ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл холбооны технологийн сургууль

БИЕ ДААЛТЫН АЖЛЫН ТАЙЛАН

Алгоритмын шинжилгээ ба зохиомж (F.CS301)
2024-2025 оны хичээлийн жилийн намрын улирал

"Divide and Conquer, Dynamic programming, Greedy algorithm"

Шалгасан багш: Д. Батмөнх

Гүйцэтгэсэн: B221910027 А. Билгүүн

Улаанбаатар 2024

Divide-and-Conquer

Divide and Conquer нь алгоритмын нэгэн аргачлал бөгөөд томоохон асуудлыг жижиг дэд асуудлуудад хуваан шийдвэрлэхэд ашиглагддаг. Энэ арга нь ихэвчлэн recursive хандлагад суурилсан бөгөөд дараах гурван үндсэн алхмыг дагаж ажилладаг.

Алгоритмын Алхмууд

1. **Хуваах (Divide):**
    - Асуудлыг жижиг, ихэвчлэн ижил хэмжээтэй дэд асуудлуудад хуваана.
    - Эдгээр дэд асуудлууд нь үндсэн асуудалтай төстэй боловч хэмжээ нь бага байна.
2. **Байлдан дагуулах (Conquer):**
    - Хуваагдсан дэд асуудал бүрийг тусад нь шийднэ. Ихэвчлэн recursive байдлаар ажилладаг.
    - Хэрэв дэд асуудлуудын хэмжээ хамгийн бага утгадаа хүрсэн бол шууд шийдлийг олж өгнө.
3. **Нэгтгэх (Combine):**
    - Дэд асуудлуудын шийдлийг нэгтгэн, анхны асуудлын шийдэлд хүргэнэ.
    - Жишээлбэл, массивыг эрэмбэлэх үед жижиг массивуудыг эрэмбэлж, дараа нь нэгтгэн том массив болгоно.

Жишээ: Merge Sort Алгоритм

Merge Sort нь **Хуваах ба Байлдан Дагуулах** аргын сонгодог жишээ юм. Энэ алгоритм массивыг эрэмбэлэхдээ дараах алхмуудыг ашигладаг:

1. **Хуваах:**
    - Оролтын массивыг хоёр хэсэгт хуваана.
    - Жишээ нь: [38, 27, 43, 3, 9, 82, 10] массивыг [38, 27, 43, 3] болон [9, 82, 10] гэж хуваана.
2. **Байлдан дагуулах:**
    - Хуваасан массив бүрийг recursive байдлаар жижиг массив болгож хуваана.
    - Хэрэв массивын хэмжээ 1 эсвэл 0 бол уг массивыг эрэмбэлэгдсэн гэж үзнэ.
3. **Нэгтгэх:**
    - Эрэмбэлэгдсэн жижиг массивуудыг нэгтгэж, том массив болгоно.
    - Жишээлбэл: [27, 38] ба [3, 43] массивыг нэгтгэж [3, 27, 38, 43] болно.

Жишээ хэрэгжүүлэлт:

```
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
    return arr
```

Dynamic Programming (DP)

Динамик программчлал нь давхцаж буй дэд асуудал болон оновчтой дэд бүтэцтэй асуудлыг шийдвэрлэхэд ашигладаг оновчлолын арга юм. Энэ арга нь дэд асуудлыг дахин дахин тооцоолохоос сэргийлж, нэг удаа шийдсэн шийдлийг хадгалж, ирээдүйд ашиглахад чиглэгддэг.

Фибоначчийн Дэс Дараалал

Фибоначчийн дарааллыг энгийн recursive аргаар тооцоолох үед давхацсан тооцоолол их гарч, үр дүнгүй байдлыг бий болгодог. Жишээлбэл, F(n) тооцоолохдоо F(n-1) ба F(n-2)-ийг олон удаа дахин тооцдог бөгөөд энэ нь экспоненциал хугацааны нарийн төвөгтэй байдалд хүргэдэг.

Динамик Программчлалын Шийдэл

1. **Recursive Хамаарал:**

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

   F(0) = 0,

   F(1) = 1.

2. **Массив Ашиглалт:**

   Фибоначчийн утгуудыг массив дотор хадгална.

   Давхардсан тооцооллоос зайлсхийж, өмнө тооцоолсон утгуудыг дахин ашиглана.

Жишээ хэрэгжүүлэлт:

```
function fibonacci(n):

if n <= 1:

return n

dp = array of size (n + 1) initialized to 0

dp[0] = 0

dp[1] = 1

for i from 2 to n:

dp[i] = dp[i - 1] + dp[i - 2]

return dp[n]
```

Greedy algorithm

Хомхойлох Алгоритм

Хомхойлох алгоритм нь оновчтой шийдлийг олохын тулд алхам бүрд тухайн үеийн хамгийн оновчтой сонголтыг хийдэг оновчлолын арга юм. Энэ нь урт хугацааны үр дагаврыг харгалзахгүйгээр "хамгийн сайн хувилбарыг одоо авах" зарчмаар ажилладаг.

Жишээ Бодлого: Fractional Knapsack Problem

Асуудлын Тайлбар:

Таны өгөгдөлд жин ба үнэтэй зүйлсийн багц байна. Цүнхний багтаамж тогтмол бөгөөд тэндээс хамгийн их үнэ цэнийг авахыг зорьдог. Зүйлсийг бүтэн эсвэл бутархай байдлаар авч болно.

Алгоритмын Алхмууд

1. **Жингийн нэгжийн үнэ цэнийг тооцоолох**

   Зүйл бүрийн жингийн нэгжид ногдох үнэ цэнийг тооцоолно.

   Item 1: 60 / 10 = 6

   Item 2: 100 / 20 = 5

   Item 3: 120 / 30 = 4

Харьцаагаар эрэмбэлэх

2. **Жингийн нэгжийн үнэ цэний харьцаагаар зүйлсийг буурах дарааллаар эрэмбэлнэ:**

   Item 1 (харьцаа = 6)

   Item 2 (харьцаа = 5)

   Item 3 (харьцаа = 4)

3. **Цүнхийг дүүргэх**

   Item 1-ийг бүхлээр нь авна.

   Үлдсэн багтаамж: 40, Нийт үнийн дүн: 60

   Item 2-ийг бүхлээр нь авна.

   Үлдсэн багтаамж: 20, Нийт үнийн дүн: 160

   Item 3-ийн 2/3 хэсгийг авна.

   Үлдсэн багтаамж: 0, Нийт үнийн дүн: $160 + 120 \times (2/3) = 220$

Recursion vs Divide-and-Conquer

Recursion гэдэг нь асуудлыг ижил төрлийн жижиг дэд асуудлуудад задалж, өөрийгөө дахин дахин дуудах замаар шийдвэрлэх ерөнхий программчлалын арга юм. Энэ аргыг математик дарааллыг тооцоолох, өгөгдлийн бүтцээр дамжих, хайлт хийх зэрэг олон янзын асуудлуудыг шийдвэрлэхэд ашигладаг. Гэвч recursion нь дэд асуудлуудыг бие даасан байдлаар задлахгүй бөгөөд заримдаа давхацсан тооцоолол үүсгэдэг. Жишээлбэл, Фибоначчийн дарааллыг recursion тооцоолох үед тооцоолол давхцах замаар илүүдэл процесс бий болдог.

Divide-and-Conquer гэдэг нь асуудлыг жижиг, бие даасан дэд асуудлуудад задалдаг recursive хандлагын тодорхой төрөл юм. Энэ аргын хүрээнд дэд асуудлуудыг тус тусад нь шийдэж, дараа нь эдгээр шийдлүүдийг нэгтгэж эцсийн үр дүнг гаргадаг. Divide-and-Conquer аргын сонгодог жишээ бол Merge Sort алгоритм юм. Энэ алгоритмд оролтын массивыг хоёр хувааж, хуваасан хэсэг бүрийг тусад нь эрэмбэлж, эцэст нь эрэмбэлэгдсэн хэсгүүдийг нэгтгэдэг. Энэ бүтэцлэгдсэн арга нь үр ашигтай бөгөөд давхацсан дэд асуудлуудын хувьд илүү үр дүнтэй байдаг.

Divide-and-Conquer vs Dynamic Programming

Divide-and-Conquer ба Динамик программчлал (DP) нь хоёулаа асуудлыг жижиг дэд хэсгүүдэд хувааж шийдэх аргууд юм. Гэхдээ эдгээр аргуудын гол ялгаа нь дэд асуудлуудыг хэрхэн зохицуулахад оршдог. Divide-and-Conquer нь дэд асуудлуудыг бие даан задлан, тус тусад нь шийдэж, дараа нь нэгтгэдэг. Энэ нь дэд асуудлууд давхцахгүй үед үр дүнтэй байдаг.

Динамик программчлал нь давхацсан дэд асуудлууд болон оновчтой дэд бүтэцтэй асуудлуудыг шийдвэрлэхэд зориулагдсан. DP нь дэд асуудлуудыг нэг удаа шийдэж, үр дүнг нь хадгалснаар давхардлыг арилгана. Жишээлбэл, 0/1 үүргэвчний асуудалд DP нь бүх боломжит хослолыг тооцоолохгүйгээр оновчтой шийдлийг хурдан гаргадаг. Ингэснээр тооцоолол илүү хурдан бөгөөд үр дүнтэй болдог.

Greedy vs Dynamic Programming

Хомхойлох алгоритмууд ба Динамик программчлал нь шийдлүүдийг оновчтой болгохыг эрмэлздэг ч шийдвэр гаргах арга барилаараа ялгаатай. Хомхойлох алгоритмууд нь тухайн мөчид хамгийн сайн сонголтыг хийж, эцсийн үр дүнд хүрнэ гэж найддаг. Энэ арга нь "шуналтай сонголтын шинж чанар" бүхий асуудлуудад сайн ажилладаг. Жишээлбэл, үйл ажиллагааны сонголтын асуудалд хомхойлох алгоритм нь хамгийн эрт дуусах үйл ажиллагааг сонгон, оролцох боломжтой хамгийн их үйл ажиллагааг үр дүнтэй олох боломжийг олгодог.

Динамик программчлал нь асуудлын тал бүрийг авч үзэн, дэд асуудлуудын үр дүнг хадгалж, шаардлагатай үед дахин ашигладаг. Энэ арга нь илүү нарийн төвөгтэй асуудлуудад илүү үр дүнтэй бөгөөд оновчтой шийдлийг баталгаажуулдаг. Жишээлбэл, хамгийн урт нийтлэг дэд дараалал (LCS) бодлогод DP нь дэд асуудлуудыг тооцоолж, хүснэгт ашиглан үр дүнг хадгалдаг. Ингэснээр илүү үр ашигтай тооцоолол хийх боломжтой болдог.

Хомхойлох алгоритмууд нь илүү хурдан бөгөөд хэрэгжүүлэхэд хялбар боловч үр дүн нь үргэлж оновчтой байх баталгаагүй. Харин Динамик программчлал нь илүү цаг хугацаа шаарддаг ч илүү нарийвчилсан, найдвартай шийдлийг гаргадаг.

Sources:

1.Ном

● Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms. 3rd ed., MIT Press, 2009.*

2.Online Resources

● GeeksforGeeks."Greedy Algorithm." https://www.geeksforgeeks.org/greedy-algorithms/

● GeeksforGeeks."Dynamic Programming.": https://www.geeksforgeeks.org/dynamic-programming/