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 (Магистр)

Бие даалт хийж гүйцэтгэсэн: Оюутан: B221910039 Б.Баярмагнай

2024 он

---

1. Divide-and-Conquer (Хувааж захирах арга)

- Хувааж захирах (Divide-and-Conquer) арга нь асуудлыг жижиг хэсгүүдэд хувааж, тэдгээрийг тус тусад нь шийдээд, эцэст нь шийдлүүдийг нэгтгэж эцсийн хариуг гаргах аргын нэг юм. Гол санаа нь асуудлыг илүү жижиг, амархан шийдэгдэх хэсгүүдэд хуваах явдал юм.

Алгоритмын бүтэц:

- Асуудлыг илүү жижиг хэсгүүдэд хуваана.
- Хуваасан хэсэг бүрийг бие даан шийднэ.
- Шийдлийг нэгтгэнэ.

Жишээ : Binary Search (Хоёртын хайлт)

- Хоёртын хайлт нь эрэмбэлэгдсэн массивт элемент хайх үед ашигладаг арга юм. Массивыг дундуур нь хуваагаад, хайж буй элементийг дундах элементтэй харьцуулж, аль талд байгааг олж тодорхойлно. Энэ аргаар массивын хэмжээг үргэлж хоёр дахин багасгасаар байна.

Жишээ:

Function mergeSort(array):

- 1. Зогсоох нөхцөл: хэрэв массивын урт 1 буюу түүнээс бага бол буцаана
  if length(array) ≤ 1:
  return array

- 2. Хуваах алхам(divide): массивыг дундуур нь хуваана
  mid = length(array) / 2
  leftHalf = array[0:mid]
  rightHalf = array[mid:length(array)]

- 3. Боловсруулах алхам(conquer): зүүн болон баруун хэсгүүдийг тус бүрд нь рекурсив байдлаар эрэмбэлнэ
  sortedLeft = mergeSort(leftHalf)
  sortedRight = mergeSort(rightHalf)

- 4. Нэгтгэх алхам(merge): эрэмбэлэгдсэн зүүн болон баруун хэсгүүдийг нэгтгэнэ
  return merge(sortedLeft, sortedRight)

Function merge(left, right):
sortedArray = []

- Зүүн болон баруун хэсгүүдэд элемент байсаар байгаа тохиолдолд
  while length(left) > 0 and length(right) > 0:
- Хоёр хэсгийн эхний элементийг харьцуулж, бага утгыг сонгоно

if left[0] ≤ right[0]:
append(sortedArray, left[0])
left = left[1:] - Зүүн хэсгээс сонгосон элементээ устгана
else:
append(sortedArray, right[0])
right = right[1:] - Баруун хэсгээс сонгосон элементээ устгана

- Үлдсэн элементүүдийг нэмнэ (аль нэг нь хоосон бол шууд үлдсэн хэсгийг нэмэх)
sortedArray += left
sortedArray += right

return sortedArray

---

2. Dynamic Programming (Динамик программчлал)

- Динамик программчлал нь дахин давтагдах тооцооллыг багасгах зорилгоор аливаа асуудлыг жижиг дэд асуудалд хувааж, дэд асуудлуудын шийдлийг хадгалах арга юм. Энэ аргаар үр дүнг дахин ашиглах боломжтой болгодог тул хурдан тооцоолол хийхэд тусалдаг.

Top-down

- Дээшээс доош арга нь динамик программчлалын аргачлалын нэг бөгөөд асуудлыг рекурсив байдлаар хамгийн том асуудлаас жижиг дэд асуудлууд уруу хуваах зарчим дээр суурилдаг.

Bottom-up

- Доороос дээш арга нь динамик программчлалын нэг аргачлал бөгөөд том асуудлыг багаас нь эхлэн бүх дэд асуудлыг шийдэж, дээд төвшний эцсийн хариулт уруу дэвшин ажилладаг.
- Bottom-Up нь интерактив буюу давталтын аргаар боддог тул их хэмжээний рекурсив дуудлагыг шаарддаггүй.

Алгоритмын бүтэц :

- Асуудлыг жижиг дэд асуудлуудад хуваана.
- Дэд асуудлын шийдлүүдийг хадгалж, давтагдах тохиолдолд хадгалсан үр дүнг ашиглана.
- Шийдлүүдийг нэгтгэн үндсэн асуудлыг шийднэ.

Жишээ : Fibonacci тоо олох

- Фибоначчийн тоонууд нь өмнөх хоёр тооны нийлбэрээр тодорхойлогддог. Жишээ нь, \(F(n) = F(n-1) + F(n-2)\) гэсэн томьёогоор тодорхойлогдоно. Динамик программчлалаар урьд өмнө бодож олсон утгуудыг хадгалах замаар тооцооллыг хурдлуулна.

3. Greedy Algorithms (Шунахай алгоритм):

- Шунахай алгоритм нь тухайн мөчид хамгийн сайн сонголтыг сонгох зарчмаар ажилладаг. Энэ аргын үед шийдвэр гаргахдаа зөвхөн тухайн мөчийн хамгийн ашигтай шийдлийг сонгодог тул бүхэл бүтэн шийдлийг олдоггүй ч богино хугацаанд шийдвэр гаргахад тохиромжтой байдаг.

Алгоритмын бүтэц :

- Тухайн мөчид хамгийн сайн сонголтыг сонгоно.
- Сонгосон шийдэлд тулгуурлан дараагийн сонголт уруу шилжинэ.
- Процессыг төгсөх хүртэл давтана.

Жишээ : Coin Change Problem (Зоосны солилцоо)

- Тодорхой зоосны утгуудын тусламжтайгаар тодорхой дүнг хэрхэн хамгийн цөөн тооны зоосоор солих вэ гэдэг асуудал. Шунахай аргаар хамгийн их утгатай зоосыг хамгийн түрүүнд сонгох замаар асуудлыг шийднэ.

Дүгнэлт : Өөр өөр нөхцөлд тохиромжтой ба тодорхой асуудлыг шийдэхдээ зөв аргыг сонгох нь үр дүнтэй алгоритмыг хөгжүүлэхэд чухал үүрэгтэй.

4. Recursion vs Divide-and-Conquer

Recursion (Давталт) :

- Рекурс нь өөрийгөө дуудаж асуудлыг шийдэх арга юм. Программчлалд рекурс нь функц өөрийгөө дуудаж, асуудлыг жижиг хэсгүүдэд хуваан шийддэг арга юм. Рекурс нь бага зэрэг тооцоолол шаардсан асуудлуудыг шийдэхэд тохиромжтой.

Divide-and-Conquer (Хувааж захирах) :

- Хувааж захирах арга нь асуудлыг жижиг хэсгүүдэд хувааж, тус бүрийг бие даан шийдээд, дараа нь шийдлүүдийг нэгтгэж асуудлыг шийдэх арга юм. Энэ арга нь рекурсив байдлаар хэрэгжих боловч үр дүнг нэгтгэх үйл явцын нэмэлт алхмуудыг агуулдаг. Рекурс ба хувааж захирах нь зарим талаар адил боловч, хувааж захирах арга нь илүү бүтцийн хувьд төвөгтэй асуудалд ашиглагддаг.

Жишээ :Факториал олох:

- Рекурс ашиглахын нэг жишээ бол факториал олох явдал юм. Жишээ нь, **\(n! = n \times (n-1)! \)** гэх мэтээр функц өөрийгөө дуудаж хариуг олдог.
- **Quick Sort :** Хувааж захирах аргыг **Quick Sort** алгоритмд ашигладаг. Энэ аргаар массивыг дундуур нь хувааж, дэд массив бүрийг тусад нь эрэмбэлээд, эцэст нь бүх массивыг нэгтгэнэ.

Дүгнэлт : Рекурс нь бага хэмжээний давталт шаардсан асуудлуудад тохиромжтой бол, хувааж захирах нь томоохон асуудлыг шийдэхэд ашиглагдах илүү том бүтэцтэй арга юм.

5. Divide-and-Conquer vs Dynamic Programming

Divide-and-Conquer :

- Хувааж захирах аргаар асуудлыг жижиг дэд хэсгүүдэд хуваагаад, бие даан шийдвэрлэж, дараа нь нэгтгэдэг. Энэ нь давтагдсан тооцоолол шаарддаггүй, тус бүрийг нэг удаа л тооцдог тул дэд асуудлууд хоорондоо хамааралгүй байна.

Dynamic Programming (Динамик программчлал) :

- Динамик программчлал нь давтагдах асуудлуудын шийдлийг хадгалж, дахин ашиглах аргыг ашигладаг. Энэ нь давтагдах тооцооллыг багасгах бөгөөд дэд асуудлуудын хамаарлыг багасгаж хурдан тооцоолол хийдэг.

Жишээ :

- Merge Sort (Хувааж захирах) : Merge Sort нь массивыг жижиг хэсгүүдэд хуваагаад, тус бүрийг нь эрэмбэлж, эцэст нь нэгтгэн эрэмбэлсэн массив гаргадаг. Энэ аргаар дэд асуудлуудын тооцоолол нь нэг удаагийн хуваалтын дараа алга болдог.
- Fibonacci (Динамик программчлал) : Фибоначчи тоог тооцоолохдоо динамик программчлал ашиглавал өмнөх утгуудыг хадгалах замаар тооцооллыг хурдлуулдаг.

Дүгнэлт : Хувааж захирах нь дэд асуудлууд хоорондоо хамааралгүй үед ашиглагдах бол динамик программчлал нь дэд асуудлууд дахин давтагдаж тохиолдох үед тохиромжтой.

6. Dynamic Programming vs Greedy

Dynamic Programming :

- Динамик программчлал нь дэд асуудлуудын шийдлүүдийг хадгалж, дахин ашиглах аргаар бүхэл асуудлыг шийддэг. Энэ аргыг ашиглах үед бүхэлдээ хамгийн зөв шийдэл гарах нөхцөлийг хангах боломжтой.

Greedy Algorithms :

- Шунахай алгоритм нь тухайн мөчид хамгийн их ашигтай шийдлийг сонгож, эцсийн шийдэлд хүрэхийг зорьдог. Энэ нь хурдан, бага тооцоолол шаарддаг боловч зарим тохиолдолд хамгийн зөв шийдэлд хүрэхгүй байж болзошгүй.

Жишээ :

- Shortest Path (Dynamic Programming) : Floyd-Warshall алгоритм нь динамик программчлалаар графын хамгийн богино замыг бүх хос цэгийн хооронд олох арга юм.
- Coin Change Problem (Greedy) : Шунахай аргаар зоосны солилцооны асуудлыг шийдэх үед хамгийн их утгатай зоосыг эхлээд сонгодог. Гэхдээ энэ нь бүх нөхцөлд хамгийн бага тооны зоос ашиглах шийдлийг өгч чадахгүй байж болзошгүй.

Дүгнэлт : Динамик программчлал нь илүү зөв шийдлийг өгдөг боловч тооцоолол өндөр байх магадлалтай, харин шунахай арга нь хурдан, энгийн боловч зөв шийдэлд хүрэх баталгаа байхгүй.

Шинж чанар	**Divide-and-Conquer (Хувааж захирах арга) Тооцоолох**	**Динамик программчлал**
Дэд асуудлууд	Бие даасан	Давхацсан
Шийдэл нэгтгэх	Тусдаа шийдлүүдийг нэгтгэнэ	Өмнө бодсон шийдлийг дахин ашиглана
Арга	Давхарлан хуваана	Давталттай эсвэл хадгалалттай ашиглана
Хадгалалт	Ерөнхийдөө хадгалалт шаарддаггүй	Дэд асуудлын шийдлийг хадгалах шаардлагатай
Үр ашиг	Давхардсан тооцооллоос үүдэн удаан байж болно	Шийдлийг хадгалснаар илүү хурдан байдаг
Түгээмэл хэрэглээ	Эрэмбэлэлт, хайлт, матрицын тооцоолол	Оновчлолын асуудлууд (Knapsack, зам гэх мэт)

Dynamic programming vs Greedy algorithms

- Динамик программчлал болон **Хомхойлох(greedy)** алгоритмууд нь оновчлолын асуудлуудыг шийдэх аргууд юм. Хоёулаа оновчтой шийдлийг олох зорилготой боловч хандлага, хэрэглээ нь өөр байдаг.

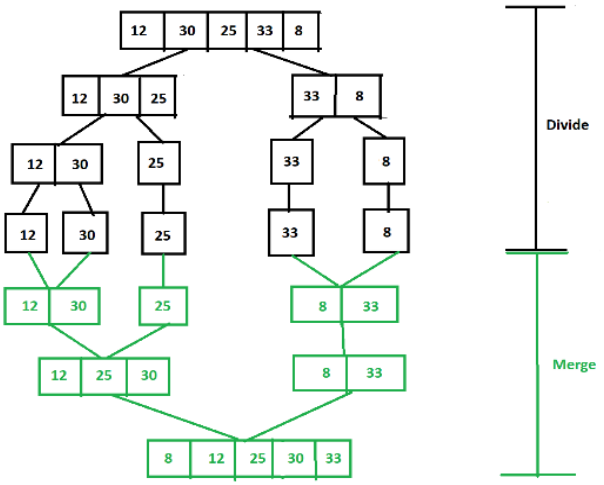
Ямар үед аль аргыг ашиглах вэ?

- **Динамик программчлалыг** дэд асуудлууд давхцаж, өмнөх шийдлүүдийг ашиглах шаардлагатай үед ашиглана. Оновчлолын асуудлуудад тохиромжтой бөгөөд тухайн үеийн сонголтууд нь оновчтой шийдэлд хүрэхгүй үед хэрэглэнэ.
- **Хомхойлох аргыг** тухайн үеийн хамгийн сайн сонголтыг хийх замаар оновчтой шийдэлд хүрч болох үед хэрэглэнэ. Энэ нь хурдан бөгөөд үр ашигтай байдаг, зөвхөн тухайн асуудал нь ийм шийдлийг зөвшөөрсөн нөхцөлд хэрэглэнэ.

Хуваах-Тооцоолох-Нэгтгэх алгоритм

Алхмууд:

- **Хуваах**: Асуудлыг жижиг дэд асуудлууд болгон хуваана.
- **Тооцоолох**: Бүх дэд асуудлыг рекурс аргаар шийднэ. Хэрэв дэд асуудал маш жижиг бол шууд шийдэж болно.
- **Нэгтгэх**: Дэд асуудлуудын шийдлүүдийг нэгтгэж эх асуудлын шийдэлд хүрнэ.



Зураг 1. Merge sort -ыг Divide-and-Conquer аргаар тооцоолох.

Divide and conquer аргын давуу талууд:

- Бие даасан дэд асуудлуудтай асуудлуудыг үр дүнтэй шийддэг.
- Том асуудлыг удирдах боломжтой дэд асуудлуудад хуваасаар нарийн төвөгтэй байдлыг багасгадаг.

Сул талууд:

- Зардал: **Divide and conquer** алхам бүрд рекурс функц дуудлага ашигладаг тул ой санамж нэмэгдэнэ.
- Нэгтгэлд хамааралтай: Шийдлийг нэгтгэх үйл явц ихэнхдээ нэмэлт тооцоо шаарддаг

Шинж чанар	**Хомхойлох алгоритм**	**Динамик программчлал**
Шийдвэрийн хандлага	Тухайн үед хамгийн сайн санагдах шийдлийг сонгодог	Memoization буюу Tabulation ашиглан дэлгэрэнгүй шийдлийг хайдаг
Уян хатан байдал	Зөвхөн тухайн үеийн шийдэл бүр оновчтой байх үед л ашиглагдана	Илүү олон төрлийн асуудалд тохиромжтой
Хадгалалт	Өмнөх шийдлүүдийг хадгалах, ашиглах шаардлагагүй	Дэд асуудлуудын шийдлийг хадгалах шаардлагатай
Үр ашиг	Ерөнхийдөө хурдан, гэхдээ зөвхөн тухайн үед оновчтой шийдэлд хүрдэг. Хамгийн оновчтой шийдэлд хүрэхгүй ч байж болно	Хомхойлох алгоритмтай харьцуулахад удаан боловч оновчтой шийдэлд хүрдэг
Түгээмэл хэрэглээ	Үйл ажиллагаа сонгох, хамгийн богино зам, холбогдсон мод	Үүргэвчний асуудал, хамгийн урт дэд дараалал

Дүгнэлт : Үндсэндээ гол ялгаа нь Хомхойлох алгоритм нь өмнөх шийдвэрүүдийг дахин харгалзахгүйгээр, зөвхөн тухайн үед хамгийн сайн сонголтыг хийдэг бол Динамик аргачлал нь өмнөх тооцоолсон үр дүнгээ хадгалж одоогийн шийдвэртэй харьцуулж оновчтой шийдлийг гаргаж чаддаг.

Ашигласан материал:

- **https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/**
- **https://sharavaa.blogspot.com/2013/09/n.html**
- **https://www.slideshare.net/badralkhurelbaatar5/ucs101-3**
- **https://www.geeksforgeeks.org/greedy-algorithms/**
- **https://www.geeksforgeeks.org/divide-and-conquer/**