COURSE : FUNDAMENTALS OF COMPUTER SCIENCE

LECTURE 11: "ABSTRACT DATA TYPES "

Instructor:
PhD, Associate Professor
Tuyatsetseg Badarch

Хийсвэр өгөгдлийн төрлүүд

Энэ хичээлээр бид хийсвэр өгөгдлийн төрлүүдийг (ADT) авч үзэх бөгөөд эдгээр нь дээд түвшний өгөгдлийн төрлүүд юм. Судлах зүйл
ADT нь өгөгдлийг хэрэгжүүлэх бүтэц болох тухай
ADT-ийн тухай товч мэдээлэл
ADT-ийн загвар
Өгөгдлийг үр ашигтай ашиглах боломжтой байдлаар хадгалах буюу зохион байгуулахыг хэлнэ. ADT - Өгөгдлийн бүтэц жишээ : Хүснэгт (Array), Холбоос (LinkedList), Стак (Stack), Дараалал (Queue), Мод (Tree), Граф (Graph), Хэш хүснэгт (Hash Table) гэх мэт байдаг.

Өгөгдлийн төрлийг өөрчилсөн байдлаар нь /Derived Data Type/

- Жагсаалт /List/,
- Массив /Array/,
- Стек /Stack/,
- Дараалал /Queue/ гэж ангилна.

Хийсвэр өгөгдлийн төрлүүд

Энэ бүлгийг судалсны дараа оюутан дараахь зүйлийг хийх чадвартай байх ёстой.

- ❑ Хийсвэр өгөгдлийн төрөл (ADT) гэсэн ойлголтыг тодорхойлох.
- ❑ Стек, стек дээрх үндсэн үйлдлүүд, тэдгээрийн хэрэглээ, тэдгээрийг хэрхэн яаж хийхийг тодорхойлох хэрэгжүүлсэн.
- ❑ Дараалал, дарааллын үндсэн үйлдлүүд, тэдгээрийн хэрэглээ, тэдгээрийг хэрхэн хийхийг тодорхойлох хэрэгжих.
- ❑ Ерөнхий шугаман жагсаалт, жагсаалт дээрх үндсэн үйлдлүүд, тэдгээрийн хэрэглээ, хэрхэн хийхийг тодорхойлох тэдгээрийг хэрэгжүүлэх боломжтой.
- ❑ Ерөнхий мод ба түүний хэрэглээг тодорхойлох.
- ❑ Хоёртын мод буюу тусгай төрлийн мод, түүний хэрэглээг тодорхойлно.
- ❑ Хоёртын хайлтын мод (BST) болон түүний програмуудыг тодорхойлно.
- ❑ График ба түүний хэрэглээг тодорхойлох.

Үндсэн ойлголт

Компьютерийн тусламжтайгаар асуудлыг шийдвэрлэх нь өгөгдөл боловсруулах гэсэн үг юм.

Өгөгдлийг боловсруулахын тулд бидэнд хэрэгтэй өгөгдлийн төрөл болон өгөгдөл дээр хийх үйлдлүүд багтана. Жишээлбэл, тоонуудын жагсаалтын нийлбэрийг олоход бид тухайн тооны төрлийг сонгох ёстой (бүхэл тоо эсвэл бодит гэх мэт) мөн үйлдлийг (нэмэх гэх мэт) тодорхойлно. Өгөгдлийн төрлийн тодорхойлолт ба өгөгдөлд хэрэглэгдэх үйл ажиллагаа (operation) нь хийсвэр өгөгдлийн төрлийн цаадах санааны нэг хэсэг юм. Англиар (ADT) гэж нэрлэнэ.

Хамгийн энгийнээр хэлбэл өгөгдөл дээр хэрхэн үйлдэл хийж байгааг нуух асуудал. Өөрөөр хэлбэл, хэрэглэгч нь ADT нь зөвхөн өгөгдлийн төрөлд зориулсан үйлдлүүдийн багц гэдгийг мэдэх хэрэгтэй, гэхдээ тэдгээрийг хэрхэн хэрэглэхийг мэдэх шаардлагагүй.

Үндсэн ойлголт

ADT сэдвээр ярихад хамгийн чухал суурь асуудлаас эхэлж ярья. Өгөгдлийн бүтэц гэдэг нь өгөгдлийг үр ашигтай ашиглах боломжтой байдлаар хадгалах буюу зохион байгуулахыг хэлнэ. Энгийн өгөгдлийн бүтцүүдийг нэрлэвэл

- Жагсаалт list
- Массив буюу Хүснэгт (Array)
- Стек, Stack
- Дараалал, Queue
- Граф, Graph
- Мод, Tree
- Овоолго, Heap
- Хэш хүснэгт, Hash Table

Жишээ нь tree нь өгөгдлийн сан зохион байгуулахад илүү тохиромжтой байдаг бол хэш хүснэгт нь хөрвүүлэгчдэд тодорхойлогч хайх зорилгоор ашиглагддаг. Өгөгдлийн бүтцүүд нь их хэмжээний өгөгдлүүдийг үр ашигтайгаар зохион байгуулах зорилготой

Энгийн ADTs

Программист үйлдлүүдийг хэрхэн хийж байгааг мэдэх шаардлагагүй . Жишээлбэл, программист z ← x + y илэрхийлэлийг ашиглаж байна гэж бодьё тэгвэл энэ нь y (бүхэл тоо)-ийн утгад нэмэх x (бүхэл тоо) ба үр дүн z (бүхэл тоо) гэж үзнэ. Гэхдээ программист нэмэлт нь яаж байгааг мэдэх шаардлагагүй байхаар гүйцэтгэсэн.

Хэрхэн хийгдэж байгааг тайлбарлая

Энэ нэмэлтийг компьютерээр хийдэг арга нь санах ойн хоёр байршилд хоёр бүхэл тоог хоёр нөхөх форматаар хадгалах юм.

Тэдгээрийг CPU регистрт ачаалж, хоёртын системд нэмж, үр дүнг буцааж хадгална гэхдээ өөр санах ойн байршилд. Гэхдээ програмист хүн үүнийг мэдэх шаардлагагүй.

Си хэл дээрх дээрх үйлдлийн бүхэл тоо нь урьдчилан тодорхойлсон үйлдлүүдтэй хийгдэх энгийн хийсвэр өгөгдлийн төрөл юм. Үйл ажиллагаа хэрхэн явагддаг хийгдэх нь программист санаа зовох зүйл биш гэсэн утга эндээс гарч ирнэ ...

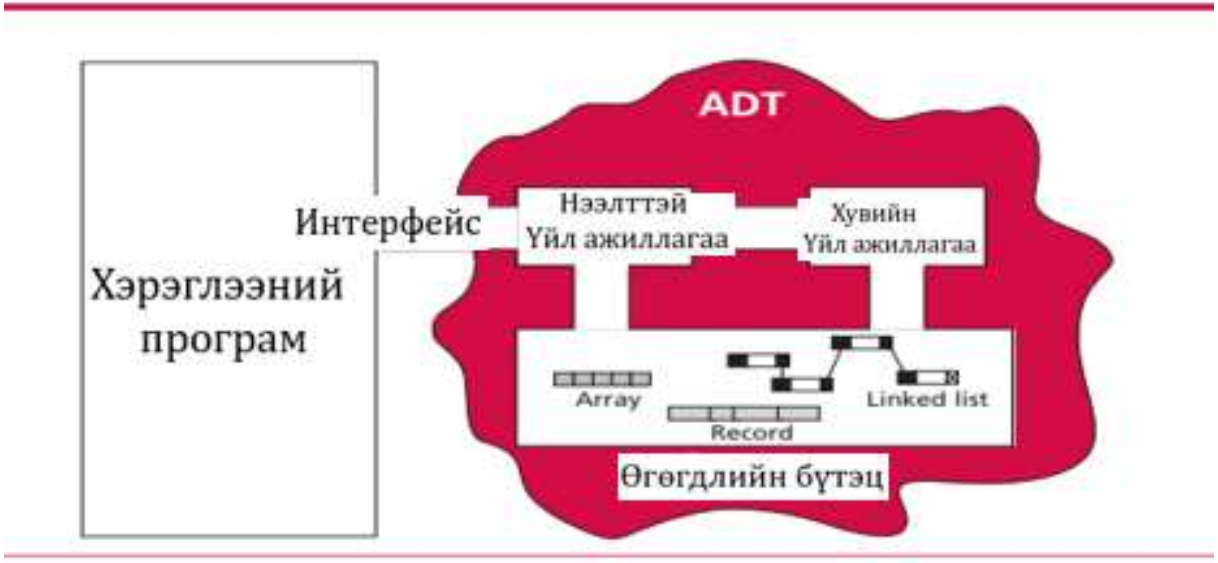
Комплекс ADTs

Хэд хэдэн энгийн ADT-ууд байна.

- бүхэл тоо,
- бодит,
- тэмдэгт,
- заагч гэх мэт

Эдгээрийг ихэнх хэл дээр ашиглах боломжтой,
ADT-ийн тусламжтайгаар хэрэглэгчид даалгаврыг хэрхэн гүйцэтгэх талаар бус харин юу хийж чадах талаар санаа тавьдаг. ADT нь зөвшөөрөгдсөн тодорхойлолтуудын багцаас бүрдэх бөгөөд программистууд тэдгээрийн хэрэгжилтийг нуун дарагдуулсан ил биш үйлдлийг ашигладаг.
Тодорхойгүй хэрэгжүүлэлт бүхий үйлдлүүдийн ерөнхий байдлыг хийсвэрлэл гэж нэрлэдэг. Хийсвэр бүтцээр үйл явцын мөн чанарыг нууцалдаг арга гэсэн үг

Хийсвэр өгөгдлийн төрлийн загвар



Зураг 1. ADT загвар

ADT загварыг Зураг 1-т үзүүлэв. Өнгөт хэсэг нь ADT-ийг илэрхийлнэ. ADT дотор загварын хоёр өөр хэсэг байдаг: өгөгдлийн бүтэц ба үйл ажиллагаа (нээлттэй болон хувийн). Массив, холбосон жагсаалт, рекорд зэрэг өгөгдлийн бүтэц нь ADT дотор байдаг бөгөөд тэдгээрийг нээлттэй болон хувийн үйл ажиллагаанд ашигладаг.

Хэрэглээний програм нь зөвхөн интерфэйсээр дамжуулан нээлттэй үйл ажиллагаанд хандах боломжтой. Интерфэйс нь нээлттэй үйл ажиллагаа, өгөгдлийн жагсаалт хоёртой холбогдож хоёр урсгалаар (шилжсэн эсвэл буцаж ирсэн) тодорхойлогдоно. Хувийн үйл ажиллагаа нь дотоод хэрэгцээнд зориулагдсан.

Хэрэгжилт

Компьютерийн хэл нь ADT багцуудыг ашиглахын тулд эхлээд хэрэгжүүлдэг мөн санд хадгална. Өөр өөр төрлийн өгөгдлийн бүтцүүд нь өөр өөрийн гэсэн тусгай үүрэгтэй болон давуу талтай байдаг. Жишээ нь tree нь өгөгдлийн сан зохион байгуулахад илүү тохиромжтой байдаг бол хэш хүснэгт нь хөрвүүлэгчдэд тодорхойлогч хайх зорилгоор ашиглагддаг. Аливаа нэгэн үр ашигтай алгоритмийг зохион байгуулахад зөв өгөглийн санг ашиглах нь чухал. Зарим нэгэн программчиллийг загварчилахдаа алгоритм гэхээсээ илүүтэйгээр өгөгдлийн санг түлхүү ашигладаг.

Стек нь хязгаарлагдмал шугаман жагсаалт бөгөөд стект хамгийн сүүлд орсон элемент хамгийн эхэнд гардаг учраас компьютерийн ухаанд стекийг LIFO (Last In First Out) бүтэц гэж бас нэрлэдэг.

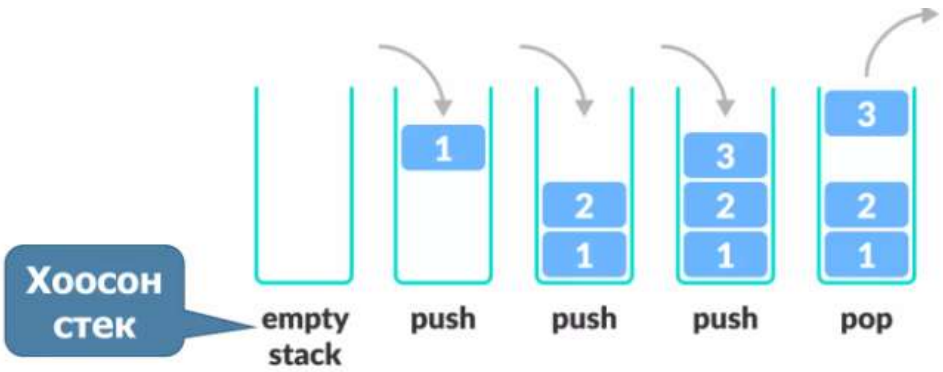
Stack

Зураг 2 Стекийн жишээнүүд

Өөрөөр хэлбэл, Стек буюу Stack нь онцгой үүрэгтэй өгөгдлийн бүтцийн төрөл. Хэрэв бид хэд хэдэн өгөгдлийг стек рүү оруулаад дараа нь устгавал өгөгдлийн дараалал урвуу болно. Жишээлбэл, 5, 10, 15, 20 гэсэн өгөгдлийн оролтыг 20, 15, 10, 5 гэж хасна. Энэ урвуу шинж чанар нь стекийг хамгийн сүүлд орж ирсэн, эхлээд гаргадаг (LIFO) өгөгдлийн бүтэц гэж нэрлэдэг болсонтой холбоотой. Өдөр тутмын амьдралдаа олон төрлийн стек ашигладаг. Жишээ болгон овоолсон зоос эсвэл овоолгын номны тухай авч үзье. Зөвхөн дээд талд байгаа объектыг нэмэх эсвэл хасах боломжтой аливаа нөхцөл байдал нь стек юм. Хэрэв бид дээд талд байгаа объектоос өөр объектыг устгахыг хүсвэл эхлээд түүний дээрх бүх объектыг устгах ёстой. Зураг 2-т стекийн гурван дүрслэлийг үзүүлэв.

Стаск

Стекээс элемент устгах нь нөгөө талаараа шинэ элемэнтэд орон зай бий болгож байгаа хэрэг юм. Ингэснээр хамгийн доор байгаа элемент нь хамгийн удаан оршин байх юм. /Зураг 3. Стекийн жишээнүүд/ Үндсэн үйлдлүүдээс гадна Peek буюу шагайх аргыг хэрэгжүүлсэн байдаг. Энэхүү арга нь хамгийн дээд талын элементийг устгалгүйгээр утгыг нь буцаадаг арга юм.



Зураг 3. Стекийн жишээнүүд

Стаск дээрх үйлдлүүд

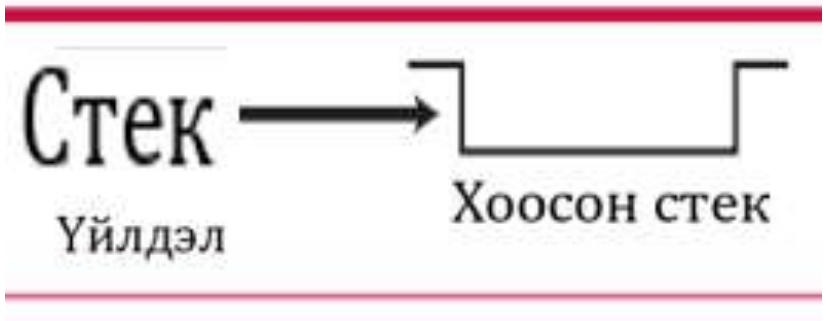
Хэдийгээр бид стекийн хувьд олон үйлдлүүдийг тодорхойлж болох ч үндсэн дөрвөн үйлдлүүд байдаг:stack, push, pop, empty

Push болон Pop аргын үндсэн шинж нь стекийн элемэнтүүдийн зүй тогтолтой байдгийг илэрхийлдэг.

Стек үйлдэл нь хоосон стек үүсгэдэг. Дараах зураг 4-т форматыг харуулж байна.

stack (stackName)

stackName нь үүсгэх стекийн нэр юм. Энэ үйлдэл нь хоосон стекийг буцаана. Зураг-т энэ үйлдлийг дүрслэн харуулсан дүрслэлийг үзүүлэв.



- **Push:** Стекийн дээд хэсэгт элемент нэмэх,
- **Pop:** Стекийн дээд хэсгээс элементийг авах.
- **IsEmpty:** Стек хоосон эсэхийг шалгах,
- **IsFull:** Стек дүүрсэн эсэхийг шалгах,
- **Peek:** Хасалгүйгээр дээд элементийн утгыг авах

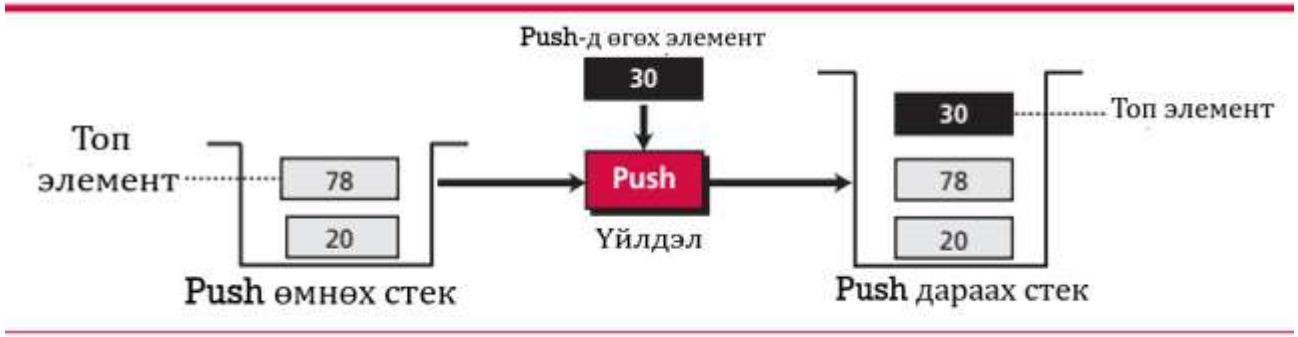
Зураг 4. Stack үйлдэл

Push үйлдэл

Түлхэх үйлдэл нь стекийн дээд талд байгаа зүйлийг оруулдаг. Стекээс элемент түлхэх нь нөгөө талаараа шинэ элемэнтэд орон зай бий болгож байгаа хэрэг юм. Ингэснээр хамгийн доор байгаа элемент нь хамгийн удаан оршин байх юм.Дараах нь форматыг харуулж байна.

```
push (stackName, dataItem)
```

stackName нь стекийн нэр бөгөөд dataItem нь дээд талд оруулах өгөгдөл юм. Түлхэх үйлдлийн дараа шинэ зүйл нь стек дэх хамгийн дээд элемент болно. Энэ үйлдэл нь дээд талд оруулсан dataItem бүхий шинэ стекийг буцаана. Зураг 5-д харуулав



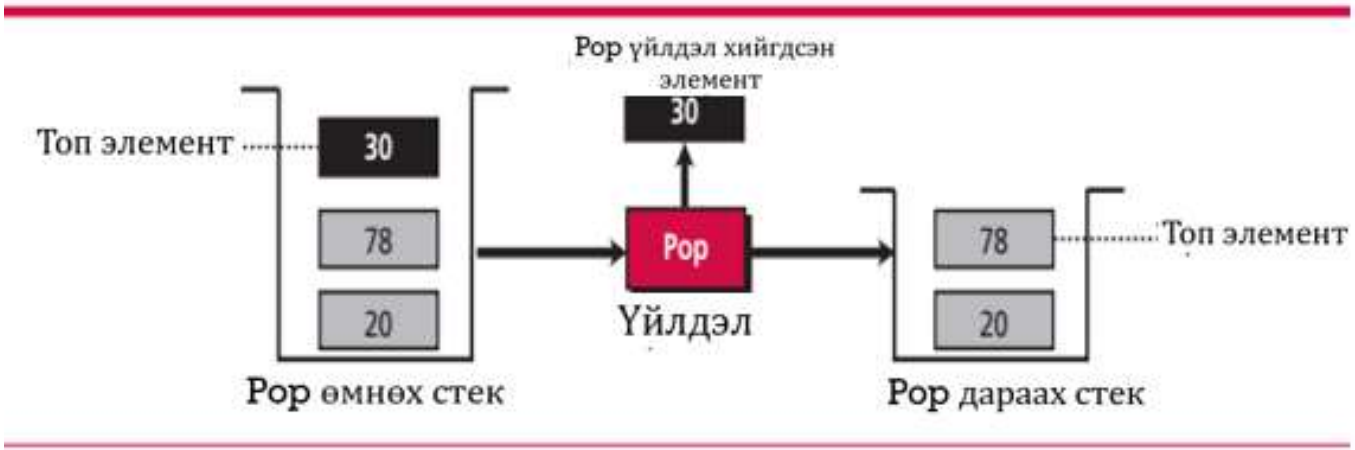
Зураг 5. Push operation

Pop Үйлдэл

Поп үйлдэл нь стекийн дээд талд байгаа зүйлийг устгадаг. Дараах Зураг 6-д форматыг харуулж байна.

pop (stackName, dataItem)

stackName нь стекийн нэр бөгөөд dataItem нь стекээс устгагдсан өгөгдөл юм. Зураг -т энэ үйлдлийг дүрслэн харуулав.



Зураг 6. Pop операци

Поп үйлдлийн дараа устгахаас өмнө дээд элементийн доорх элемеэнт нь дээд элемент болдог. Энэ үйлдэл нь нэг элемент багатай шинэ стекийг буцаана.

Empty үйлдэл

Хоосон үйлдэл нь стекийн төлөвийг шалгадаг. Дараах нь форматыг харуулж байна.

```
empty (stackName)
```

```
bool isempty() {
if(top == -1)
return true;
else return false;
}
```

stackName нь стекийн нэр юм. Хэрэв стек хоосон байвал энэ үйлдэл үнэнийг буцаана стек хоосон биш бол худал.

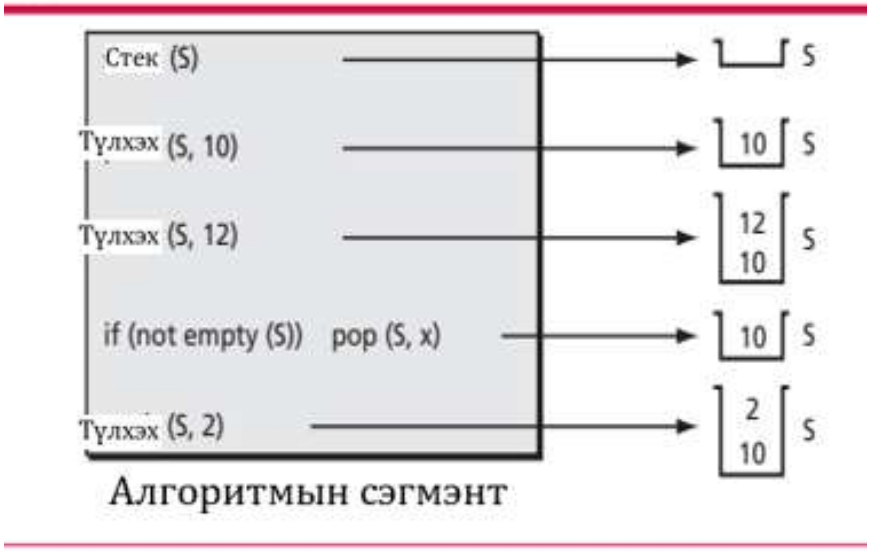
С програмчлалын хэлээр **isempty()** функцийг ашиглахад массив дахь индекс 0-ээс эхэлдэг тул бид **top**-ийг -1 гэж эхлүүлнэ. Тиймээс дээд хэсэг нь тэгээс доош байгаа эсэхийг шалгаж, стек хоосон байгаа эсэхийг тодорхойлдог.

Жишээ

Зураг 9-д стек дээр S ээр өмнө нь тодорхойлсон үйлдлүүдийг ашигладаг алгоритмын сегментийг үзүүлсэн.

Дөрөв дэх үйлдэл нь стекийг нээхээс өмнө стекийн төлөвийг шалгадаг дээд элемент.

Дээд талын элементийн утгыг x хувьсагчид хадгална. Энэ утгыг ашиглаад алгоритмын сегментийн төгсгөлд автоматаар устгагдана. Дараах Stack классын зарлалтаар MaxSize хэмжээтэй stack массив, стекийн оройг тодорхойлох top хувьсагч болон стекд элемент хийх push() функцийг тодорхойллоо.



Зураг 7. Жишээ

Стек хэрэглээ

Стек програмуудыг дөрвөн том ангилалд ангилна:

- өгөгдлийг буцаах,
- өгөгдөл холбох,
- өгөгдлийн ашиглалтыг хойшлуулах,
- алхмуудыг буцаах.

Өгөгдлийн зүйлийг буцаах нь тухайн өгөгдлийн багцыг эхнийх нь болгохын тулд дахин эрэмбэлэхийг шаарддаг болон сүүлчийн зүйлсийг эхний болон сүүлчийн хоёрын хоорондох бүх байрлалыг сольж байна бас харьцангуй солилцсон. Жишээлбэл, жагсаалт (2, 4, 7, 1, 6, 8) (8, 6, 1, 7, 4, 2) болно.

Стек гэдэг нь компьютерийн шинжлэх ухааны нэгэн энгийн өгөгдлийн бүтэц бөгөөд үүнийг хийсвэр байдлаар илэрхийлэх боломжтой, эсвэл эсвэл шугаман жигсаалтын хэлбэрээр орой хэсгээсээ нэмж хасдаг байдалтайгаар тодорхойлж болно.

Стек хэрэглээ

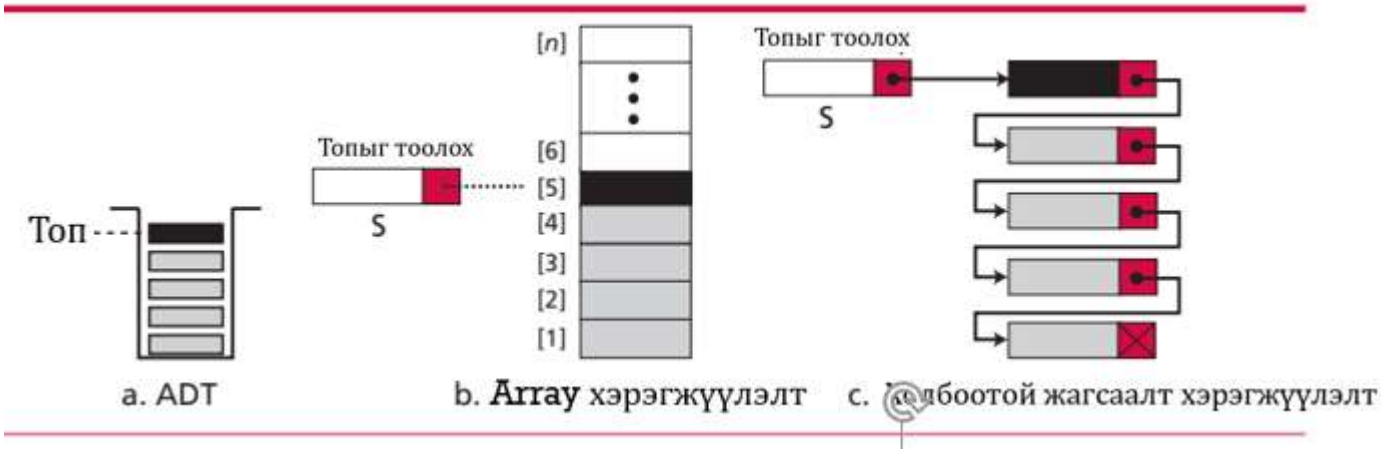
Хэдийгээр стек нь маш хязгаарлагдмал үйлдэлтэй боловч компьютерийн програмчлалд чухал үүрэгтэй өгөгдлийн бүтцүүдийн нэг юм.
Жишээ нь нэг ажлын явцад өөр зүйл хийхээр түр хойшлуулах, эсвэл програм бүхэлдээ өөр өөр зарчмаар ажиллах зэрэг олон алгоритмд стек үндсэн үүрэг гүйцэтгэнэ. Жишээ нь CALL, RETURN зэрэг функцэд стекийг ашигладаг байна.
Бүхэл тоог аравтын бутархайгаас дурын суурь руу хөрвүүлэх энгийн UML диаграммыг өгсөн.
Хэдийгээр алгоритм нь маш энгийн боловч хөрвүүлсэн бүхэл тоонуудын цифрүүдийг үүсгэгдсэн байдлаар нь хэвлэвэл бид урвуу дарааллаар цифрүүдийг авна.
Компьютерийн аль ч хэл дээрх хэвлэх заавар нь тэмдэгтүүдийг зүүнээс баруун тийш хэвлэдэг боловч алгоритм нь баруунаас зүүн тийш цифрүүдийг үүсгэдэг.
Бид асуудлыг шийдэхийн тулд стекийн урвуу шинж чанарыг (LIFO бүтэц) ашиглаж болно.
Дараах алгоритмд аравтын бүхэл тоог хоёртын тоо руу хөрвүүлэх, үр дүнг хэвлэх псевдокодыг харуулж байна.

Алгоритм

```
Algorithm: Аравтаас хоёрт руу
Purpose:   Өгөгдсөн бүхэл тооны хоёртын утгыг хэвлэх
Pre:  Өгөгдсөн бүхэл тоо
Post: Хоёртын тоо хэвлэгдэх
Return: None
{
    stack (S)
    while ( тоо  ≠ 0)
    {
        үлдэгдэл ← number mod 2
        push (S, үлдэгдэл)
        тоо  ← тоо  / 2
    }
    while (Хоосон биш (S))
    {
        pop (S, x)
        print (x)
    }
    return
}
```

Бид эхлээд хоосон стек үүсгэнэ. Дараа нь бит үүсгэхийн тулд while давталтыг ашигладаг, гэхдээ хэвлэхийн тулд тэдгээрийг стек руу push хийнэ.
Бүх битүүд үүссэний дараа хэсгээс гарна. Дараа нь стекээс битүүдийг гаргаж аваад хэвлэхийн тулд өөр loop ашиглана. Битүүд нь байсан дарааллаар нь урвуу дарааллаар хэвлэгддэг.

Жишээ

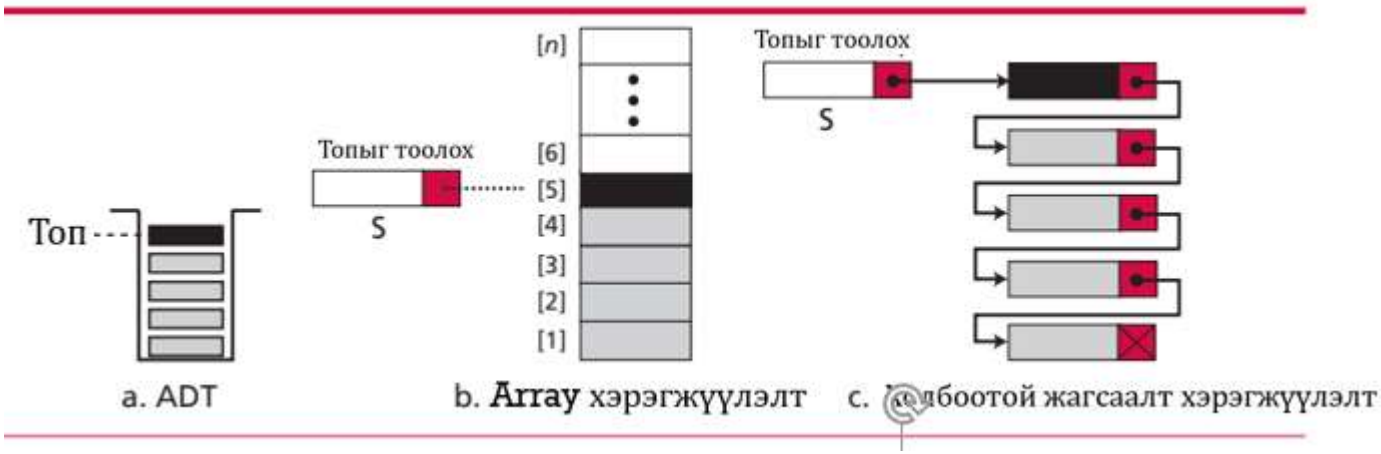


Зураг 8. Stack хэрэгжүүлэлт

Холбоос (**LinkedList**) : Холбоос гэдэг нь нэг элементийн дараагийн элемент санах ойн хаана байгааг заасан хаяг юм. Элемент, холбоос гэсэн хосууд гинж байдлаар залгагдаж өгөгдлийг бүрдүүлдэг. Дотроо **Singly, Doubly, Circular** гэсэн төрлүүдтэй /Зураг 8. **Stack** хэрэгжүүлэлт/

Array болон **LinkedList** алийг нь ч ашиглаж **Stack** үүсгэж боломжтой.

Жишээ



Зураг 9. Stack array, linked lists хэрэгжүүлэлт

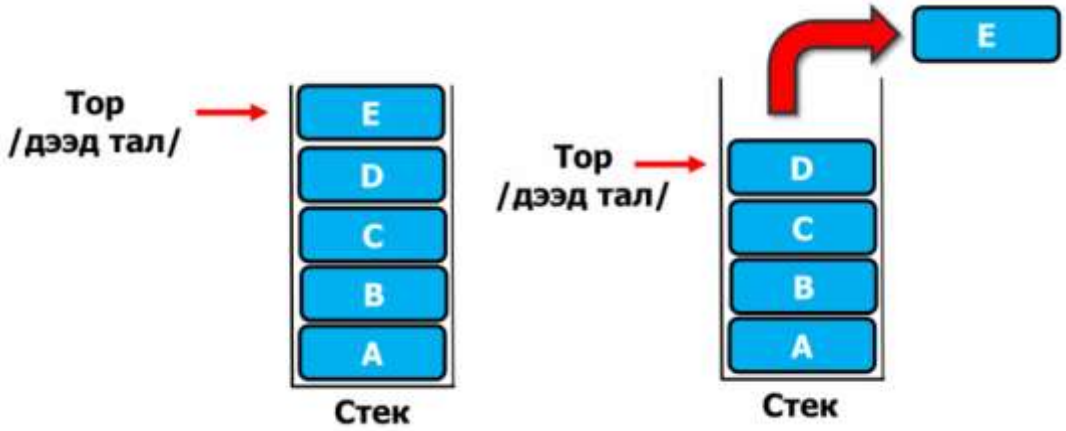
Холбоостой жагсаалтын хэрэгжилт нь ижил төстэй: бидэнд стекийн нэртэй нэмэлт зангилаа бий. Энэ зангилаа нь тоологч ба дээд элементийг заадаг заагч гэсэн хоёр талбартай.

Зураг 9-д таван зүйл бүхий ADT стекийн жишээг үзүүлэв. Манай массивын хэрэгжилтэд хоёр талбартай бичлэгтэй байна. Эхний талбарыг массивын талаарх мэдээллийг хадгалахад ашиглаж болно: бид үүнийг тоолох талбар болгон ашигласан. Энэ нь агшин бүрт стек дэх өгөгдлийн зүйлийн тоог харуулдаг.

Хоёр дахь талбар нь бүхэл тоо. Энэ нь дээд элементийн индексийг агуулна.

Алгоритмуудын жишээ

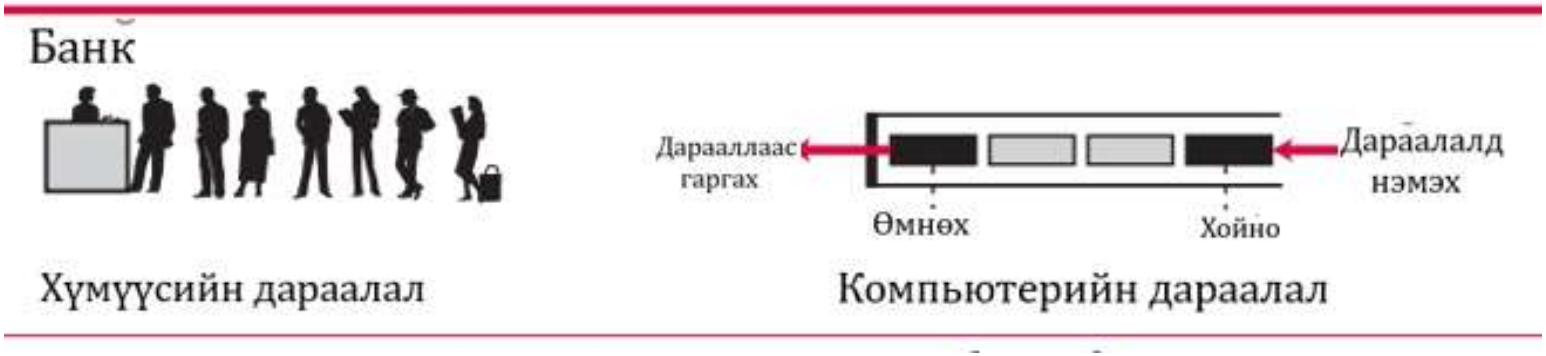
Бид хэрэгжүүлэлт бүрт стекээр тодорхойлсон дөрвөн үйлдлийнхээ дөрвөн алгоритмыг псевдокодоор тодорхойлсон. Эдгээр алгоритмуудыг өөрчилж, стек, түлхэх, поп, хоосон гэсэн дөрвөн
алгоритмыг үүсгэж болно. Жишээ нь Зураг 10. Stack pop үйлдийг дүрслэв.



Зураг 10. Stack pop үйлдэл

Стекийн өгөгдлийг устгахыг **Pop үйлдэл** гүйцэтгэнэ. **Pop()** үйлдлийн массивд ашиглахад өгөгдлийн элементийг устгаагүй боловч дээд **top** хэсгийг нэг байрлалд бууруулж дараагийн утгыг зааж өгнө. Гэхдээ холбоост жагсаалтад ашиглахад **pop()** нь өгөгдлийн элементийг устгаж, санах ойн зайг хуваарилдаг.

Дараалал



Зураг 11. Дарааллыг ойлгох үндэс

Дараалал гэдэг нь зөвхөн нэг төгсгөлд өгөгдөл оруулах боломжтой шугаман жагсаалт бөгөөд арын хэсэг гэж нэрлэгдэж, нөгөө төгсгөлөөс нь урд тал гэж нэрлэгддэг. Эдгээр хязгаарлалтууд нь өгөгдлийг хүлээн авсан дарааллаар нь дарааллаар нь боловсруулдаг. Жишээ нь Зураг 11. Дарааллын үйлдийг дүрслэв. Өөрөөр хэлбэл, дарааллын бүтэц. нь эхлээд орж, эхлээд гарна (FIFO)
Дээрх зураг 11-т нь дарааллын хоёр дүрслэлийг харуулж байна, нэг нь хүмүүсийн дараалал, нөгөө нь дараалал.

Дараалал

Дараалал нь өдөр тутмын амьдралаас танил болсон. Автобусны буудал дээр автобус хүлээсэн хүмүүсийн дараалал нь дараалал, утасны операторын хариу өгөхөөр хүлээгдэж буй дуудлагын жагсаалт, компьютерт хариулах хүлээгдэж буй ажлын байрны жагсаалт нь дараалал юмв Зураг 12. Дараалалын жишээ . Хүмүүс болон өгөгдөл хоёулаа арын эгнээнд орж, урд талд ирэх хүртэл урагшилдаг. Тэд дарааллын урд ирсний дараа компьютерийг дараалалд үлдээж, үйлчилдэг.

Зураг 12. Дараалалын жишээ

Дараалал

Дараалал гэдэг нь дараах үйлдлүүдийг хийх боломжийг олгодог хийсвэр өгөгдлийн бүтэц (Abstract data structure) юм.

- **Enqueue:** Дарааллын төгсгөлд элемент нэмэх,
- **Dequeue:** Дарааллын урд талын элементийг арилгах,
- **IsEmpty:** Дараалал хоосон байгаа эсэхийг шалгах,
- **IsFull:** Дараалал дүүрэн эсэхийг шалгах,
- **Peek:** Дарааллын урд талын элементийг устгахгүйгээр утгыг авах

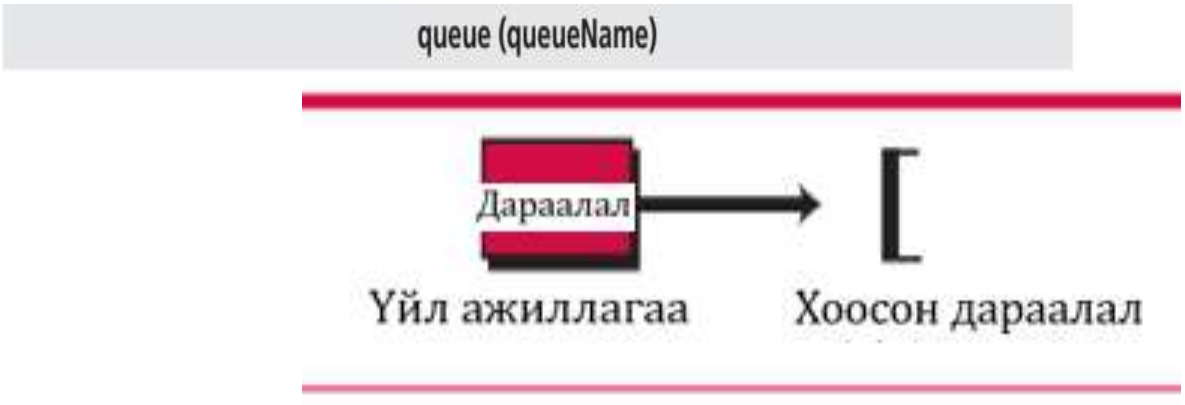
Дараалал дээрх үйлдлүүд

Хэдийгээр бид дарааллын олон үйлдлийг тодорхойлж болох ч үндсэн дөрвөн үйлдлүүд нь:

- Дараалал - queue,
- Дараалалд оруулах - enqueue,
- Дааллаас гаргах - dequeue,
- Хоосон - empty,

Дарааллын үйлдэл нь хоосон дараалал үүсгэдэг. Дараах зураг 13-т т үндсэн форматыг үзүүлэв.

queueName нь үүсгэх дарааллын нэр юм. Энэ үйлдэл нь хоосон дарааллыг буцаана.



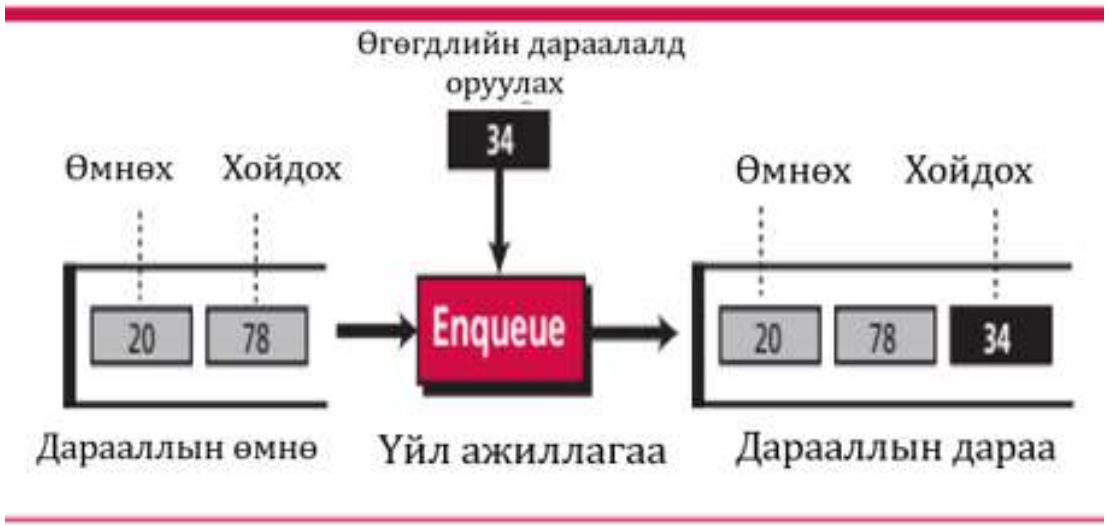
Зураг 13. Дараалал дахь үйлдэл

Enqueue Дараалал

Дараалалд оруулах үйлдэл нь дарааллын ард байгаа зүйлийг оруулдаг.

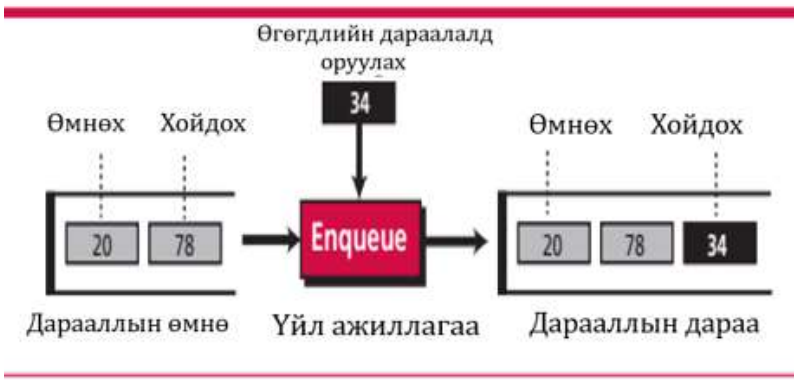
```
enqueue (queueName, dataItem)
```

queueName нь дарааллын нэр бөгөөд dataItem нь дарааллын арын хэсэгт оруулах өгөгдөл юм. Дараалалд орсны дараа шинэ зүйл нь дарааллын сүүлчийн зүйл болно.
Энэ үйлдэл нь ар талд оруулсан dataItem бүхий шинэ дарааллыг буцаана.
Зураг 14-т үйлдлийн дүрслэлийг үзүүлсэн.



Зураг 14. The enqueue процесс

Enqueue Дараалал



Зураг 15а. The enqueue процесс

Алхам 1 - Дараалал дүүрэн эсэхийг шалгах,

Алхам 2 - Хэрэв дараалал дүүрсэн бол халих алдаа гарна,

Алхам 3 - Хэрэв дараалал дүүрээгүй бол дараагийн хоосон зайг зааж өгөхийн тулд арын заагчийг нэмэгдүүлнэ,

Алхам 4 - Мэдээллийн элементийг ар тал руу чиглэсэн дарааллын байршилд нэмнэ,

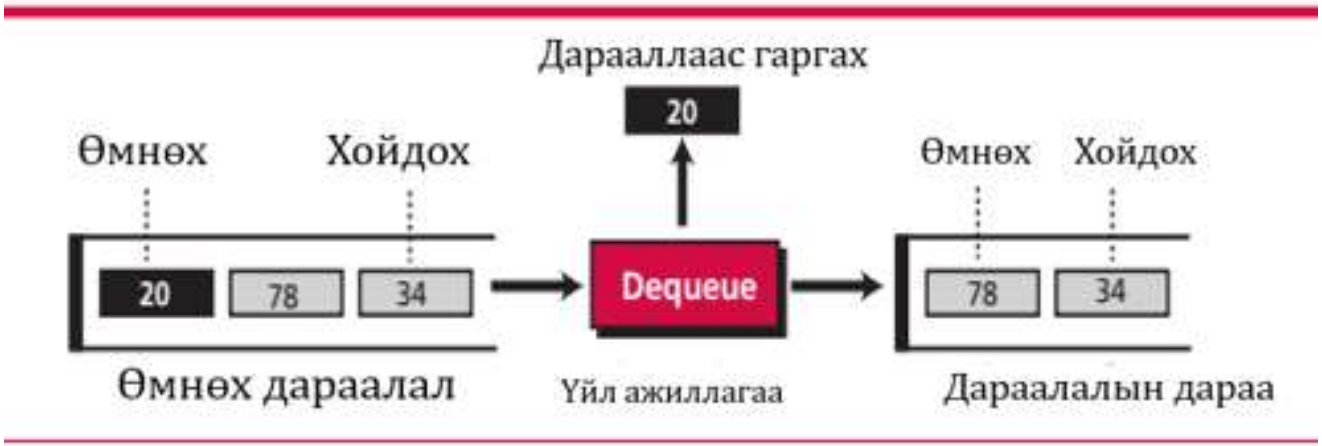
Алхам 5 - Амжилттай буцна.

Enqueue Дараалал

Дарааллаас хасах үйлдэл нь дарааллын урд байгаа зүйлийг устгадаг.
Зураг 15ab dequeue процессийг харуулсан.

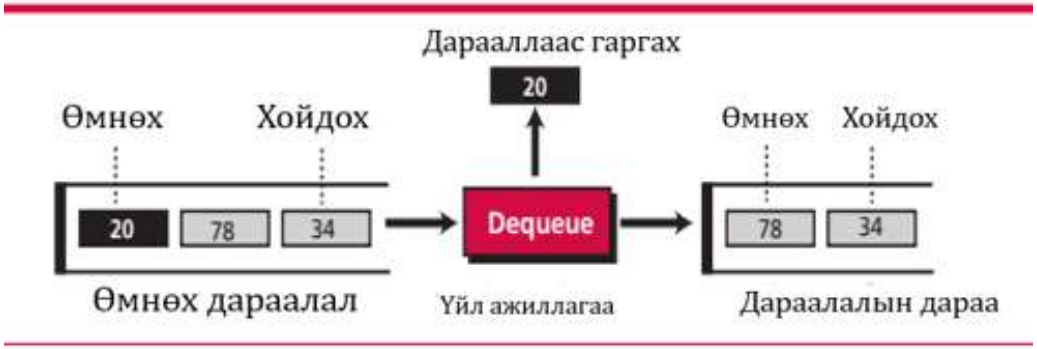
```
dequeue (queueName, dataItem)
```

queueName нь дарааллын нэр бөгөөд dataItem нь -аас устгагдсан өгөгдөл юм
Тодорхой хугацааны дараа урд талын элементийг дагасан зүйл урд хэсэг болно
Энэ үйлдэл нь нэг элемент багатай шинэ дарааллыг буцаана.



Зураг 15b dequeue процесс

Enqueue Дараалал



Зураг 16. The dequeue процесс

Алхам 1 - Дараалал хоосон байгаа эсэхийг шалгах,

Алхам 2 - Хэрэв дараалал хоосон байвал доорхи алдааг гарна,

Алхам 3 - Хэрэв дараалал хоосон биш бол урагш чиглэсэн өгөгдөлд хандах,

Алхам 4 - Дараагийн боломжтой өгөгдлийн элемент рүү чиглүүлэхийн тулд урд талын заагчийг нэмэгдүүлэх,

Алхам 5 - Амжилттай буцна. Зураг 16. The dequeue процесс

Empty Үйлдэл

Хоосон үйлдэл нь дарааллын төлөвийг шалгадаг.

```
empty (queueName)
```

queueName нь дарааллын нэр юм. Хэрэв дараалал хоосон байвал энэ үйлдэл үнэнийг буцаана дараалал хоосон биш бол худал.

Queue ADT

- Дараалал - queue,
- Дараалалд оруулах - enqueue,
- Дааллаас гаргах - dequeue,
- Хоосон - empty,

Жишээ

Зураг 17 -т өмнө нь тодорхойлсон үйлдлүүдийг Q дарааламд ашигладаг алгоритмын зарчмыг үзүүлэв. Дөрөв дэх үйлдэл нь урд талын элементийг дарааллын дарааллаас гаргах оролдлогын өмнө дарааллын төлөвийг шалгадаг. Урд талын элементийн утгыг х хувьсагчд хадгалах ч энэ утгыг ашигладаггүй – Ийм учраас энэ нь алгоритмын сегментийн төгсгөлд автоматаар хаягдах зарчимтай.

Зураг 17. Жишээ

Дараалалын хэрэглээ

Дараалал бол **FIFO** зрчимтай. Дараалал нь өгөгдөл боловсруулах бүх бүтцийн хамгийн түгээмэл бүтцийн нэг юм. Үйлдлийн систем, сүлжээ гэх мэт мөн үйлчлүүлэгчийн хүсэлт, ажлын байр, захиалга боловсруулах зэрэг онлайн бизнесийн програмуудад дарааллыг ашигладаг. Компьютерийн системд ажил боловсруулах, хэвлэх гэх мэт системийн үйлчилгээнд дараалал шаардлагатай.

Өгөгдлийн зарим шинж чанараар нь мэдээллийн санг зохион байгуулахад дарааллыг ашиглаж болно. Жишээлбэл, компьютерт 1000-аас бага, 1000-аас дээш гэсэн хоёр ангилалд хадгалагдан эрэмбэлэгдсэн өгөгдлийн жагсаалт байна гэж төсөөлье. Бид хоёр дарааллыг ашиглан категориудыг салгаж, өгөгдлийн дарааллыг хадгалах боломжтой. Дараах алгоритмд энэ үйлдлийн псевдокодыг харуулж байна.

Алгоритм

```
Algorithm: Төрөлжүүлэгч
Purpose: Өгөгдлийг хоёр төрөлд хуваагаад, хоёр тусдаа лист үүсгэх
Pre: Өгөгдсөн , эх лист
Post: Хоёр листийг хэвлэх
Return: Үгүй
{
        queue (Q1)
        queue (Q2)
        while Листэнд их өгөгдөл байх
        {
            if (data < 1000)
            {
                enqueue (Q1, data)
            }
            if (data ≥ 1000)
            {
                enqueue (Q2, data)
            }
        }
        while (Хоосон биш (Q1))
        {
            dequeue (Q1, x)
            Хэвлэх (x)
        }
        while (Хоосон биш Q2))
        {
            dequeue (Q2, x)
            Хэвлэх (x)
        }
        return
}
```

Өгөгдлийн зарим шинж чанараар нь мэдээллийн санг зохион байгуулахад дарааллыг ашиглаж болно. Жишээлбэл, компьютерт 1000-аас бага, 1000-аас дээш гэсэн хоёр ангилалд хадгалагдан эрэмбэлэгдсэн өгөгдлийн жагсаалт байна гэж төсөөлье. Энэ хоёр дарааллыг ашиглан категориудыг салгаж, өгөгдлийн дарааллыг хадгалах

Жишээ

Дарааллын өөр нэг түгээмэл хэрэглүүр бол хурдын хооронд тэнцвэрийг бий болгох, тохируулах явдал юм. Жишээлбэл, CPU нь принтерт холбогдсон байна гэж бодъё.

Принтерийн хурдыг CPU-ийн хурдтай харьцуулах аргагүй юм. Хэрэв CPU нь CPU-ийн үүсгэсэн зарим өгөгдлийг хэвлэгчийг хэвлэхийг хүлээх юм бол CPU удаан хугацаанд ажиллахгүй байх болно.

Үүний шийдэл нь дараалал. CPU нь дараалалд багтаах боломжтой олон тооны өгөгдлийг үүсгэж дараалал руу илгээдэг. Хэсэг хэсгүүдийг аажим аажмаар задалж, хэвлэгчээр хэвлэнэ. Энэ зорилгоор ашигладаг дарааллыг ихэвчлэн дамар дараалал гэж нэрлэдэг.

Дарааллын хэрэгжүүлэлт

ADT түвшинд бид дараалал болон дарааллын дөрвөн үйлдлийг (дараалал, дараалал, дараалал, хоосон) ашигладаг: хэрэгжилтийн түвшинд өгөгдлийн бүтцийг ашиглана.
ADT-ийн дарааллыг массив эсвэл холбосон жагсаалтыг ашиглан хийж болно. Массивын хэрэгжилтэд бид гурван талбар бүхий рекордийг үүсгэнэ.

- Эхний талбар: дарааллын талаарх мэдээллийг хадгалахад ашиглана: Бид үүнийг дараалалд байгаа өгөгдлийн зүйлийн одоогийн тоог харуулсан тоолох талбар болгон ашигладаг.

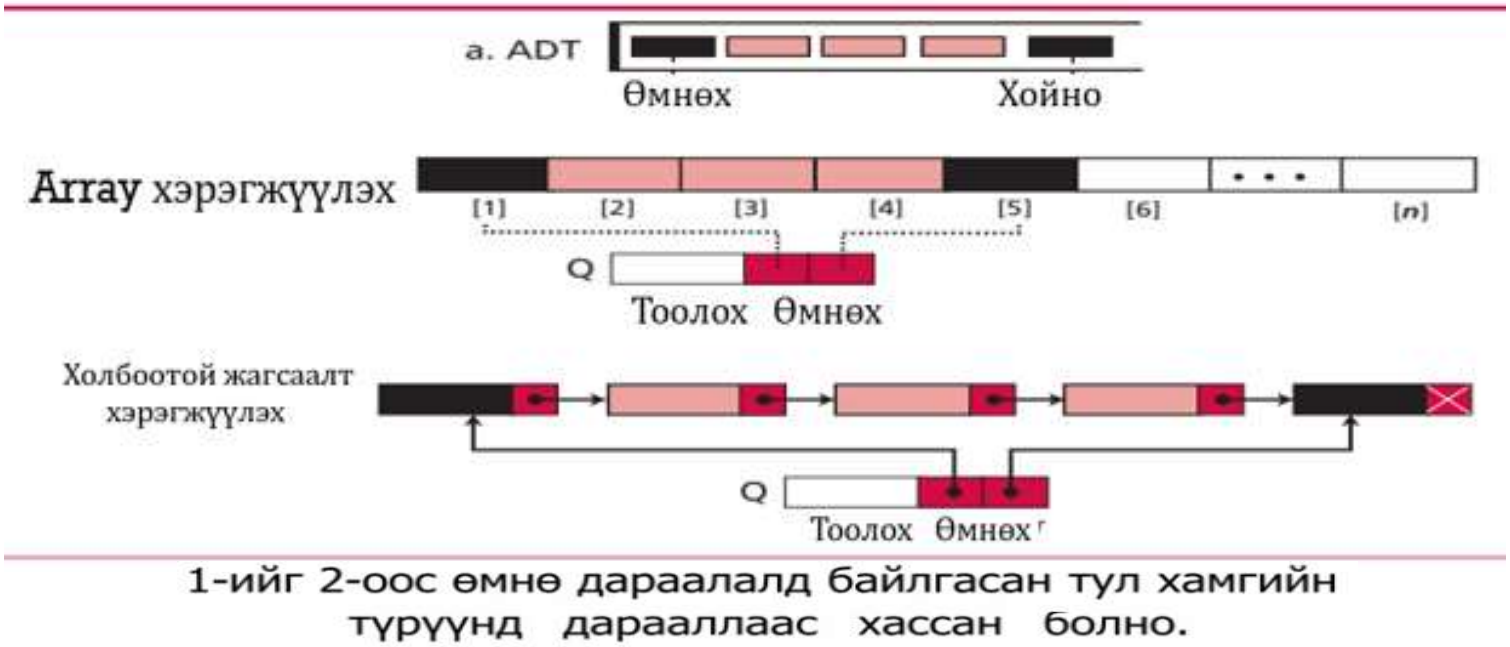
- Хоёр дахь талбар нь урд талын элементийн индексийг агуулсан бүхэл тоо.

- Гурав дахь талбар нь мөн бүхэл тоо бөгөөд арын элементийн индексийг агуулна.

Холбоостой жагсаалтын хэрэгжилт нь дээрхтэй ижил төстэй:
Энд дарааллын нэртэй нэмэлт зангилаа, энэ зангилаа нь мөн гурван талбартай: *тоо, урд талын элементийг заадаг заагч, хойд элементийг заадаг заагч.*

Алгоритм

Хэрэгжүүлэлт бүрт дарааллын хувьд тодорхойлсон дөрвөн үйлдлийнхээ дөрвөн алгоритмыг псевдокодоор бичиж болно.
Дараалалд хэрэгтэй дөрвөн алгоритмыг үүсгэхийн тулд эдгээр алгоритмуудыг өөрчилж болно: дараалал, дараалал, дараалал, хоосон.
Дараалалд оруулах нь зөвхөн дарааллын төгсгөлд хийгддэг бөгөөд устгах нь зөвхөн дарааллын эхэнд хийгддэг зураг 18-лд үзүүлсэн.



Зураг 18. Queue хэрэгжүүлэлт

Ерөнхий шугаман жагсаалт

Өмнөх хоёр хэсэгт тодорхойлсон стек, дараалал нь хязгаарлагдмал шугаман жагсаалт юм. Генерал шугаман жагсаалт гэдэг нь жагсаалтын эхэнд, дунд эсвэл төгсгөлд оруулах, устгах гэх мэт үйлдлүүдийг хийх боломжтой жагсаалт юм. Зураг 19-т ерөнхий шугаман жагсаалтийг үзүүлсэн.

Зураг 19. Ерөнхий шугаман жагсаалт (General linear list)

Үргэлжлэл

Бид ерөнхий шугаман жагсаалтыг дараах шинж чанартай элементүүдийн цуглуулга гэж тодорхойлдог.

- ❑ Элементүүд нь ижил төрлийн байна.
- ❑ Элементүүдийг дэс дарааллаар байрлуулсан бөгөөд энэ нь эхний элемент мөн сүүлчийн элементээс тогтоно.
- ❑ Эхнийхээс бусад элемент бүр нь сүүлийнхээс бусад элемент бүр өвөрмөц өмнөхтэй байна
- ❑ Элемент бүр нь түлхүүр талбар бүхий бичлэг юм.
- ❑ Элементүүдийг гол утгад нь үндэслэн эрэмбэлдэг.

Хэдийгээр бид ерөнхий шугаман жагсаалтад олон үйлдлийг тодорхойлж болох ч бид зөвхөн зургааг нь авч үздэг.
Энэ бүлгийн нийтлэг үйлдлүүд: жагсаах, оруулах, устгах, сэргээх, эргүүлэх, хоосон болгох.

List үйлдэл

Жагсаалтын үйлдэл нь хоосон жагсаалтыг үүсгэдэг.

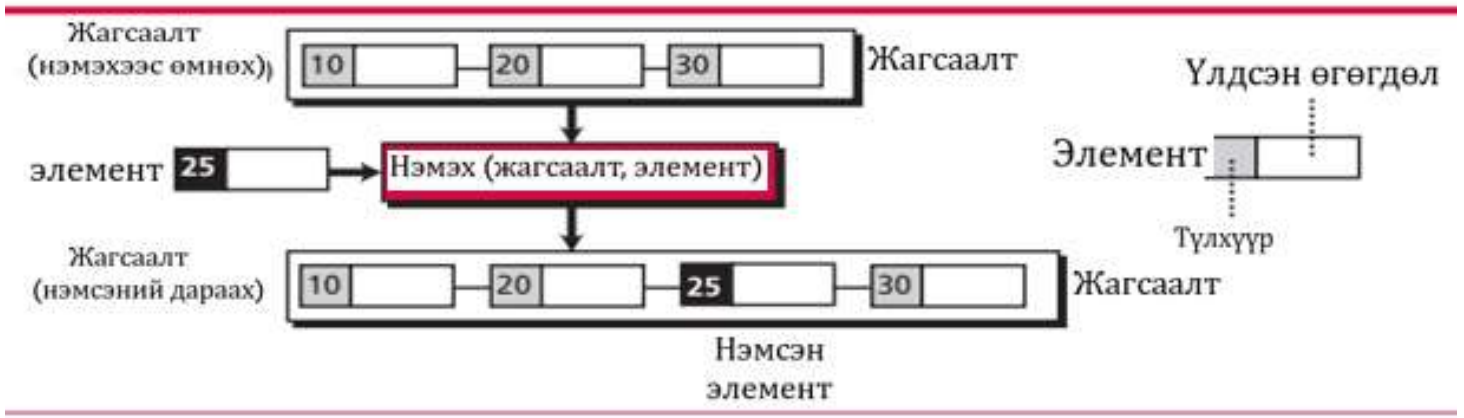
```
list (listName)
```

listName нь үүсгэх ерөнхий шугаман жагсаалтын нэр юм.

Бид ерөнхий шугаман жагсаалт дахь өгөгдлийг эрэмбэлсэн гэж үздэг тул оруулга нь элементүүдийн дарааллыг хадгалахын тулд хийгдэх ёстой. Элементүүдийг хаана байрлуулахыг тодорхойлохын тулд хайлт хийх шаардлагатай. Гэсэн хэдий ч хайлт нь ADT түвшинд биш харин хэрэгжилтийн түвшинд хийгддэг. Нэмж дурдахад, бид энгийн шугаман жагсаалтад давхардсан өгөгдлийг зөвшөөрдөггүй гэж үздэг. Тиймээс бид товчлуурын дарааллыг хадгалсан байршилд элемент оруулдаг.

```
insert (listName, element)
```

List Үйлдэл

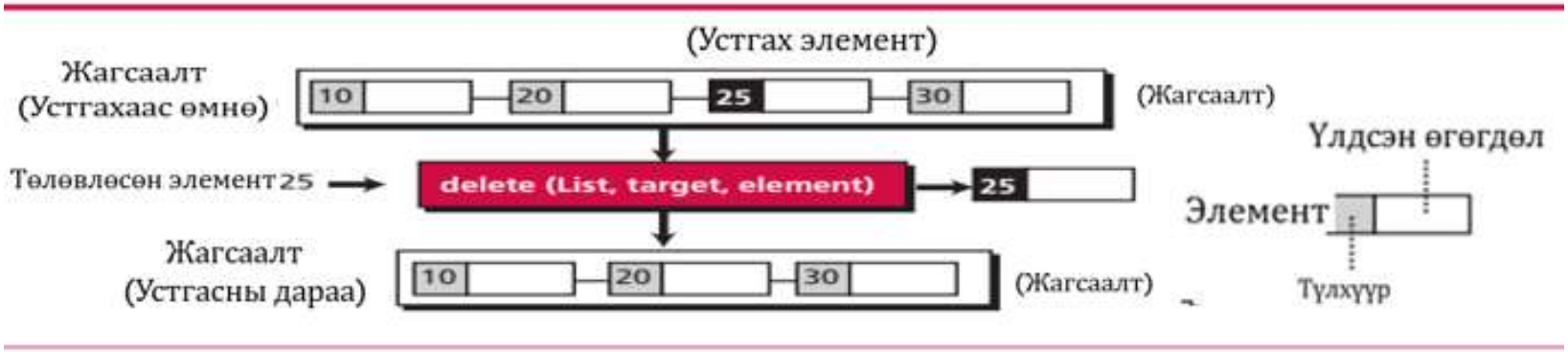


Зураг 20. The insert процесс

Ерөнхий жагсаалтаас устгах (Зураг 20) мөн устгагдах өгөгдлийн байршлыг олохын тулд жагсаалтаас хайх шаардлагатай. Өгөгдлийн байршлыг олсны дараа устгаж болно

```
delete (listName, target, element)
```

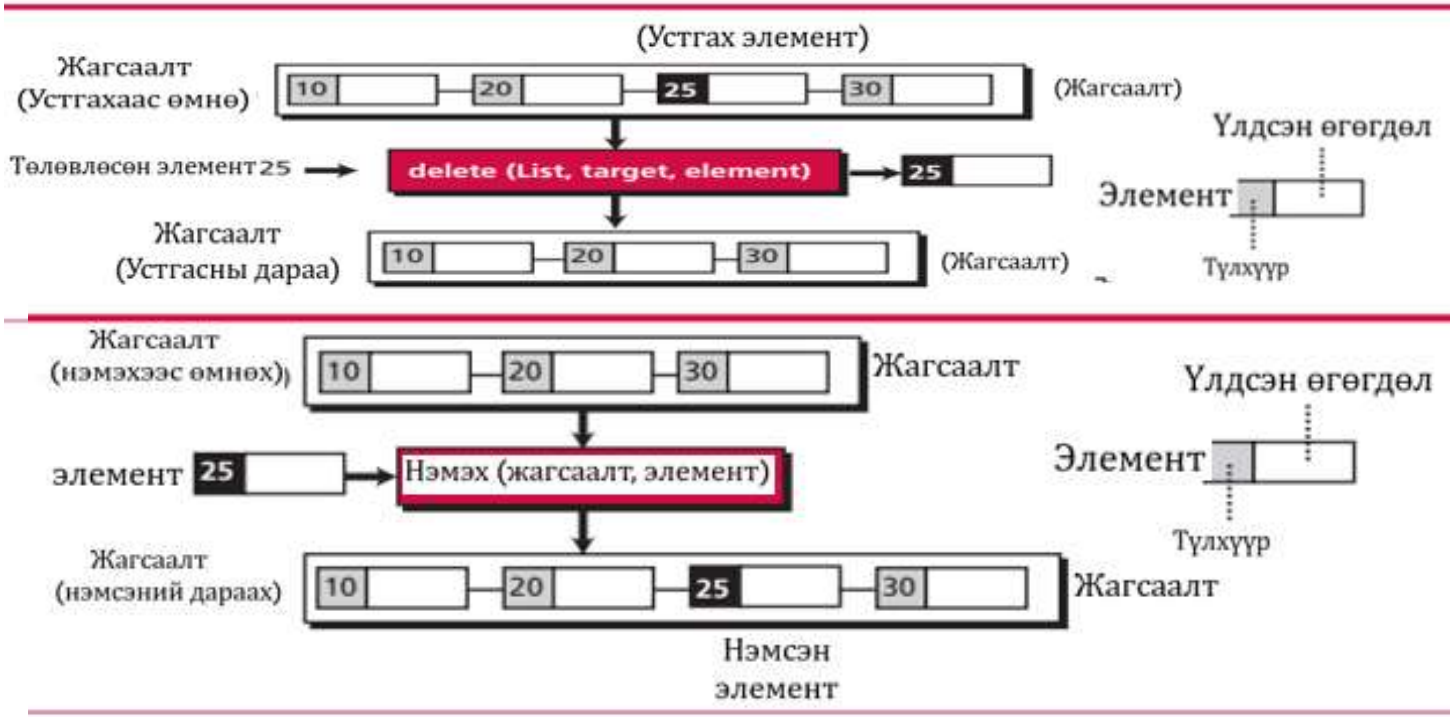
List Үйлдэл



Зураг 21. delete Процесс

зорилт нь жагсаалтын элементүүдийн түлхүүртэй ижил төрлийн өгөгдлийн утга юм. Хэрэв зорилтот утгатай тэнцүү түлхүүр утгатай элемент олдвол тэр элементийг устгана. Устгах үйлдлийг 21-р зурагт графикаар үзүүлэв.

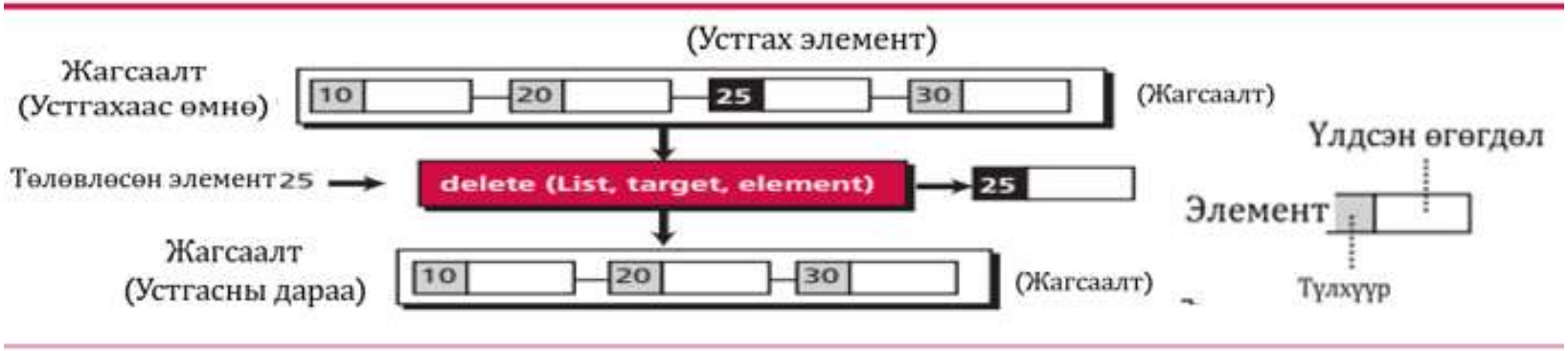
Харьцуулалт



Зураг 22. Харьцуулалт

Оруулах, устгахтай адил ерөнхий жагсаалтыг эхлээд хайх хэрэгтэй бөгөөд хэрэв өгөгдөл олдвол буцааж авах боломжтой (Зураг 20) .

Delete үйлдэл



Зураг 23. delete процесс

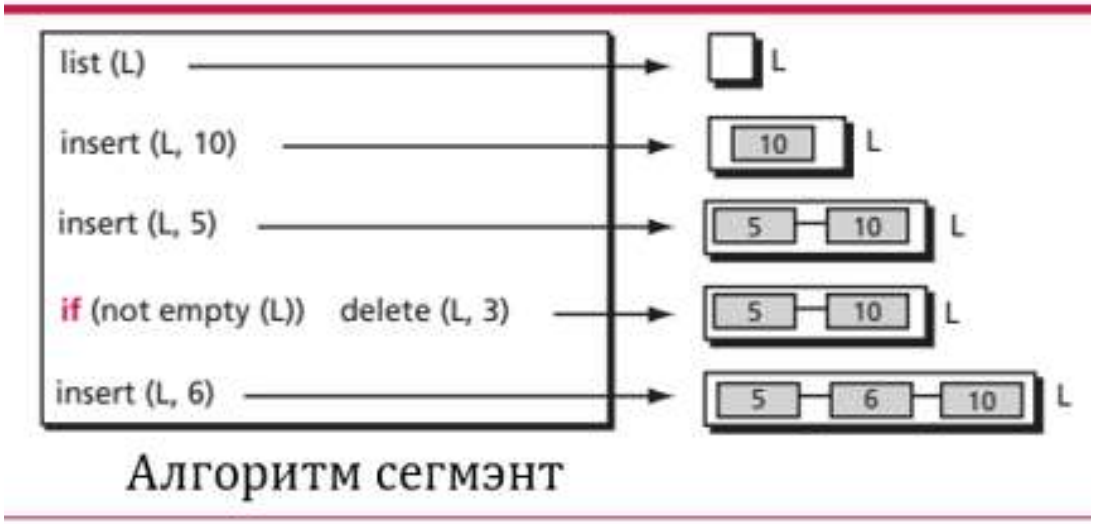
Оруулах, устгахтай адил ерөнхий жагсаалтыг эхлээд хайх хэрэгтэй бөгөөд хэрэв өгөгдөл олдвол буцааж авах боломжтой.

```
retrieve (listName, target, element)
```

зорилт нь жагсаалтын элементүүдийн түлхүүртэй ижил төрлийн өгөгдлийн утга юм. Зураг 23-д сэргээх ажиллагааг графикаар харуулав. Хэрэв зорилтот утгатай тэнцүү түлхүүр утгатай элемент олдвол тухайн элементийн хуулбарыг татаж авах боловч элемент жагсаалтад хэвээр үлдэнэ.

Ерөнхий шугаман жагсаалтын үйлдлүүд

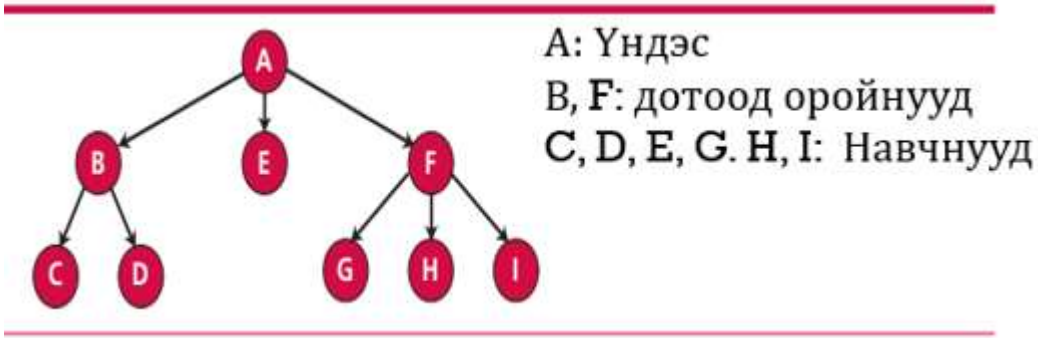
Ерөнхий шугаман жагсаалтыг элементүүдэд санамсаргүй байдлаар эсвэл дарааллаар хандах тохиолдолд ашигладаг. Жишээлбэл, коллежид семестер бүрт элссэн оюутнуудын мэдээллийг хадгалахад шугаман жагсаалтыг ашиглаж болно (Зураг 24) .



Зураг 24. Жишээ

Мод

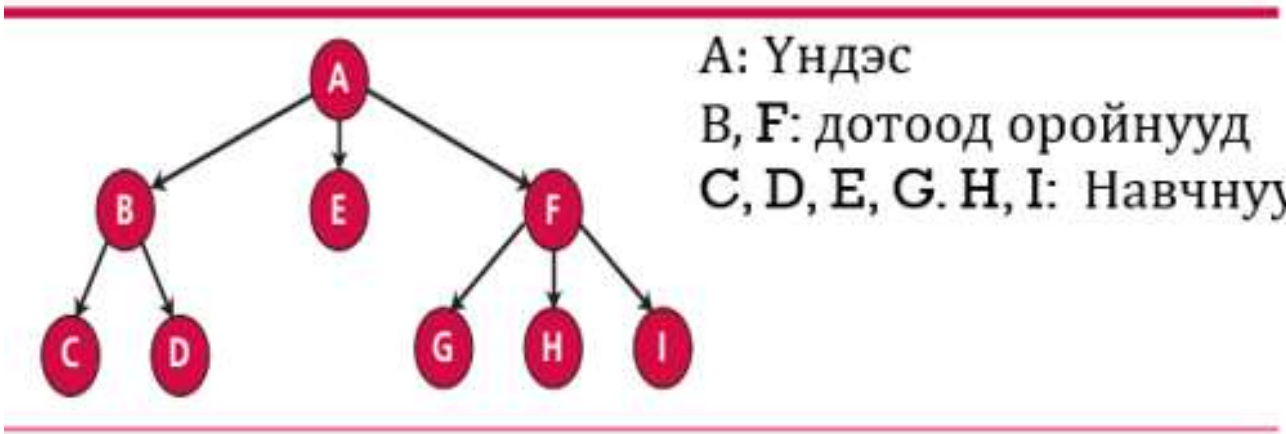
Мод нь зангилаа (эсвэл орой) гэж нэрлэгддэг хязгаарлагдмал олонлог элементүүд ба зангилааны хосыг холбодог нум гэж нэрлэгддэг чиглүүлсэн шугамуудын хязгаарлагдмал багцаас бүрдэнэ (Зураг 24) . Хэрэв мод хоосон биш бол үндэс гэж нэрлэгддэг зангилааны аль нэг нь ирж буй нумгүй болно. Модны бусад зангилаанууд нь дараалсан нумуудын дараалал болох өвөрмөц замыг дагах замаар үндэснээс нь хүрч болно. Модны бүтцийг ихэвчлэн дээд талд нь үндсийг нь доош нь доош нь татдаг



Зураг 25. Tree бүтэц

Мод

Мод (Tree) : 1 үндэстэй, түүнээс салаалсан мөчиртэй өгөгдлийн бүтцийг хэлнэ. Хамгийн түгээмэл бөгөөд энгийн нь хоёртын мод. 1 үндэс → 2 мөчир салаална. Тэдгээр нь үндэс-н хувьд дотоод оройнууд буюу хүүхэд болох ч цааш салаалах юм бол өөрсдөө эцэг болдог. Харин 1 түвшинд байгаа нь ах дүүс болж хамгийн сүүлийн түвшинд байгаа нь навч гэж нэрлэгддэг (Зураг 26) . General Tree, Binary Tree, Binary Search Tree, AVL Tree, Red-Black Tree, N-ary Tree гэх мэт олон янз бий. Мөн Heap, Trie гэх мэт онцгой хувилбарууд ч байдаг.



Зураг 26. Tree бүтцийн илэрхийлэл

Мод бүтцийн Нум

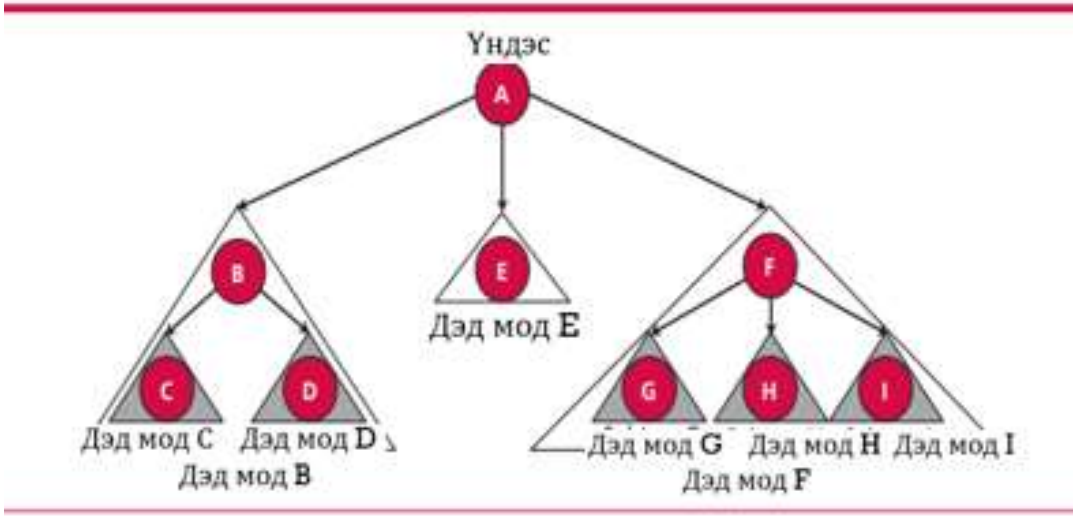
Бид модны оройг үндэс, навч, дотоод зангилаа гэсэн гурван төрөлд хувааж болно. Хүснэгт 12.1-д зангилааны төрөл тус бүрээр зөвшөөрөгдөх гарах ба ирж буй нумын тоог харуулав.

Хүснэгт 12.1 Ирж буй болон гарах нумын тоо

Оройн төрөл	Оролтын нум	Гаралтын нум
Үндэс	0	0 or more
Навч	1	0
Дотоод	1	1 түүнээс олон

Өгөгдсөн зангилаанаас шууд хандах боломжтой (нэг нумаар) зангилааг хүүхэд гэж нэрлэдэг: хүүхдэд шууд хандах зангилааг эцэг эх гэж нэрлэдэг. Нийтлэг эцэг эхтэй зангилаануудыг ах дүүс гэж нэрлэдэг. Зангилааны удам нь тухайн зангилаагаар хүрч болох бүх зангилаа бөгөөд бүх удамд хүрч болох зангилааг өвөг гэж нэрлэдэг. Модны зангилаа бүр дэд модтой байж болно.

Subtree бүтэц

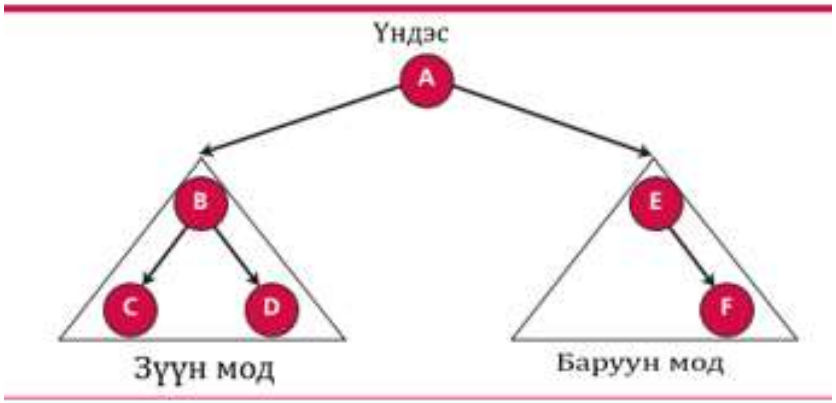


Зураг 27. Subtrees

Хэдийгээр моднууд нь индекс файлууд гэх мэт компьютерийн шинжлэх ухаанд олон хэрэглээтэй боловч тэдгээрийг судлах нь энэ номын хамрах хүрээнээс гадуур юм (Зураг 27) . Бид модыг нэг төрлийн мод болох хоёртын модыг хэлэлцэх оршил болгон танилцуулж байна.

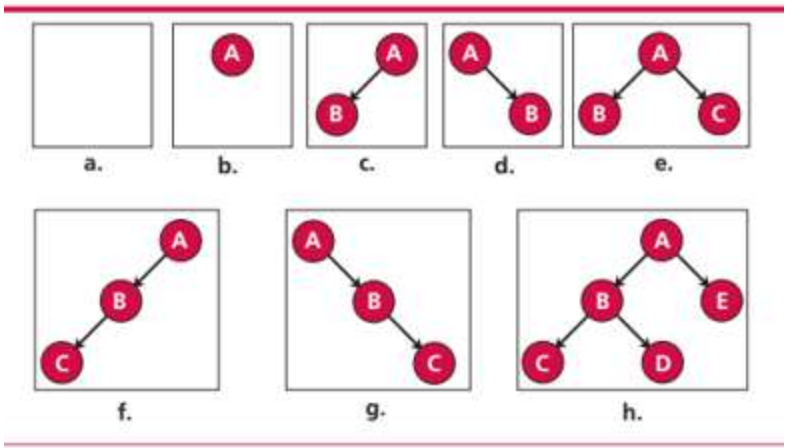
Хоёртын мод

Хоёртын мод гэдэг нь ямар ч зангилаа хоёроос илүү дэд модтой байж болохгүй мод юм. Өөрөөр хэлбэл, зангилаа нь тэг, нэг эсвэл хоёр дэд модтой байж болно. Эдгээр дэд модыг зүүн дэд мод, баруун дэд мод гэж тодорхойлсон. Зураг 28- д хоёр дэд модтой хоёртын модыг үзүүлэв.
Дэд мод бүр нь өөрөө хоёртын мод гэдгийг анхаарна уу.



Зураг 28. Хоёртын мод (A binary tree)

Хоёртын модны рекурсив тодорхойлолт



Зураг 29. Хоёртын модны жишээ

бүтэц эсвэл ADT-ийг рекурсив байдлаар тодорхойлж болно. Хоёртын модны рекурсив тодорхойлолтыг доор өгөв. Энэ тодорхойлолт дээр үндэслэн хоёртын мод нь үндэстэй байж болох ч дэд мод бүр үндэстэй байж болохыг анхаарна уу (Зураг 29) .

Хоёртын модны үйлдлүүд

Хоёртын модны хувьд тодорхойлсон хамгийн нийтлэг зургаан үйлдэл нь мод (хоосон мод үүсгэх) оруулах, устгах, авах, хоосон болгох, дамжих үйлдлүүд юм. Эхний тав нь нарийн төвөгтэй бөгөөд энэ номын хамрах хүрээнээс давсан. Бид энэ хэсэгт хоёртын модыг гатлах талаар ярилцана.

Хоёртын модоор дамжин өнгөрөх нь модны зангилаа бүрийг урьдчилан тодорхойлсон дарааллаар нэг удаа, зөвхөн нэг удаа боловсруулахыг шаарддаг. Дамжуулах дарааллын хоёр ерөнхий хандлага нь гүнээс эхлээд өргөн, эхлээд өргөнөөр дамжин өнгөрөх юм.

Хоёртын мод нь үндэс, зүүн дэд мод, баруун дэд модноос бүрддэгийг харгалзан үзвэл бид гүнээс эхлээд гүйх зургаан өөр дарааллыг тодорхойлж чадна. Компьютерийн эрдэмтэд уран зохиолд эдгээр дарааллын гуравт стандарт нэр өгсөн: нөгөө гурав нь нэргүй боловч
амархан гаргаж авдаг. Стандарт дамжуулалтыг Зураг 12.24-т үзүүлэв.

Жишээ

Зураг 30. Хоёртын модны гүн

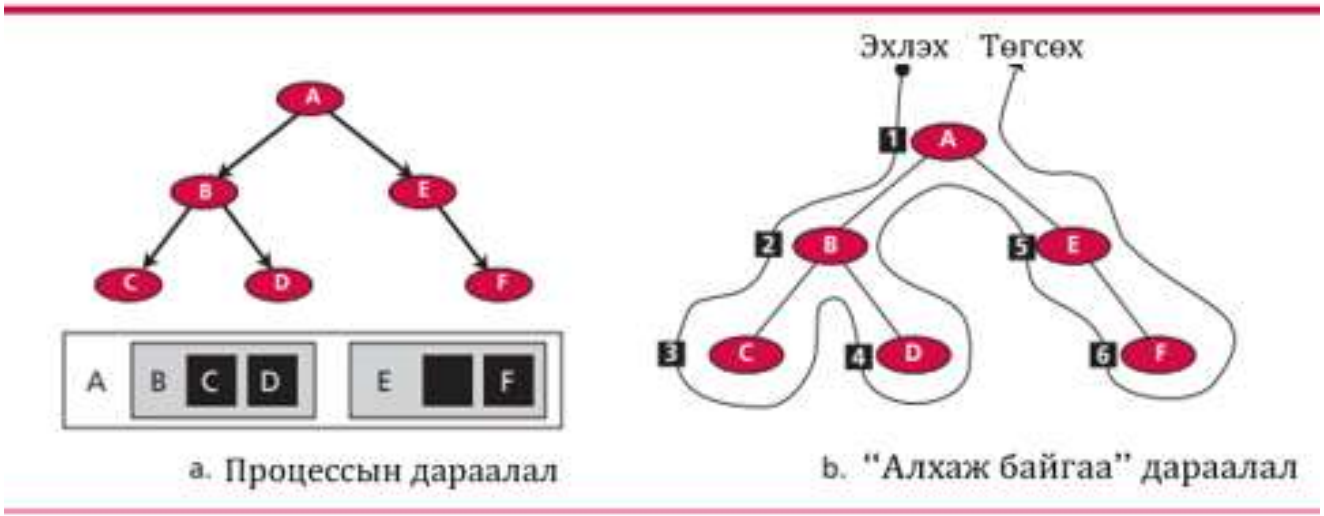
❑ Урьдчилан захиалга өгөх. Урьдчилан эрэмбэлэхийн тулд эхлээд үндсэн зангилаа, дараа нь зүүн дэд мод, дараа нь баруун дэд мод боловсруулагдана. Угтвар угтвар нь үндсэн зангилаа дэд моднуудын өмнө боловсруулагдаж байгааг илтгэнэ.

❑ Захиалга хоорондын зай. Дарааллаар дамжихдаа эхлээд зүүн дэд мод, дараа нь үндсэн зангилаа, эцэст нь баруун дэд мод боловсруулагдана. (Зураг 30) .

❑ Захиалгын дараа дамжих. Захиалгын дараа дамжих үед зүүн ба баруун дэд модыг боловсруулсны дараа эх зангилаа боловсруулагдана. Угтвар бичлэг нь дэд модны дараа үндэс боловсруулагдаж байгааг харуулж байна.

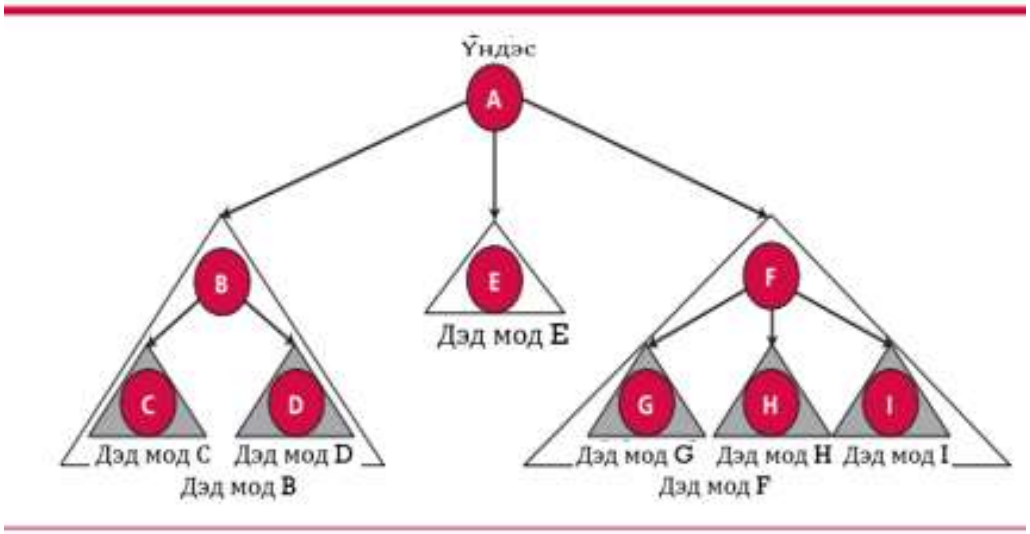
Жишээ

Зураг 31-д бид модны зангилаа бүрийг урьдчилан захиалах замаар хэрхэн зочилдог болохыг харуулж байна. Зураг дээр мөн алхах дарааллыг харуулав. Урьдчилан захиалгын дагуу бид зангилаа зүүн талаас нь хүрдэг. Зангилаануудыг дараах дарааллаар зочилно: A, B, C, D, E, F.



Зураг 31. Хоёртын модны жишээ

Subtree бүтэц

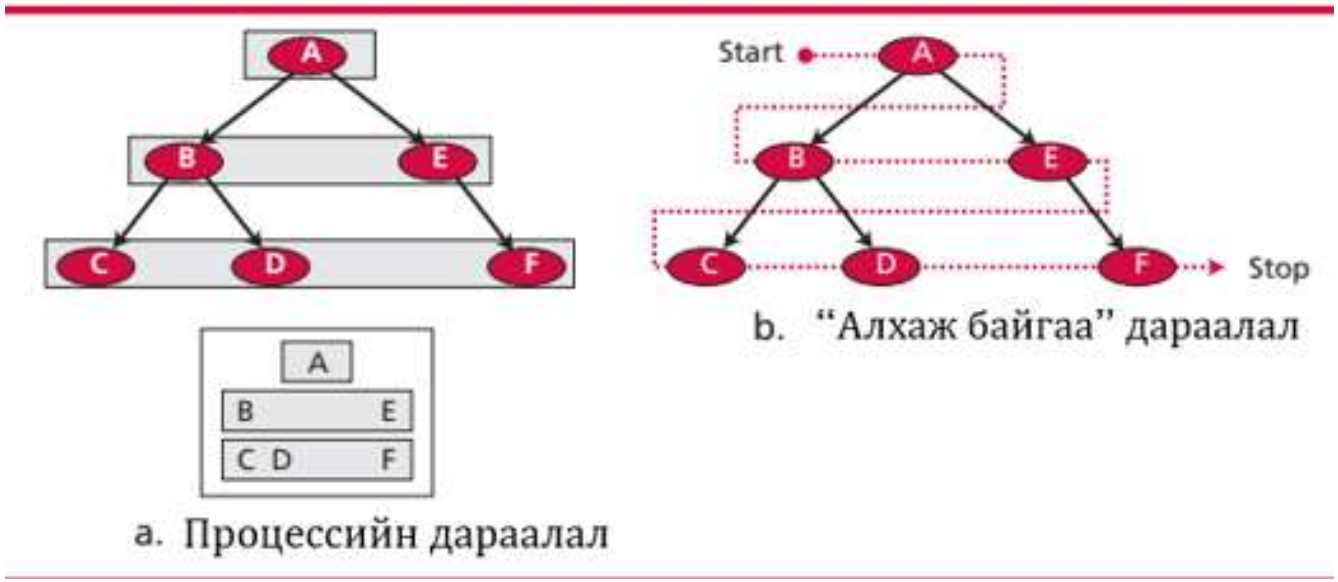


Зураг 31. Subtrees Модны салаанууд

Хэдийгээр моднууд нь индекс файлууд гэх мэт компьютерийн шинжлэх ухаанд олон хэрэглээтэй боловч тэдгээрийг судлах нь энэ номын хамрах хүрээнээс гадуур юм (Зураг 31) . Бид модыг нэг төрлийн мод болох хоёртын модыг хэлэлцэх оршил болгон танилцуулж байна.

Жишээ

Зураг 32-д бид модны зангилаа бүрийг өргөн-эхний хөндлөн огтлолцолоор хэрхэн зочилж байгааг харуулж байна. Зураг дээр мөн алхах дарааллыг харуулав. Замын дараалал нь A, B, E, C, D, F байна.



Зураг 32. Жишээ

Хоёртын модны хэрэглээ

Хоёртын мод нь компьютерийн шинжлэх ухаанд олон хэрэглээтэй байдаг. Энэ хэсэгт бид зөвхөн хоёрыг нь дурдъя: Хаффманы кодчилол ба илэрхийллийн мод.

Хаффман кодчилол нь тэмдэгтүүдийн мөрөөс хувьсах урттай хоёртын код үүсгэхийн тулд хоёртын модыг ашигладаг шахалтын техник юм.

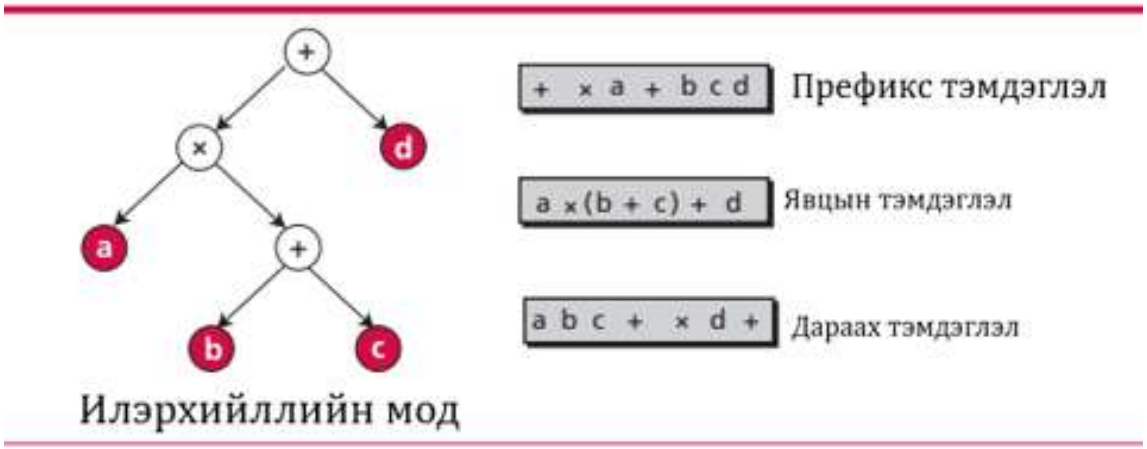
Арифметик илэрхийлэл нь infix, postfix, prefix гэсэн гурван өөр хэлбэрээр илэрхийлэгдэж болно. Infix тэмдэглэгээнд оператор нь хоёр операндын хооронд ирдэг. Postfix тэмдэглэгээнд оператор нь хоёр операндынхоо ард, угтвар тэмдэглэгээнд хоёр операндын өмнө ирдэг. Эдгээр форматыг A ба B хоёр операнд нэмэхийн тулд доор харуулав.

Хэдийгээр бид алгоритмууд болон програмчлалын хэлнүүдэд infix тэмдэглэгээг ашигладаг ч хөрвүүлэгч тэдгээрийг үнэлэхээсээ өмнө ихэвчлэн постфиксийн тэмдэглэгээ болгон өөрчилдөг. Энэ хувиргалтыг хийх нэг арга бол илэрхийллийн мод үүсгэх явдал юм.

Хоёртын модны хэрэглээ

Prefix: + A B	Infix: A + B	Postfix: A B +

Илэрхийллийн модны үндэс ба дотоод зангилаа нь оператор, навч нь операнд юм. Гурван стандарт дамжуулалт (урьдчилан захиалга, дараалал, дараалал: Зураг 12.4) нь infix, postfix, угтвар гэсэн гурван өөр илэрхийллийн форматыг төлөөлдөг. Эрэмбэ дарааллаар дамжих нь infix илэрхийллийг, дарааллын дарааллаар дамжин өнгөрөх нь дараах илэрхийллийг, урьдчилсан эрэмбэлэх нь угтвар илэрхийллийг үүсгэдэг. Зураг 33-д илэрхийлэл ба түүний илэрхийллийн модыг үзүүлэв. Зөвхөн infix тэмдэглэгээнд хаалт хэрэгтэй гэдгийг анхаарна уу.

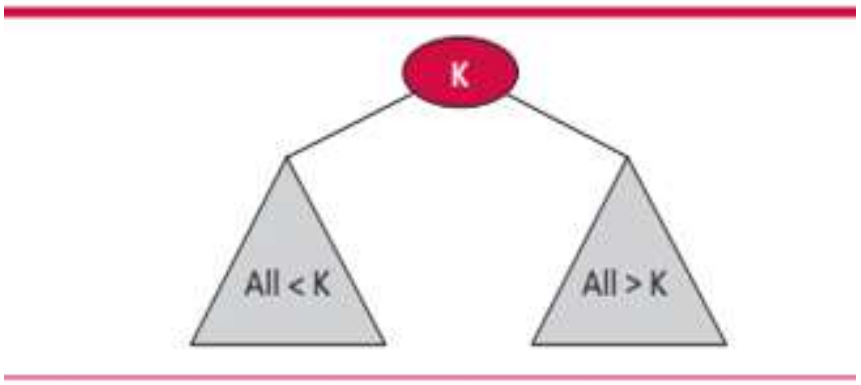


Зураг 33. Expression tree

Хоёртын модны хэрэглээ BSTs

Хоёртын модыг массив эсвэл холбогдсон жагсаалт ашиглан хэрэгжүүлж болно. Холбоосын жагсаалтын хэрэгжилт нь устгах, оруулахад илүү үр дүнтэй бөгөөд илүү түгээмэл байдаг.

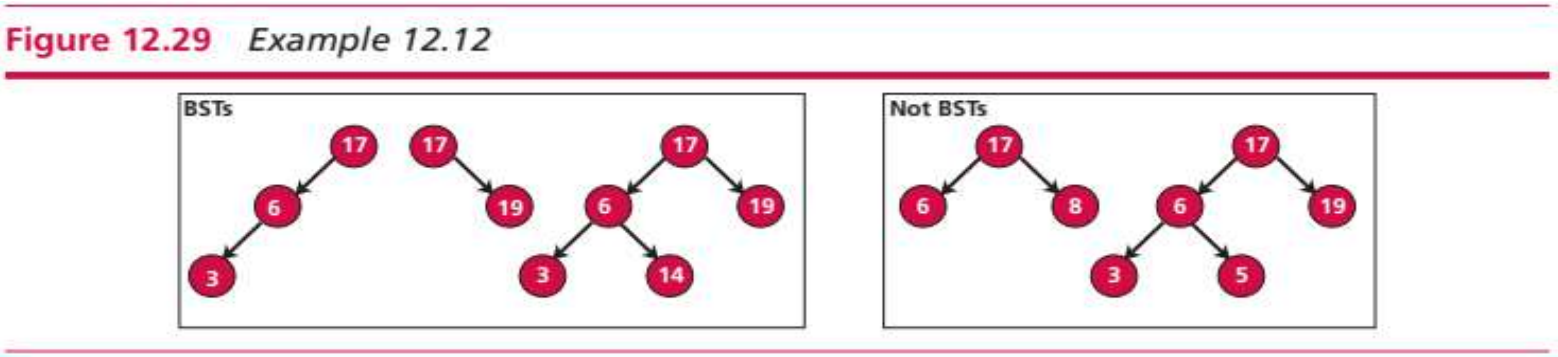
Хоёртын хайлтын мод (BST) нь нэг нэмэлт шинж чанартай хоёртын мод юм: зангилаа тус бүрийн гол утга нь зүүн дэд мод бүрийн бүх зангилааны гол утгаас их, баруун дэд мод бүрийн бүх зангилааны утгаас бага байна. Зураг 34-д санааг харуулав.



Зураг 34. Binary search tree (BST)

Хоёртын модны хэрэглээ

Зураг 35-д BST байдаг, зарим нь биш хоёртын модыг харуулав. Хэрэв бүх дэд мод нь BST, бүхэл мод нь мөн BST байвал модыг BST гэдгийг анхаарна уу.



Зураг 35. Example 12.12

BST-ийг сонирхолтой болгодог өөр нэг онцлог нь бид 8-р бүлэгт ашигласан хоёртын хайлтын хувилбарыг хоёртын хайлтын модонд ашиглаж болно. Зураг 35-д BST хайлтын UML-г харуулав.

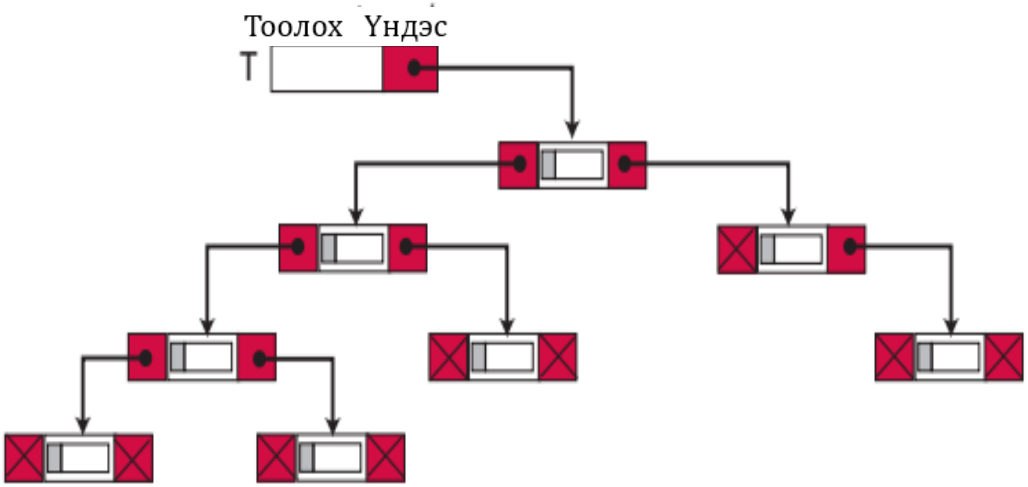
Хоёртын модны хэрэглээ

Хоёртын хайлтын модны ADT нь ижил үйлдэлтэй ерөнхий шугаман жагсаалтад бидний тодорхойлсонтой төстэй юм. Үнэндээ бид өнөөдөр ерөнхий шугаман жагсаалтаас илүү BST жагсаалтыг харж байна. Шалтгаан нь BST хайлт нь шугаман жагсаалтаас илүү үр дүнтэй байдаг: ерөнхий шугаман жагсаалт нь дараалсан хайлтыг ашигладаг бол BST нь хоёртын хайлтын хувилбарыг ашигладаг.

BST-ийг массив эсвэл холбосон жагсаалт ашиглан хэрэгжүүлж болно. Гэсэн хэдий ч холбогдсон жагсаалтын бүтэц нь илүү түгээмэл бөгөөд илүү үр дүнтэй байдаг. Шугаман хэрэгжилт нь зүүн ба баруун гэсэн хоёр заагч бүхий зангилааг ашигладаг. Зүүн заагч нь зүүн дэд модыг, баруун заагч нь баруун дэд модыг заана. Хэрэв зүүн дэд мод хоосон байвал зүүн заагч хоосон байна: баруун дэд мод хоосон байвал баруун заагч нь хоосон байна. Ерөнхий шугаман жагсаалтын холбогдсон жагсаалтын хэрэгжилтийн нэгэн адил BST-тэй холбогдсон жагсаалтын хэрэгжилт нь BST-тэй ижил нэртэй дамми зангилааг ашигладаг.

Хоёртын модны хэрэглээ

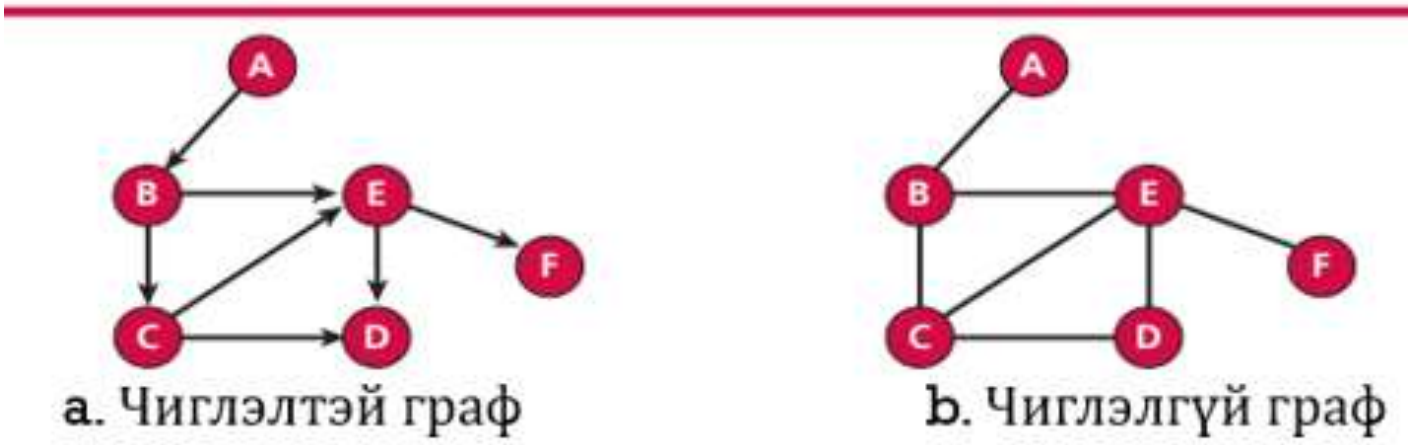
Figure 12.31 *BST implementation*



Зураг 36. BST implementation

Энэхүү дамми зангилааны өгөгдлийн хэсэг нь модны зангилааны тоо зэрэг модны талаарх мэдээллийг агуулж болно. Заагч хэсэг нь модны үндсийг заана.
Зураг 36-д зангилаа бүрийн өгөгдлийн талбар нь бичлэг болох BST- ийг үзүүлэв.

Граф

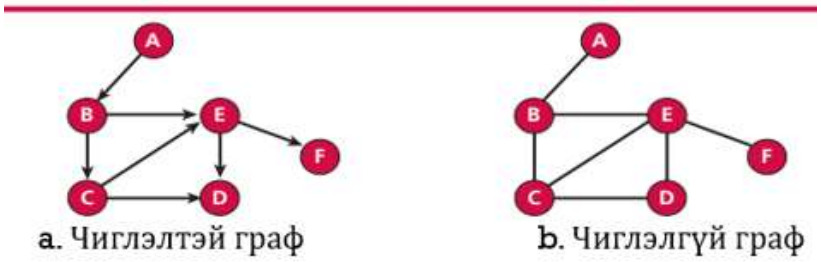


Зураг 37. Graph

Граф (**Graph**) : Орой болон ирмэгүүдийн олонлогоос бүтсэн бүтцийг граф гэж нэрлэдэг. Графын модноос ялгагдах онцлог шинж бол оройнууд хоорондоо холбогдож битүү зам (цагираг) үүсгэж болдог. Мод шиг нэг оройг нь үндэс гэж онцолдоггүй, хоёр орой хоорондоо орох, гарах ирмэгүүдээр холбогдож болдог. Бас **Finite, Infinite, Simple, Trivial, Null, Complete** гэх мэт нэлээн олон төрөлтэй.

Граф

График нь орой гэж нэрлэгддэг олон тооны зангилаа, ирмэг эсвэл нум гэж нэрлэгддэг оройг
холбосон шугамуудын багцаас бүтсэн ADT юм. Мод нь зангилаа нь зөвхөн нэг эцэг эхтэй байж болох шаталсан бүтцийг тодорхойлдог бол график дахь зангилаа бүр нэг буюу хэд хэдэн эцэг эхтэй байж болно. Графикууд нь чиглүүлсэн эсвэл чиглүүлээгүй байж болно. Чиглүүлсэн график буюу диграфт хоёр оройг холбосон ирмэг бүр нь нэг оройгоос нөгөө орой руу чиглэсэн чиглэлтэй (сумын үзүүрээр зурагт үзүүлсэн) байна. Чиглэлгүй графикт чиглэл байхгүй. Зураг 38- т чиглүүлсэн график (a) ба чиглээгүй график (b) хоёрын жишээг үзүүлэв. График дахь оройнууд нь объект эсвэл ойлголтыг, ирмэг эсвэл нумууд нь эдгээр объект эсвэл ойлголтуудын хоорондын хамаарлыг илэрхийлж болно. Графикийг чиглүүлсэн бол хамаарал нь нэг талтай, хэрэв график чиглээгүй бол харилцаа хоёр талтай байна.



Зураг 38. Graph

Граф

Хотуудын газрын зураг, хотуудыг холбосон замыг компьютерт чиглүүлээгүй график ашиглан дүрсэлж болно. Хотууд нь оройнууд бөгөөд чиглээгүй ирмэгүүд нь тэдгээрийг холбосон замууд юм. Хэрэв бид хотуудын хоорондох зайг харуулахыг хүсвэл жигнэсэн графикийг ашиглаж болох бөгөөд ирмэг бүр нь тухайн ирмэгээр холбогдсон хоёр хотын хоорондох зайг илэрхийлэх жинтэй байдаг.

Графикийн өөр нэг хэрэглээ нь компьютерийн сүлжээнд байдаг (Бүлэг 6). Оройнууд нь зангилаа эсвэл зангилааг төлөөлж болно; ирмэгүүд нь замыг төлөөлж болно. Ирмэг бүр нь нэг зангилаанаас зэргэлдээх төв рүү хүрэх зардлыг тодорхойлдог жинтэй байж болно. Чиглүүлэгч нь график алгоритмыг ашиглан пакетын эцсийн цэг болон өөр хоорондын хамгийн богино замыг олох боломжтой.

Унших ном

Энэ бүлэгт авч үзсэн сэдвүүдийн талаар илүү дэлгэрэнгүй мэдээлэл авахыг хүсвэл дараах номуудыг уншихыг зөвлөж байна.

- Gilberg, R. and Forouzan, B. Data Structures – A Pseudocode Approach with C, Boston, MA: Course Technology, 2005
- Goodrich, M. and Tamassia, R. Data Structures and Algorithms in Java, New York: Wiley, 2005
- Nyhoff, L. ADTs, Data Structures, and Problem Solving with C++, Upper Saddle River, NJ: Prentice-Hall, 2005

АШИГЛАСАН МАТЕРИАЛ

Foundations of Computer Science, Behrouz A. Forouzan, Fourth Edition, Cengage Learning EMEA, 2018

Бүлгийн гарчиг: 12-р бүлэг Abstract data types

АНХААРАЛ ХАНДУУЛСАНД БАЯРЛАЛАА