Бие Даалт 2

Сэдвүүдийг судалж, өөрийн ойлгосноо жишээ бодлогоор тайлбарлах.

1. Divide and Conquer:

- Энэ арга нь асуудлыг жижиг хэсгээр хуваан шийдвэрлэсээр гол асуудлыг шийдэх аргад хүрнэ.

Жишээ нь: Merge Sort алгоритм эрэмбэлэх массивыг нэг хувьсагчтай болтол хувааж түүнийгээ эрэмбэлэн нэгтгэж гаргадаг.

Эх сурвалж: https://www.youtube.com/watch?v=3j0SWDX4AtU

2. Dynamic Programming

- Энэ арга нь дэд асуудлуудын шийдлийг хадгалан тэднийг дахин шийдэхийг багасгаж гол асуудлын шийдэлд хурдан хүргэнэ.

Жишээ нь: Fibonacci-ын жагсаалтын тоог олохдоо эхний гишүүнүүдийн хариуг хадгалж, дараагийн гишүүнүүд олохдоо хадгалсан хариугаа ашиглаж олдог.

Эх сурвалж: https://www.youtube.com/watch?v=Hdr64lKQ3e4

3. Greedy Algorithm

- Энэ арга нь тухайн асуудлын хамгийн сайн шийдэл эсвэл хамгийн хямд гэж үзэж болохуйц сонголтыг хийдэг. Заримдаа хангалттай алхмаас илүү алхмыг сонгож болно.

Жишээ нь: Knapsack асуудлын хувьд, шуналтай аргаар хамгийн өндөр үнэ-жин харьцаатай зүйлсийг сонгох замаар шийдэлд хүрэх боломжтой.

Эх сурвалж: F.CS301_Lecture.pdf 119-р хуудас

Харьцуулалт.

1. Recursion vs Divide and Conquer

Recursion нь функц өөрийгөө дуудаж, асуудлыг жижиг холбоостой хувилбаруудад хувааж шийдвэрлэх арга юм.

Divide and Conquer нь асуудлыг бүрэн тусгаарласан дэд асуудлууд болгон хувааж, тэдгээрийг тус тусад нь шийдвэрлэн дараа нь нийлүүлж шийдвэрлэх арга.

Жишээ нь: Merge Sort алгоритм рекурси ашиглан жагсаалтыг хуваадаг ч, тэдгээр жагсаалтууд бүрэн тусгаарлагдсан ба ямар ч жижиг холбоос байхгүйгээр шийдвэрлэгдэн шийдвэрлд хүрдэг.

Эх сурвалж: “recursion vs divide and conquer” гэж Google.com дээр хайхад олдсон эхний илэрц

2. Divide and Conquer vs Dynamic Programming

Divide and Conquer нь хуваагдсан дэд асуудлуудыг шийдвэрлэдэг гэхдээ дэд асуудлын хариуг хадгалдаггүй.

Dynamic Programming дахин давтагддаг дэд асуудлуудыг шийдэн хариуг хадгалан түүнийгээ дахин хэрэглэдэг.

Жишээ нь: Фибоначчийн тоог олохдоо рекурси ашиглан асуудлыг жижигрүүлдэг. Гэхдээ энэ аргаар тооцоолоход ижил тооцооллыг олон удаа давтдаг тул үр дүнгийн хувьд удаан болдог. Динамик программчлалаар Фибоначчийн дарааллыг тооцоолохдоо давхар давтагдлыг багасгаж, өмнө тооцоолсон утгуудыг хадгалан дахин ашигладаг. Энэ нь илүү үр дүнтэй, хурдан шийдэлд хүргэдэг.

Эх сурвалж: https://www.youtube.com/watch?v=Hdr64lKQ3e4

3. Dynamic Programming vs Greedy Algorithms

Динамик программчлал нь бүх боломжийг тооцон, хадгалсан үр дүнг ашиглах замаар оновчтой шийдэлд хүрдэг.

Шуналтай алгоритм нь тухайн үеийн хамгийн сайн шийдэл гэж үзэх сонголтыг хийх боловч, заримдаа эцсийн шийдэл нь хамгийн оновчтой бус байж болзошгүй.

Жишээ: Knapsack асуудлыг динамик программчлалаар шийдэж болно, харин шуналтай аргаар зарим нэг хувилбарт (жишээ нь, хагас Knapsack) шийдэлд хүрэх боломжтой.

Эх сурвалж: https://www.youtube.com/watch?v=Hdr64lKQ3e4, F.CS301_Lecture.pdf 119-р хуудас