > 1:
        mid = Len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0
        while i < Len(left_half) and j < Len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < Len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < Len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
```

Энд **Merge Sort** нь асуудлыг хувааж, жижиг хэсгүүдийг эрэмбэлж, эцэст нь нийлүүлдэг.

2. Divide-and-Conquer vs Dynamic Programming

- **Divide-and-Conquer**:
  - Жишээ: Merge Sort (Үүн дээр дээр дурдсан жишээг ашиглав).
- **Dynamic Programming**:.
  - Жишээ: Fibonacci Sequence (Memorization):

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

Энд **Fibonacci Sequence** нь өмнөх дэд асуудлуудын шийдлийг хадгалах замаар илүү хурдан шийдлийг олж авдаг.

3. Dynamic Programming vs Greedy

- **Dynamic Programming**:.
  - Жишээ: Knapsack Problem (Багажны хэт их ачаалал авах асуудал).

```
python

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

    for i in range(n + 1):
        for w in range(capacity + 1):
            if i == 0 or w == 0:
                DP[i][w] = 0
            elif weights[i-1] <= w:
                DP[i][w] = max(values[i-1] + DP[i-1][w-weights[i-
1]], dd P[i-1][w])
            else:
                DP[i][w] = DP[i-1][w]
    return DP[n][capacity]
```

Энд **Knapsack Problem** нь өмнөх шийдлүүдийг хадгалж, хамгийн оптималь шийдлийг гаргадаг.

Greedy:

Жишээ: Activity Selection Problem (Үйл ажиллагааны сонголт).

```
python

def activity_selection(start, finish):
    n = Len(start)
    selected_activities = []

    # Өнөөдрийн хамгийн эхний төгсгөлтэй үйл ажиллагааг сонгоно
    i = 0
    selected_activities.append(i)

    for j in range(1, n):
        if start[j] >= finish[i]:
            selected_activities.append(j)
            i = j

    return selected_activities
```

Activity Selection асуудалд **Greedy** нь хамгийн эхний төгсгөлтэй үйл ажиллагааг сонгоод, дараа нь бусад үйл ажиллагаануудтай харьцуулах замаар шийдэл гаргана.

Ашигласан эх сурвалж;

https://www.geeksforgeeks.org/comparison-among-greedy-divide-and-conquer-and-dynamic-programming-algorithm/

https://github.com/breezy-codes/Greedy-Algorithm