**ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ**

**Мэдээлэл холбооны технологийн сургууль**

****

**БИЕ ДААЛТЫН АЖЛЫН ТАЙЛАН**

**Алгоритм шинжилгээ ба зохиомж  (F.CSM301)**

**2023-2024 оны хичээлийн жилийн хавар**

**Хичээл заасан багш:** Д.Батмөнх

**Бие даалт хийж гүйцэтгэсэн:** Оюутан: B221910005 Н.Цацрал

Улаанбаатар хот

2024

**Divide-and-Conquer**

Divide and Conquer алгоритм нь нарийн төвөгтэй асуудлыг жижиг, илүү удирдах боломжтой хэсгүүдэд хувааж, хэсэг бүрийг тус тусад нь шийдэж, дараа нь анхны асуудлыг шийдэхийн тулд шийдлүүдийг нэгтгэдэг асуудлыг шийдвэрлэх стратеги юм. Энэ бол компьютерын шинжлэх ухаан, математикийн шинжлэх ухаанд өргөн хэрэглэгддэг алгоритмын арга юм. Divide-and-conquer алгоритмыг хэсэгчлэх, боловсруулах, нэгтгэх гэсэн гурван үе шатанд хувааж болно.

**1. Хэсэгчлэх(divide):**

-Анхны асуудлыг жижиг дэд асуудал болгон хуваа.

-Дэд асуудал бүр нь ерөнхий асуудлын нэг хэсгийг төлөөлөх ёстой.

Зорилго нь асуудлыг цаашид хуваах боломжгүй болтол нь хуваах явдал юм.

**2.Боловсруулах (conquer):**

-Жижиг дэд бодлого бүрийг тус тусад нь шийдэж дахин дуудах байдлаар дуудна.

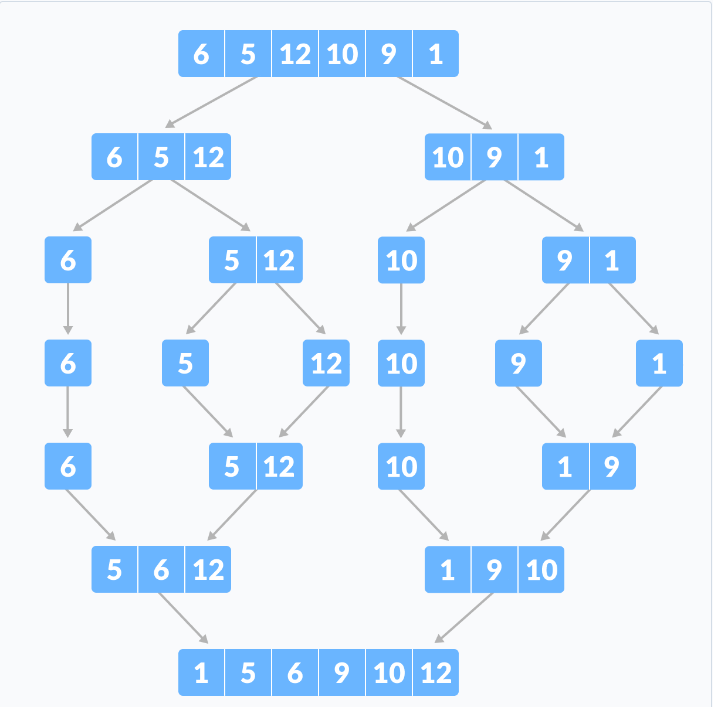
Зорилго нь эдгээр дэд асуудлын шийдлийг олох явдал юм.

**3. Нэгтгэх(combine):**

Бүх асуудлын эцсийн шийдлийг олохын тулд дэд асуудлуудыг нэгтгэнэ.

Жижиг дэд асуудлууд шийдэгдсэний дараа бид том асуудлын шийдлийг олж авахын тулд тэдгээрийн шийдлүүдийг рекурсив байдлаар нэгтгэдэг.Зорилго нь дэд асуудлын үр дүнг нэгтгэх замаар анхны асуудлын шийдлийг томьёолох явдал юм.

Жишээ: Merge sort  алгоритмд элементүүдийн жагсаалтыг эрэмбэлэхийн тулд "Divide and sort" стратегийг ашигладаг. Доорх зурган дээр Merge Sort ашиглан массивыг эрэмбэлэхийн тулд хуваах, нэгтгэх төлөвийг харуулав.



*Зураг 1.1 Merge Sort ашиглан массивыг эрэмбэлэхийн тулд хуваах, нэгтгэх төлөвийг харуулав.*

**Dynamic Programming**

Динамик программчлал нь давхцаж буй дэд асуудлууд болон оновчтой дэд бүтцийн шинж чанартай асуудлуудыг үр дүнтэй шийдвэрлэхэд тусалдаг компьютерын программчлалын техник юм.

Хэрэв аливаа асуудлыг дэд асуудалд хувааж, тэдгээр нь эргээд жижиг дэд асуудалд хуваагдаж, эдгээр нь давхцаж байгаа бол дэд асуудлын шийдлүүдийг цаашид ашиглах зорилгоор хадгалах боломжтой. Ийм байдлаар CPU-ийн үр ашгийг нэмэгдүүлэх боломжтой.Энэхүү шийдвэрлэх аргыг **динамик программчлал** гэж нэрлэдэг.Иймд асуудлууд нь оновчтой шийдлийг олохын тулд ижил дэд асуудлын утгыг дахин дахин тооцоолох явдал юм.

Динамик программчлал нь дэд асуудлын үр дүнг хадгалах замаар ажилладаг бөгөөд ингэснээр тэдгээрийг шийдэлд ашиглах шаардлагатай үед дуудаж ажиллуулахад бэлэн болгож тэдгээрийг дахин тооцоолох шаардлагагүй болно.Дэд асуудлын утгыг хадгалах энэ аргыг санах ой (**memozation**) гэж нэрлэдэг. Массив дахь утгуудыг хадгалснаар өмнө нь тулгарч байсан дэд асуудлуудыг тооцоолох цагийг хэмнэдэг.

**Дээрээс доош (Top-Down)**: Томоос жижиг рүү задарч, шаардлагатай тооцооллыг санах ойд хадгалж, утга нь байвал шууд ашиглана.

Шинэ утга тооцоолох үед map эсвэл массивыг ашиглан тухайн утгыг санах ойд хадгална. Хэрэв өмнө нь тооцоологдсон утга байвал шууд санах ойгоос уншиж авдаг.

Давуу тал: Давтагдсан тооцооллуудыг багасгаж, хурдан ажиллах боломжтой.

Сул тал: Давталт ашиглахгүй тул функц дуудахад **рекурс** (өөрийгөө дуудах) шаардлагатай байдаг. Их хэмжээний n үед стекийн асуудал үүсэх магадлалтай.

Доорх жишээ код нь **дээрээс доош (top-down)** динамик программчлалын аргыг ашиглаж, Фибоначчийн тооцооллыг хийхэд зориулагдсан. Код нь мемоизацийг ашиглаж байгаа бөгөөд аль хэдийн тооцоолсон утгуудыг **map** буюу хадгалах сан ашиглан хадгалж, давтагдсан тооцооллыг багасгах боломжтойг харуулж байна

var m = map(0 → 0, 1 → 1)

function fib(n)

    if key n is not in map m

        m[n] = fib(n − 1) + fib(n − 2)

    return m [n]

Кодын ажиллагаа

1. **map (мэп) нь  0 → 0** болон **1 → 1** гэсэн үндсэн утгуудыг агуулна.
2. Хэрэв **n** түлхүүр **m**-д байхгүй бол **fib(n - 1) + fib(n - 2)** утгыг тооцоолж **m[n]** болгон хадгална.
3. **m[n]**-г буцаана. Энэ нь **n** дэх Фибоначчийн тоо байна.

Энэ аргын гол давуу тал нь нэг удаа тооцоологдсон утгыг **m**-д хадгалснаар давтагдахгүй байх боломжтой, ингэснээр тооцооллын үр ашиг нэмэгддэг.

Алгоритмын ажиллах чиглэлийг өөрчлөх замаар, өөрөөр хэлбэл үндсэн хувилбараас эхлээд шийдэл рүү шилжих замаар бид динамик программчлалыг доороос дээш ажиллуулах боломжтой.

function fib(n)

    if n = 0

        return 0

    else

        var prevFib = 0, currFib = 1

        repeat n − 1 times

            var newFib = prevFib + currFib

            prevFib = currFib

            currFib  = newFib

    return currFib

Дээрх жишээ код дараах байдлаар ажиллана:

1. **n** нь 0 бол шууд 0-г буцаана.
2. Үгүй бол **prevFib** болон **currFib** хувьсагчийг ашиглан Фибоначчийн дарааллын өмнөх хоёр утгыг хадгална.
3. **repeat n - 1 times** давталт нь **prevFib** болон **currFib** утгыг шинэчилж, дараагийн Фибоначчийн тоог олох хүртэл ажиллана.
4. Эцэст нь **currFib** утгыг буцаах бөгөөд энэ нь **n** дэх Фибоначчийн тоо байна

**Доороос дээш (Bottom-Up)**: Багаас том руу явж, бүх дэд асуудлыг эхлээд тооцоолж, хамгийн сүүлийн хариуг гаргана.**prevFib** болон **currFib** гэх мэт хувьсагч ашиглан Фибоначчийн тоонуудыг **давталтаар** (итерац) тооцоолдог. Энэ нь хүснэгтэд утгуудыг хадгалахын оронд зүгээр л хоёр хувьсагчийг шинэчилж яваад хамгийн сүүлд шаардлагатай хариуг буцаана.

Давуу тал**:** Давталтаар шийддэг тул стекийн ачаалал бага, санах ойг бага ашиглана.

Сул тал:Бүх дэд асуудлуудыг эхлээд тооцоолж, хамгийн сүүлд шаардлагатай хариуг буцаадаг тул зарим тохиолдолд илүү тооцоолол хийх шаардлага гарч болзошгүй.

Different Types of Dynamic Programming Algorithms

Longest Common Subsequence

Floyd-Warshall Algorithm

Greedy algorithm (хомхойлох алгоритм)

Хомхойлох алгоритм нь одоо байгаа хамгийн сайн хувилбарыг сонгох замаар асуудлыг шийдвэрлэх арга юм. Сонголт буруу байсан ч алгоритм нь өмнөх шийдвэрийг хэзээ ч буцаадаггүй. Энэ нь дээрээс доош (top-down )чиглэсэн байдлаар ажилладаг.

Greedy алгоритм нь үргэлж оновчтой шийдэл өгдөггүй бөгөөд зөвхөн тодорхой асуудалд л зөв шийдэл болдог. Шуурхай шийдвэр гаргалт (локал оновчлол) нь эцсийн оновчтой хариуг өгөх эсэх нь асуудлаас хамаарна. Гэсэн хэдий ч, хэрэв асуудал дараах шинж чанартай байвал алгоритмыг ямар нэгэн асуудалд ашиглаж болох эсэхийг тодорхойлж болно.

1. **Локал оновчтой сонголт хийх**: Тухайн алхамд хамгийн сайн, ашигтай, эсвэл зардал багатай сонголтыг хийнэ.

Нэгэнт сонгосон өмнөх алхмуудыг дахин авч үзэхгүйгээр алхам бүртээ хамгийн сайн сонголтыг сонгосноор асуудлыг шийдэх оновчтой шийдлийг олж чадвал хомхойлох аргыг ашиглан асуудлыг шийдэж болно.

2. Optimal Substructure

Хэрэв асуудлын оновчтой ерөнхий шийдэл нь түүний дэд асуудлын оновчтой шийдэлтэй тохирч байвал асуудлыг хомхойлох аргыг ашиглан шийдэж болно. Энэ шинж чанарыг оновчтой дэд бүтэц гэж нэрлэдэг.

**Хомхойлох алгоритмын давуу тал**

Алгоритмыг тайлбарлахад илүү хялбар байдаг.

Алгоритмын гүйцэтгэл хурдан бөгөөд ихэнх Greedy алгоритмууд O(nlogn) эсвэл O(n) цагийн нарийвчлалтай байдаг.

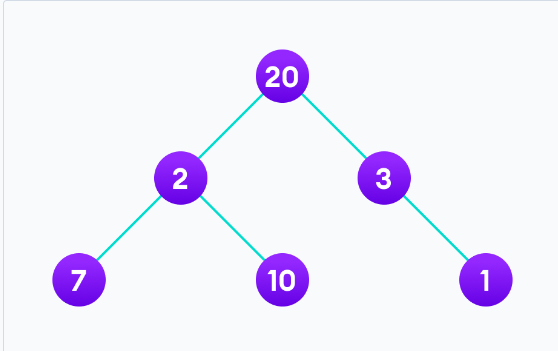
Энэ алгоритм нь бусад алгоритмуудаас илүү сайн ажиллах боломжтой (гэхдээ бүх тохиолдолд биш).

**Хомхойлох алгоритмын сул тал**

Дээр дурдсанчлан, шунахай алгоритм нь үргэлж оновчтой шийдлийг гаргаж чаддаггүй. Энэ бол алгоритмын гол сул тал юм.

Хэрэглэх хүрээ хязгаарлагдмал: Зөвхөн локал оновчлол глобал оновчлолыг хангаж чаддаг асуудлуудад л тохиромжтой.

Жишээлбэл, бид доорх графикаас үндэснээс навч хүртэлх хамгийн урт замыг олохыг хүсэж байна гэж бодъё. Энд хомхойлох алгоритмыг ашиглая.



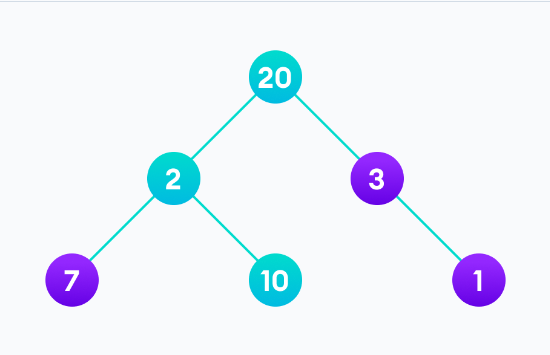
Хомхойлох алгоритм:

1. Үндэс зангилаа 20-оор эхэлье. Баруун талын навч зангилааны жин 3, зүүн навч зангилааны жин 2 байна.

2. Бидний шийдэх гол асуудал бол хамгийн том замыг олох явдал юм. Тиймээс, одоогийн сонголтын  оновчтой шийдэл нь 3. Тиймээс, шунахай алгоритм нь 3-ыг сонгох болно.

3. Эцэст нь 3 аас цааш цорын ганц навч зангилааны жин 1. Энэ нь бидний эцсийн үр дүн 20 + 3 + 1 = 24 болно.

Гэсэн хэдий ч энэ нь оновчтой шийдэл биш юм. Доорх зурган дээр үзүүлсэн шиг илүү их жинтэй (20 + 2 + 10 = 32) өөр нэг зам бий.



Иймд  хомхойлох алгоритм нь үргэлж  оновчтой/боломжтой шийдлийг өгдөггүйг харж болно.

**Хомхойлох алгоритм (Greedy algorithm)**

Эхлэхийн тулд шийдлийн багц (хариултуудыг агуулсан) хоосон байна.

Алхам бүрд шийдэлд хүрэх хүртэл шийдлийн багцад нэг зүйл нэмэгддэг.

Хэрэв шийдлийн багц боломжтой бол одоогийн зүйл хадгалагдана.

Үгүй бол тухайн зүйлээс татгалзаж, дахин хэзээ ч авч үзэхгүй.

Одоо энэ алгоритмыг ашиглан асуудлыг шийдэж үзье.

Problem: You have to make a change of an amount using the smallest possible number of coins.

Amount: $18

Available coins are

  $5 coin

  $2 coin

  $1 coin

There is no limit to the number of each coin you can use.

Шийдэл:

Хоосон шийдлийн багц үүсгэнэ = { }. Боломжтой зоос нь {5, 2, 1} байна.

Бид нийлбэр = 18-ыг олох ёстой. Нийлбэр = 0 гэж эхэлье.

Нийлбэр > 18 болтол хамгийн их утгатай (жишээ нь 5) зоосыг үргэлж сонгоно. (Алхам бүрд хамгийн том утгыг сонгохдоо бид хүссэн шийдэлдээ илүү хурдан хүрнэ.

Эхний давталтад шийдлийн багц = {5} ба нийлбэр = 5 байна.

Хоёр дахь давталтаар шийдлийн багц = {5, 5} ба нийлбэр = 10 байна.

Гурав дахь давталтаар шийдлийн багц = {5, 5, 5} ба нийлбэр = 15.

Дөрөв дэх давталтаар шийдлийн багц = {5, 5, 5, 2} ба нийлбэр = 17. (Бид энд 5-ыг сонгох боломжгүй, учир нь нийлбэр = 20 нь 18-аас их байна. Тиймээс бид 2-ыг сонгоно. хамгийн том нь 2.)

Үүний нэгэн адил тав дахь давталтдаа 1-ийг сонго. Одоо нийлбэр = 18, шийдлийн багц = {5, 5, 5, 2, 1}.

Хомхойлох алгоритмын төрлүүд

-Selection Sort

-Knapsack Problem

-Minimum Spanning Tree

-Single-Source Shortest Path Problem

-Job Scheduling Problem

-Prim's Minimal Spanning Tree Algorithm

-Kruskal's Minimal Spanning Tree Algorithm

-Dijkstra's Minimal Spanning Tree Algorithm

-Huffman Coding

-Ford-Fulkerson Algorith

**Recursion vs Divide-and Conquer**

Энэ хоёр алгоритм нь хэдий  асуудлыг дэд асуудлуудад хувааж шийддэг аргууд боловч өөр өөрсдийн ялгаатай шинж чанаруудтай.

|  |  |  |
| --- | --- | --- |
| **Шинж чанар** | **Recursion** | **Divide-and-Conquer** |
| Үндсэн зарчим | Асуудлыг өөрийгөө дуудаж шийддэг | Асуудлыг тодорхой хэсгүүдэд хувааж,нэгтгэдэг |
| Хэрэглээ | Энгийн дэд асуудалтай логикт тохиромжтой | Илүү том нарийн бүтэцтэй асуудалд тохиромжтой |
| Жишээ алгоритм | Фибоначчи, факториал | Merge Sort, Quick Sort, Binary Search |
| Үр дүн гаралтын хурд | Хялбар алгоритмуудад тохиромжтой | Илүү үр дүнтэй, логарифм хурдтай байж чадна |
| Дараалал хийгдэх алхмууд | Зөвхөн өөрийгөө дуудаж үр дүнд хүрдэг | Хэсэгчлэх,хуваах, нэгтгэх гэсэн тодорхой алхмуудтай |

Divide-and-Conquer арга нь асуудлыг тодорхой зарчмаар хэд хэдэн дэд асуудалд хувааж, эдгээрийг тусад нь шийдээд дараа нь нэгтгэж эцсийн хариуг олох аргыг хэлнэ.Recursion буюу дахин дуудах аргачлал  нь функц өөрийгөө дотроосоо дахин дуудах зарчмаар бодлогыг хуваан боддог. Энэ арга нь том асуудлыг илүү жижиг хувилбар болгон шийдэхдээ өөрийгөө ашигладаг.

**Divide-and Conquer vs Dynamic Programming**

**Dynamic Programming** нь асуудлыг шийдэхэд санах ойг ашиглан өмнө тооцоолсон хариуг хадгалахад чиглэдэг бол **Divide-and-Conquer** нь асуудлыг олон дэд асуудалд задалж, тэдгээрийг нэгтгэхэд чиглэдэг.

|  |  |
| --- | --- |
| **Divide-and conquer** | **Dynamic Programming** |
| Дэд асуудлууд нь хоорондоо давхцахгүй | Давхацсан дэд асуудлуудтай байх боломжтой |
| Санах ойн хадгалалт шаардлагагүй | Өмнөх хариуг хадгалж,дахин ашиглана. |
| Хуваах нэгтгэх замаар ажиллана. | Мемоизаци ашиглан хэрэгжүүлнэ |
| Merge sort,Quick sort,Binary Search | Knapsack,Фибоначчи,Floyd-Warshall |

**Dynamic Programming** (Динамик программчлал)  Санах ойг ашиглан өмнө тооцоолсон хариуг хадгалж, дахин тооцоолол хийхээс сэргийлдэг тул үр дүнтэй. Энэ аргыг ихэвчлэн (bottom-up) болон **мемоизаци** (top-down) хандлагуудаар хэрэгжүүлдэг. **Divide-and-Conquer** Асуудлыг хэд хэдэн жижиг дэд асуудалд хувааж, тус бүрийг нь шийдвэрлээд эцэст нь нэгтгэж, анхны асуудлын хариуг олдог.

**Dynamic Programming vs Greedy**

|  |  |
| --- | --- |
| **Динамик программчлалын арга** | **Хомхойлох арга** |
| Асуудлуудыг дэд асуудлуудад хувааж шийднэ. | Алхам бүрт оновчтой сонголт хийнэ |
| Өмнөх хариуг хадгалж дахин ашиглана. | Санах ойд хадгалалт шаардлагагүй. |
| Бүх боломжийг тооцоолж хамгийн оновчтойг сонгоно. | Локал хамгийн сайн сонголт дээр тулгуурлана. |
| Бүх тохиолдолд оновчтойг сонгоно. | Зөвхөн зарим асуудлуудад оновчтой шийдэл өгнө. |
| Кнапсак,Longest Common Subsequence | Dijkstra(Shortest path),Prim’s,Kruskal’s(MinimumSpanning Tree) |

Dynamic Programming нь давхацсан дэд асуудалтай, хамгийн оновчтой шийдлийг шаардсан тохиолдолд тохиромжтой.

Greedy Algorithm нь тухайн үеийн хамгийн ашигтай сонголтыг хийж, эцсийн хариуг хурдан гаргах шаардлагатай, мөн локал оновчлол нь глобал оновчтой шийдэлд хүргэх асуудлуудад тохиромжтой.

кодын хэрэглээндээр яаж ялгах вэ  
  
яаж таних вэ

Ашигласан материал

* Introduction to Algorithms (Fourth Edition) by Thomas H.Cormen, Charles E. Leiserson, Ronald L.Rivest, Clifford Stein
* EITCA Academy <https://mn.eitica.org>
* Алгоритм шинжилгээ ба зохиомж F.CS301 хичээлийн лекцийн материал