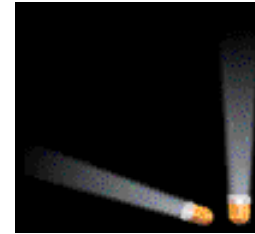
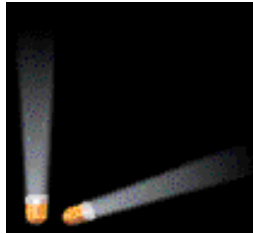
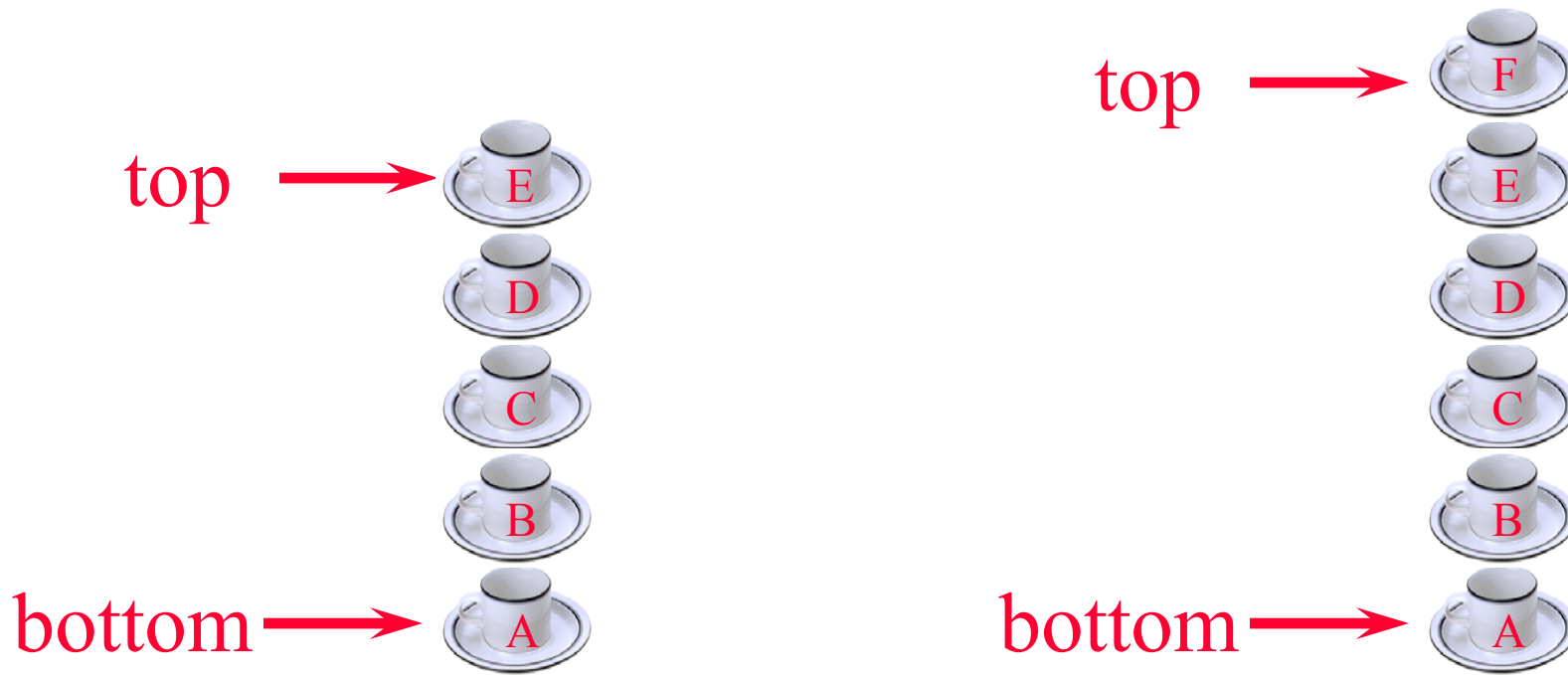


# Стек



- Шугаман жагсаалт.
- Нэг төгсгөлийг нь **top-орой**.
- Нөгөө төгсгөлийг нь **bottom-ёроол**.
- Нэмэлт ба хасалтыг зөвхөн **оройгоос** нь хийнэ.

# Аяганы стек



- Стек аяга нэмэх.
- Стекээс аяга авах.
- Стек бол LIFO жагсаалт.

# Интерфейс - Stack

```
public interface Stack
{
    public boolean empty();
    public Object peek();
    public void push(Object theObject);
    public Object pop();
}
```

# Хаалт хослох

- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)$ 
  - $u$  байршлын нээх хаалтад харгалзах  $v$  байршлын хаах хаалтын хослолыг  $(u,v)$  гэж бичвэл
    - $(2,6)$   $(1,13)$   $(15,19)$   $(21,25)$   $(27,31)$   $(0,32)$   $(34,38)$
- $(a+b))*((c+d)$ 
  - $(0,4)$
  - байршил 5 –ын хаах халтад харгалзах нээх хаалт алга
  - $(8,12)$
  - байршил 7 –ийн нээх хаалтад харгалзах хаах хаалт алга

# Хаалт хослох

- Илэрхийллийг зүүнээс баруун тийш шинжих
- Нээх хаалт тааралдвал түүний байршлыг стект нэмнэ
- Хаах хаалт тааралдвал харгалзах байршлыг стекээс авна

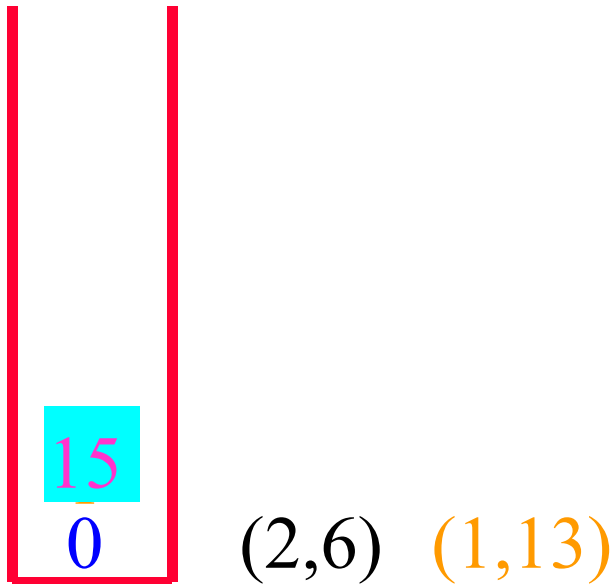
# Жишээ

- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)$

2  
1  
0

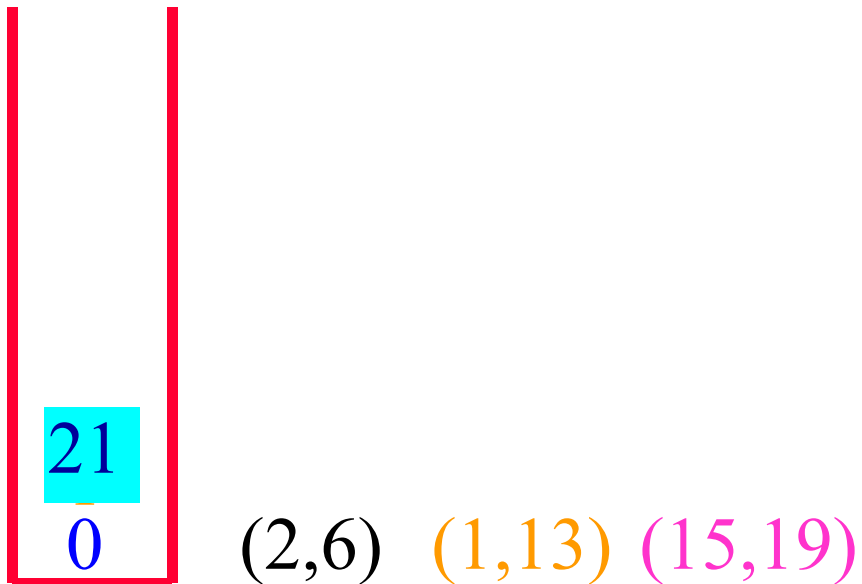
# Жишээ

- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l)/(m-n)$



# Жишээ

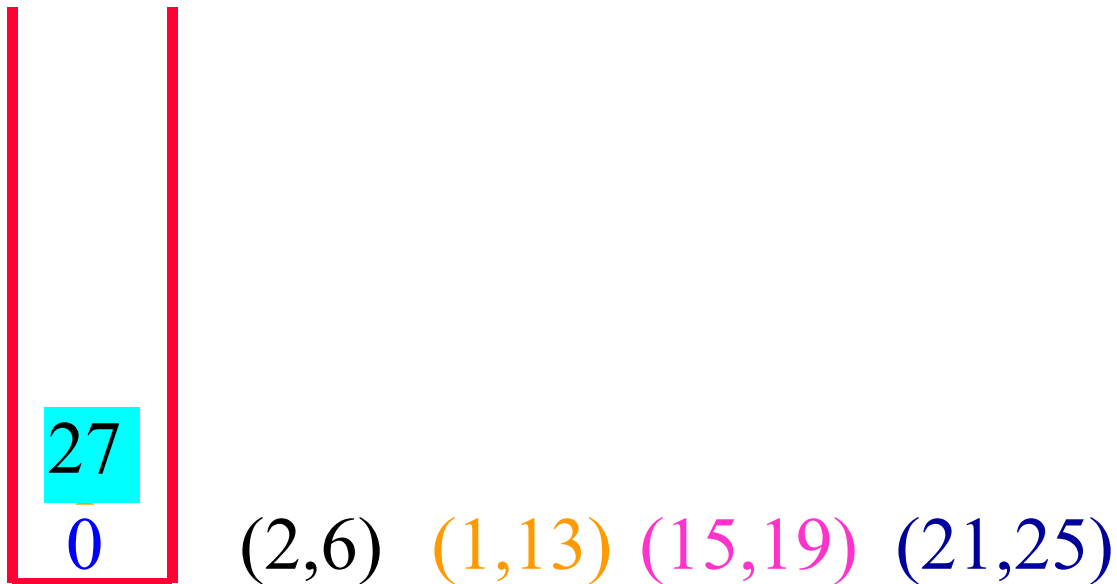
- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l)/(m-n)$





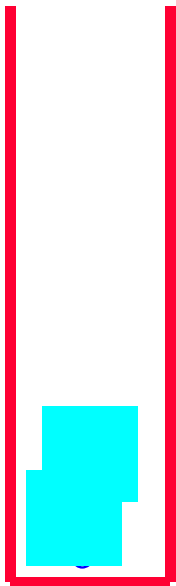
# Жишээ

- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)$



# Жишээ

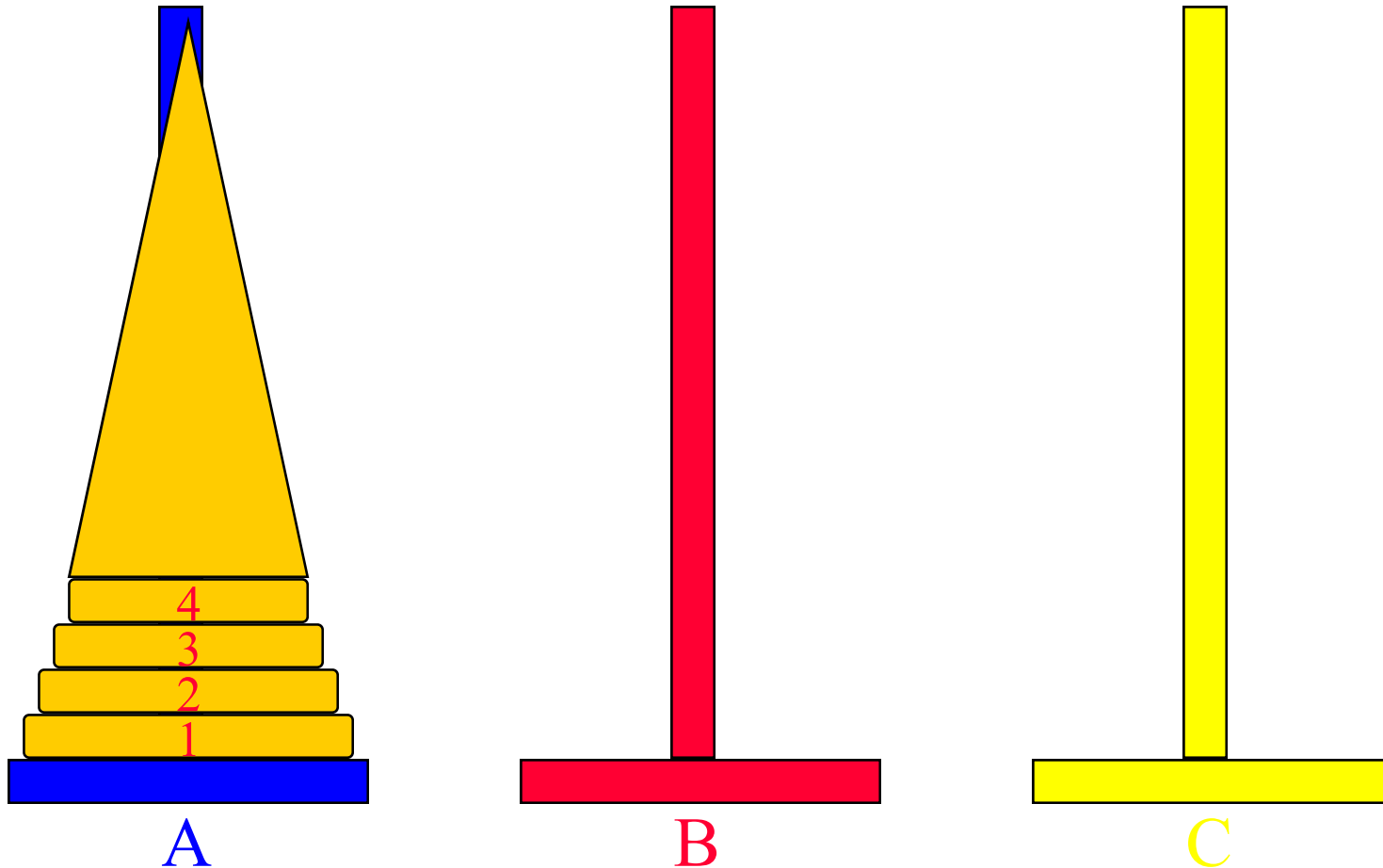
- $((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)$



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

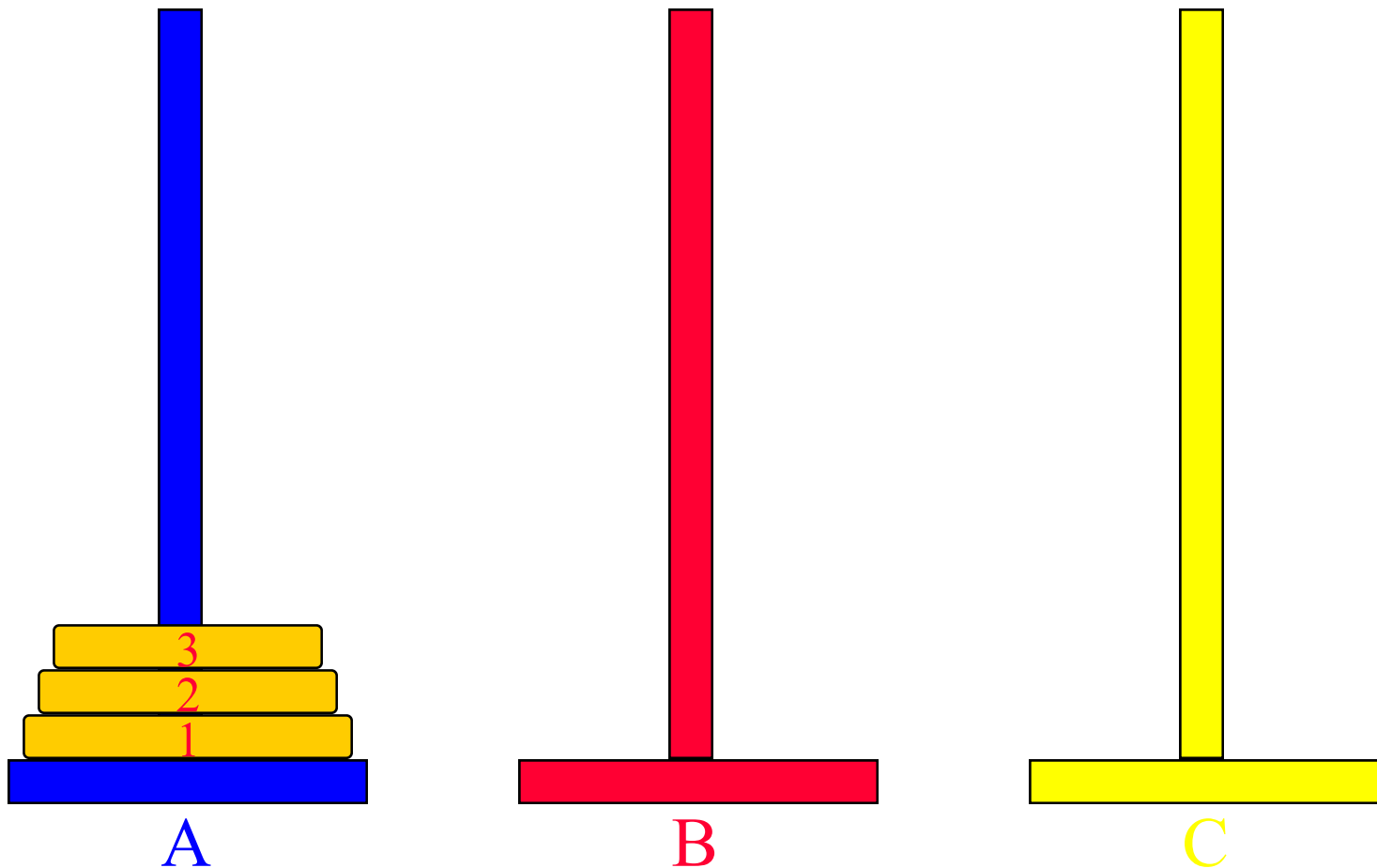
- ГЭХ МЭТ

# Ханойн цамхаг



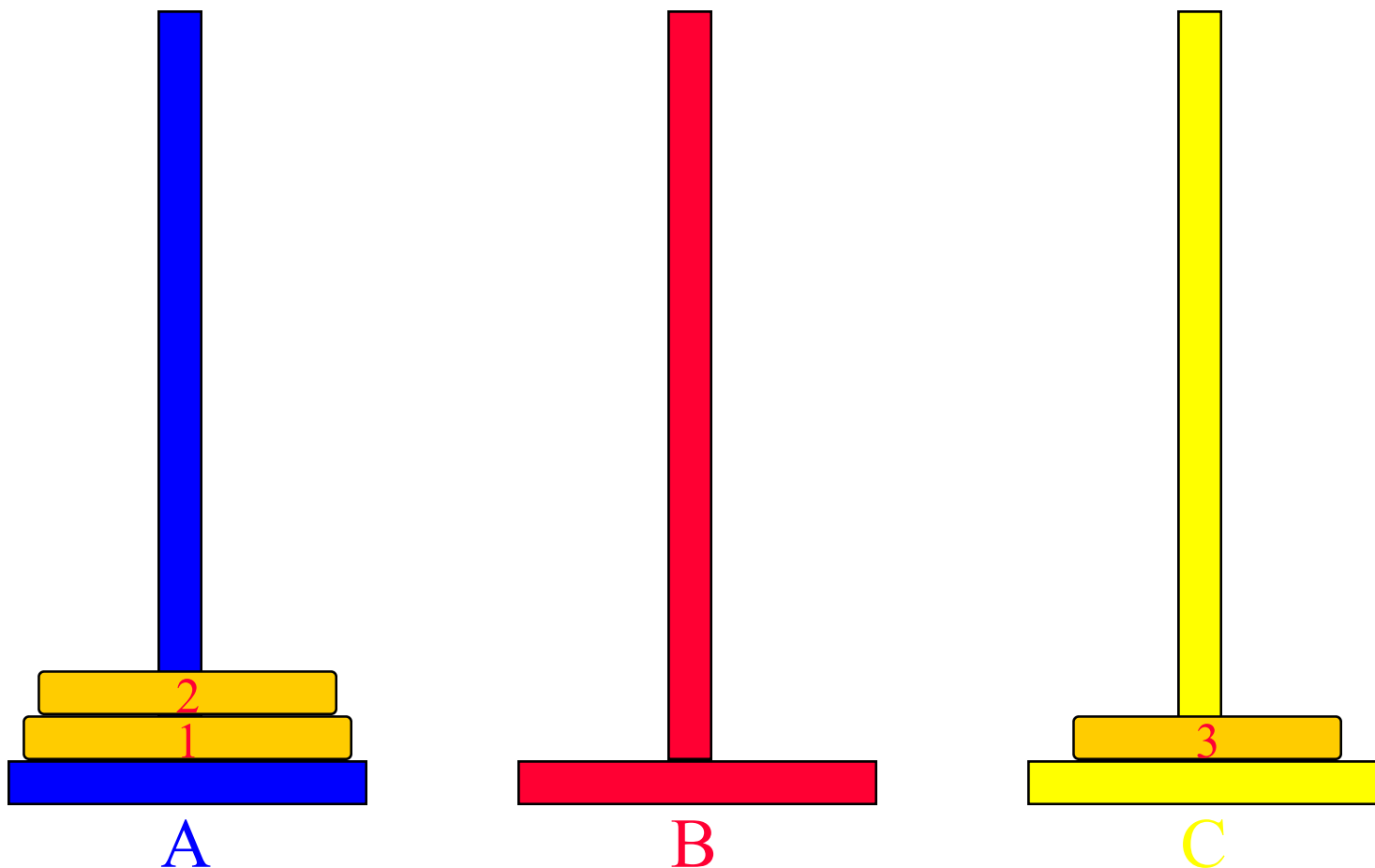
- 64 алтан дискийг A цамхгаас C цамхагт шилжүүлнэ
- Цамхаг бүр стек шиг ажиллана
- Том дискийг жижиг диск дээр тавьж болохгүй

# Ханойн цамхаг



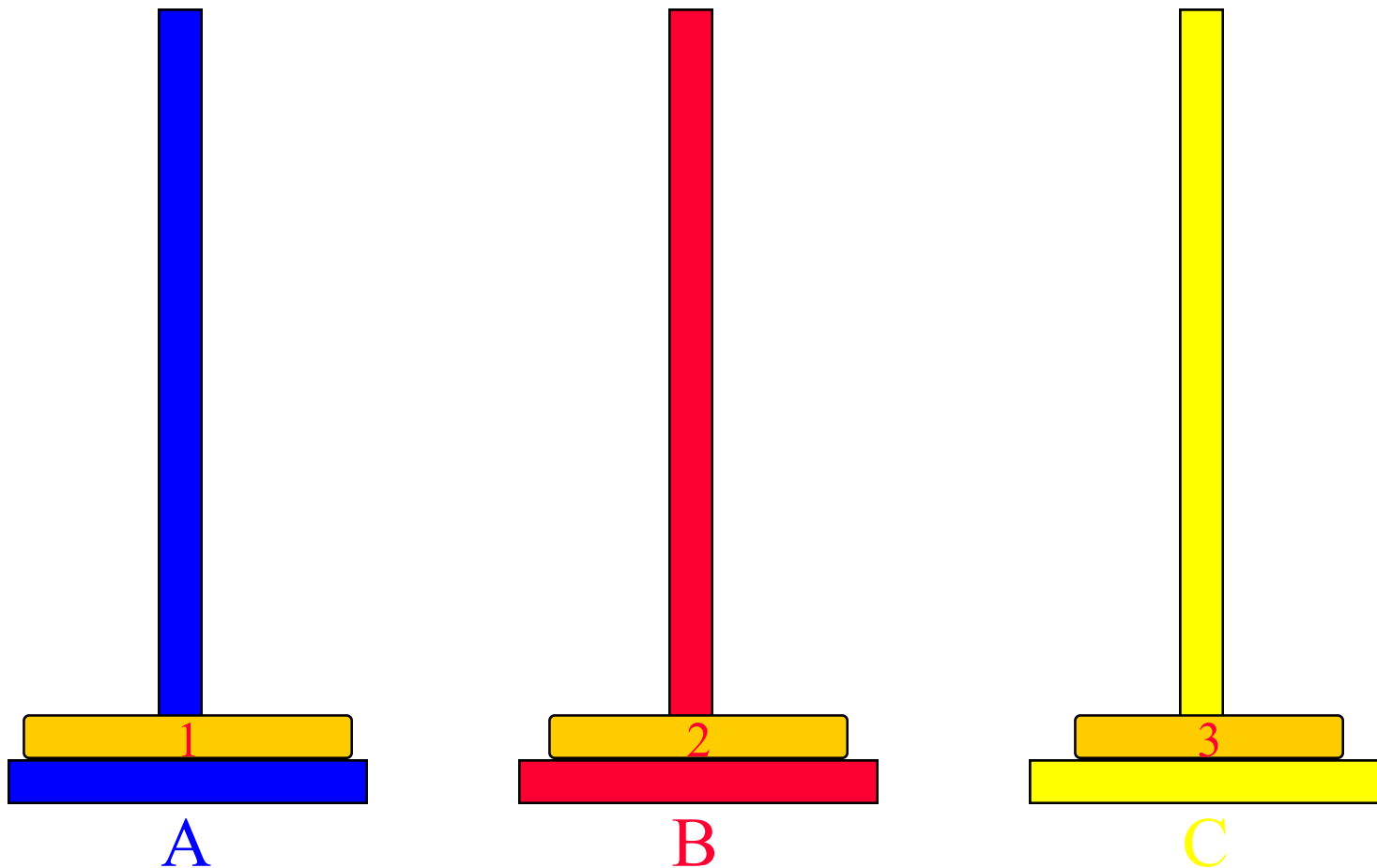
- 3 дисктэй Ханойн цамхаг

# Ханойн цахаг



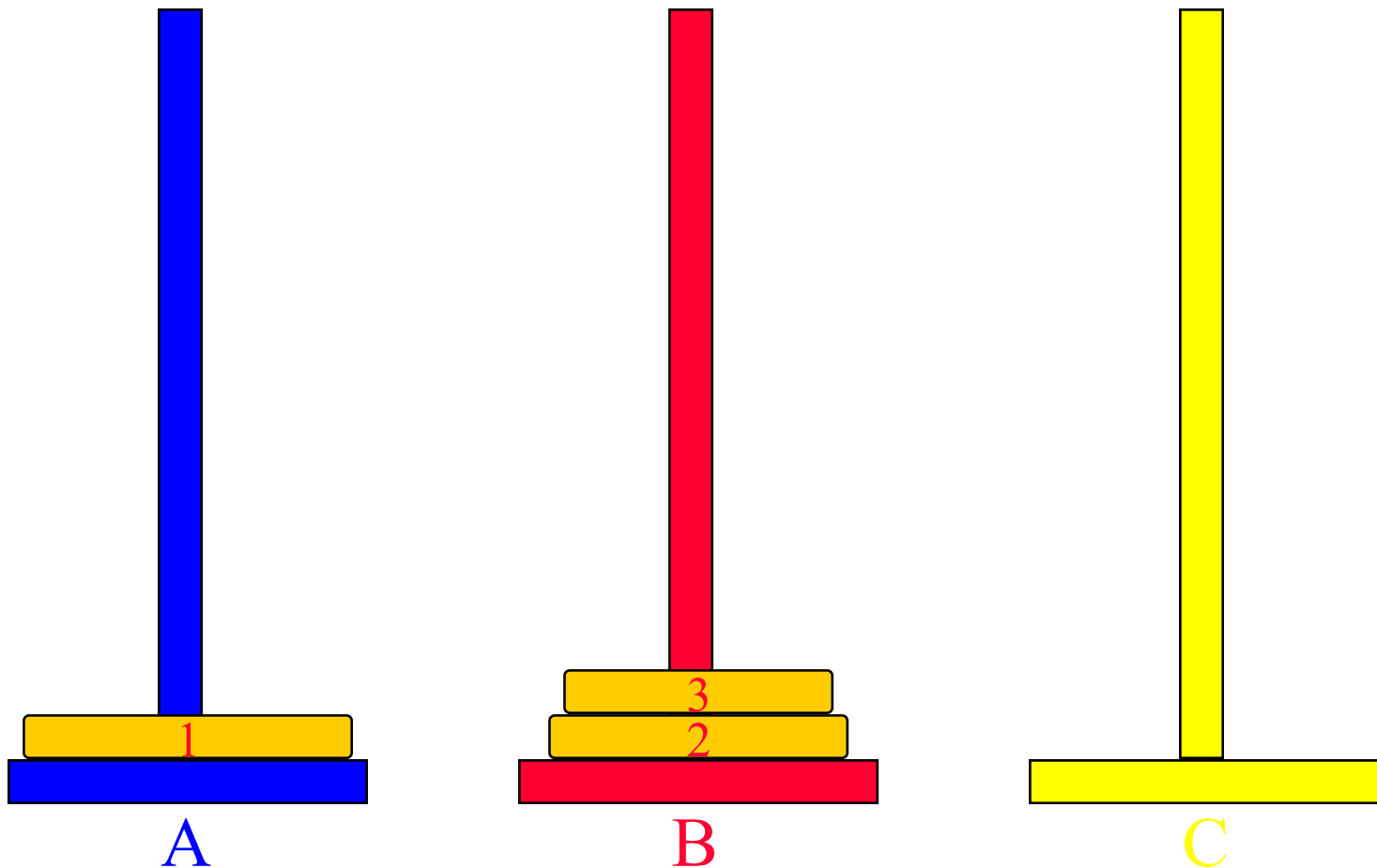
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



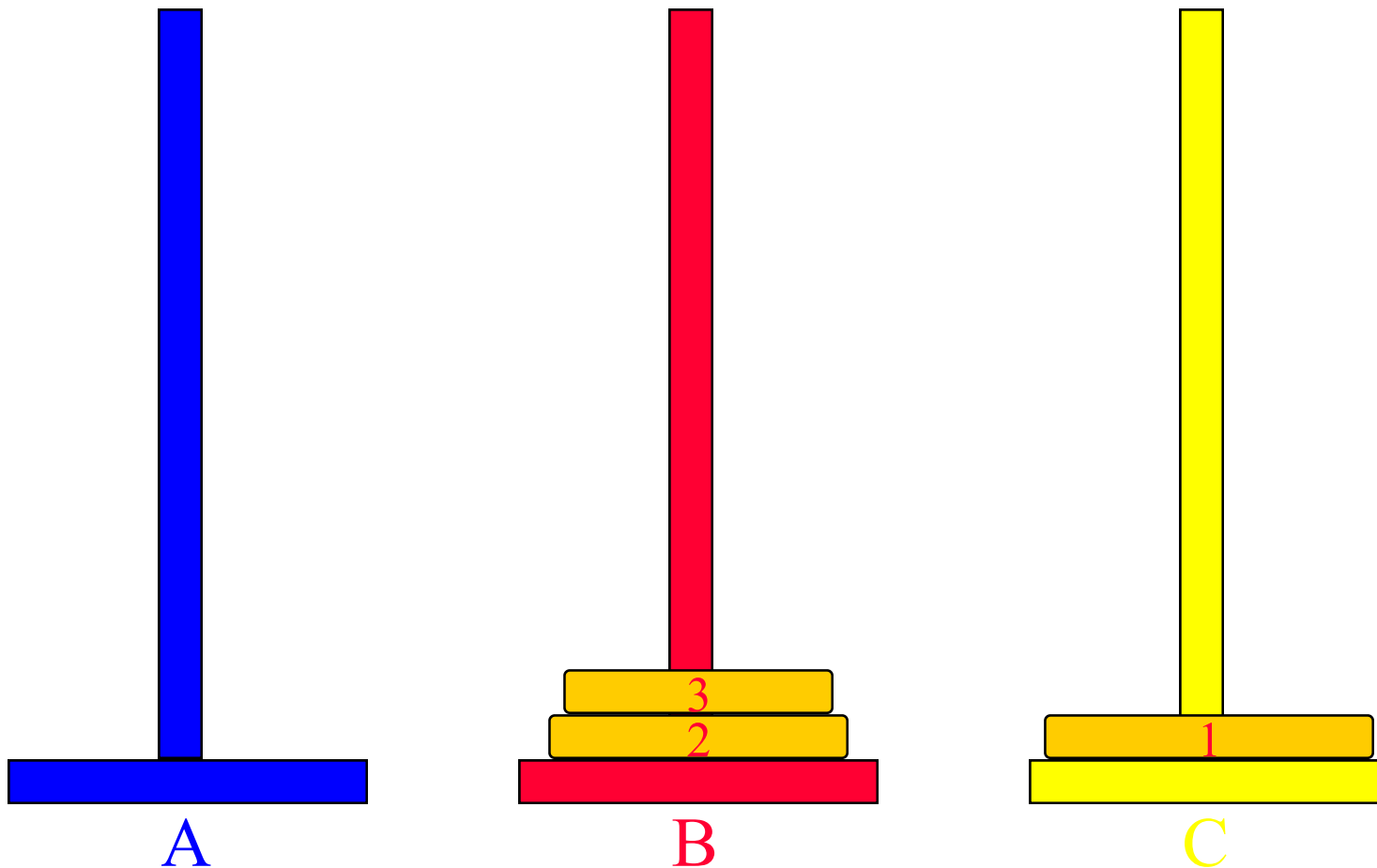
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



- 3 дисктэй Ханойн цамхаг

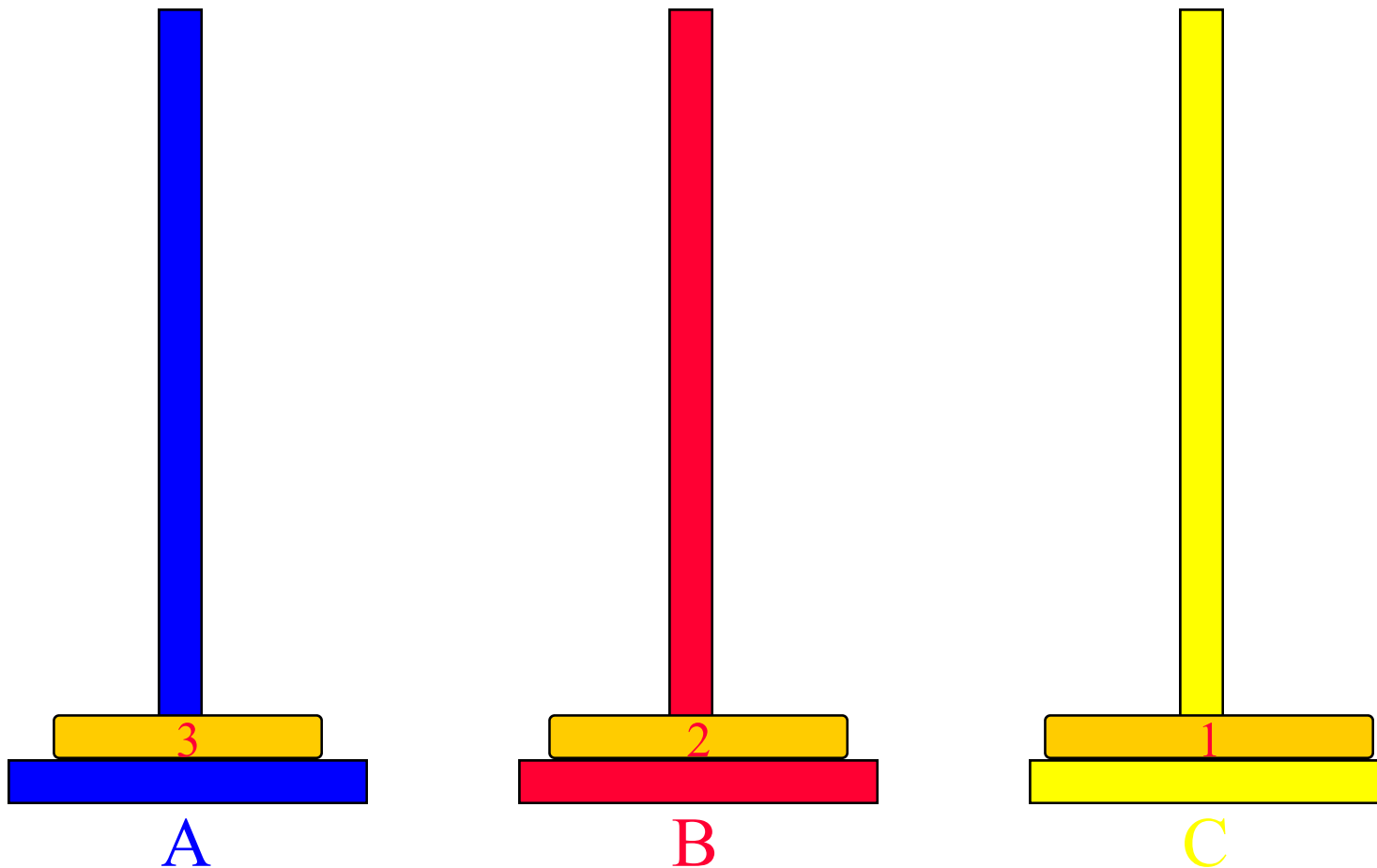
# Ханойн цамхаг



- 3 дисктэй Ханойн цамхаг

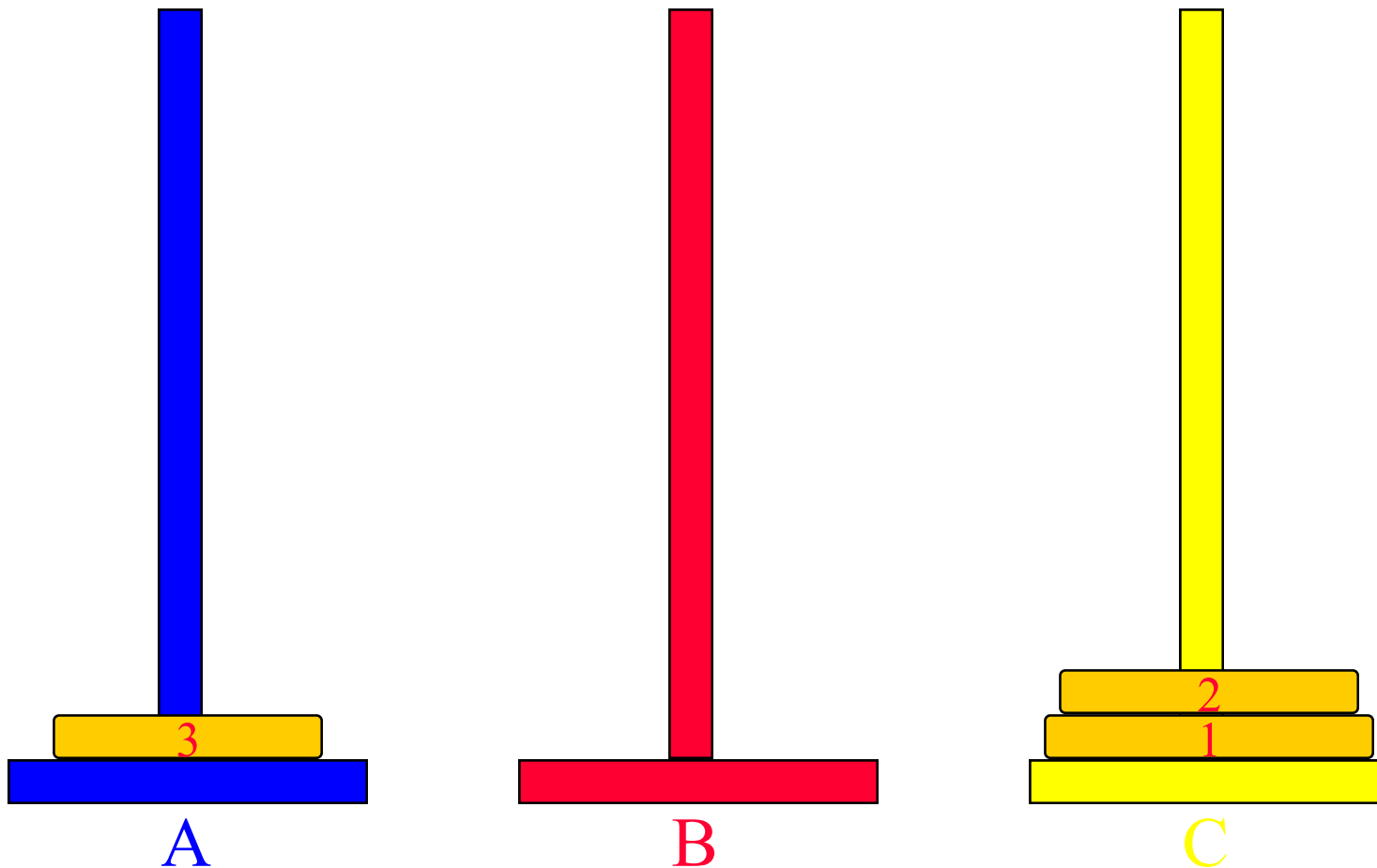


# Ханойн цамхаг



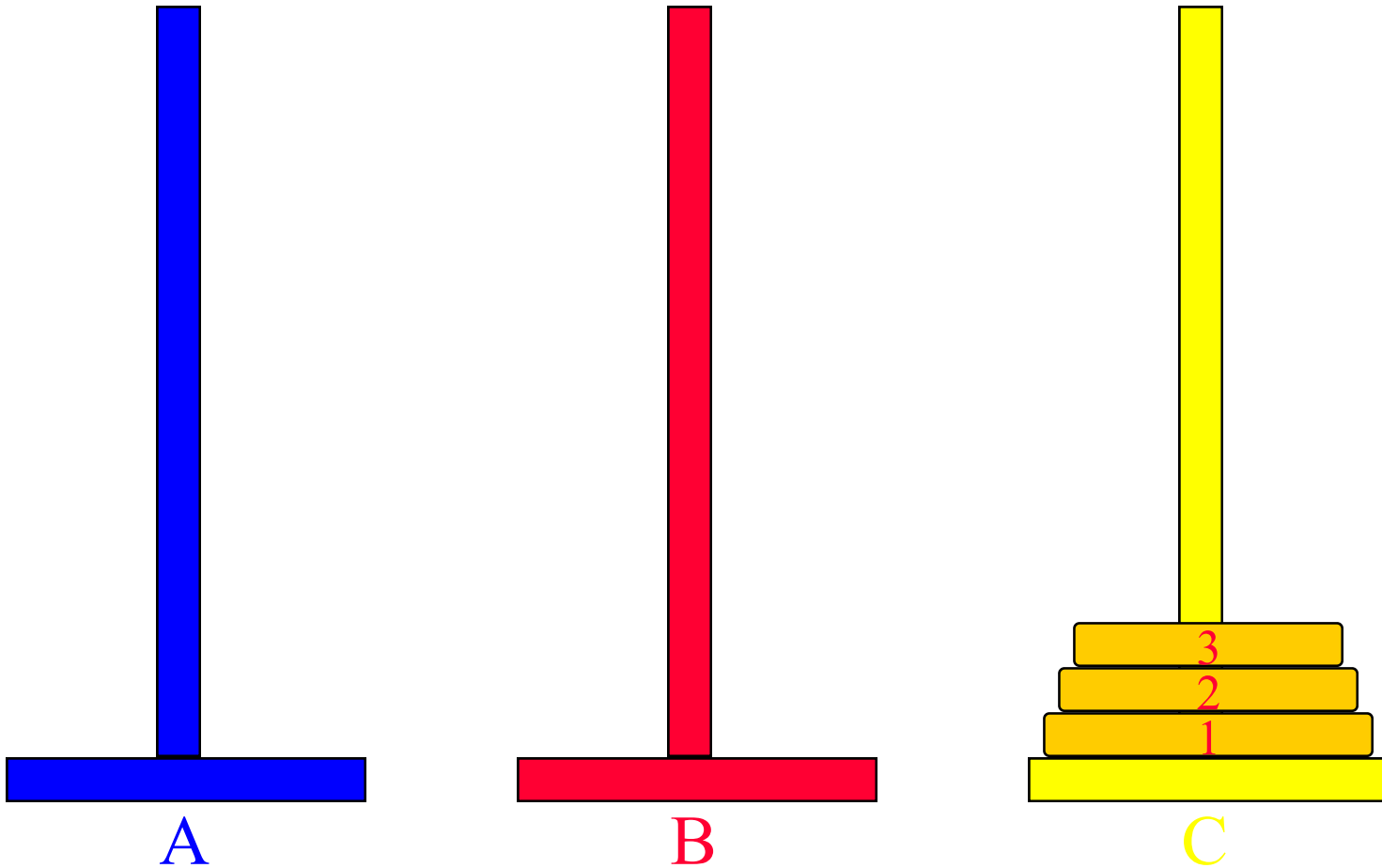
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



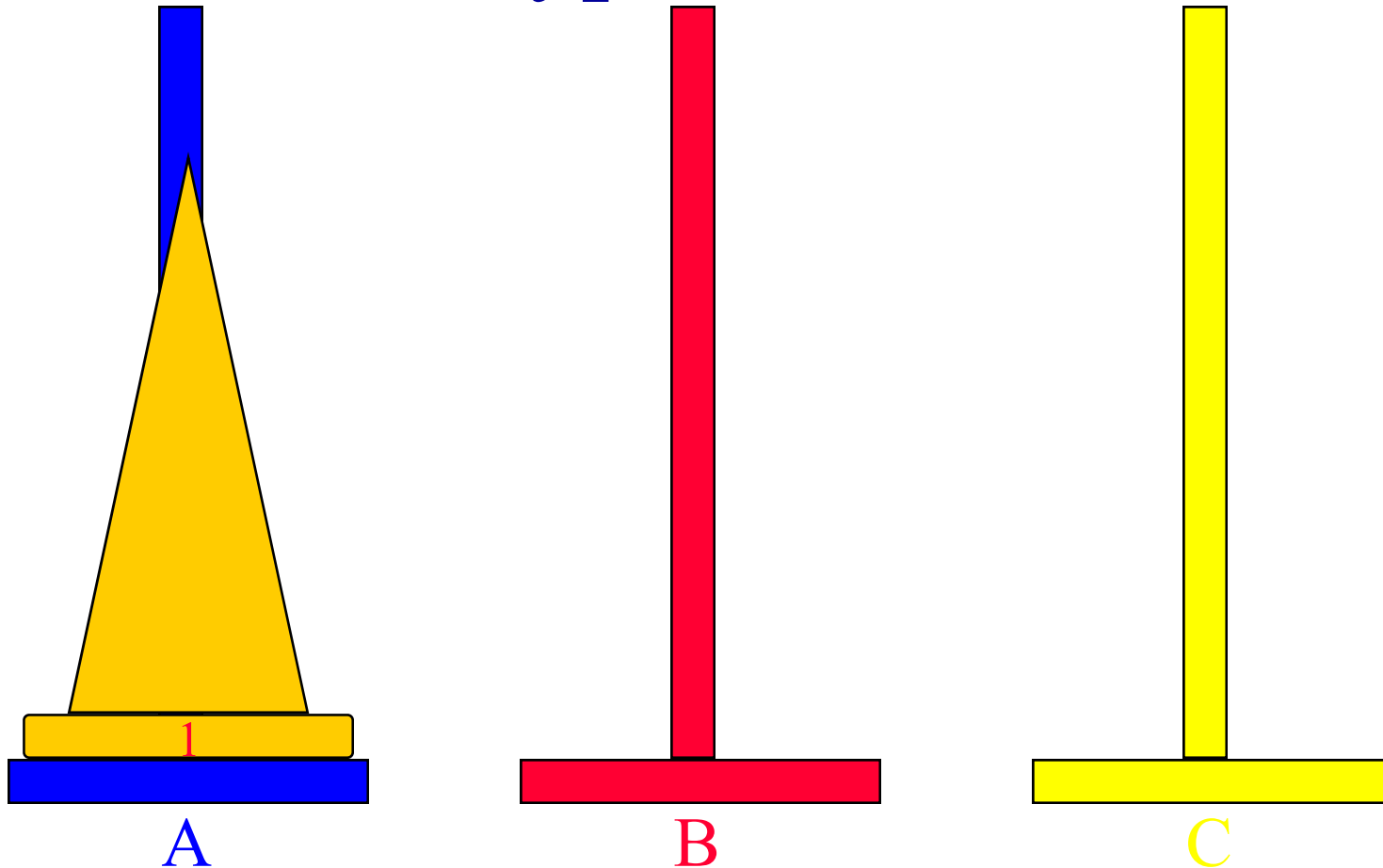
- 3 дисктэй Ханойн цамхаг

# Ханойн цамхаг



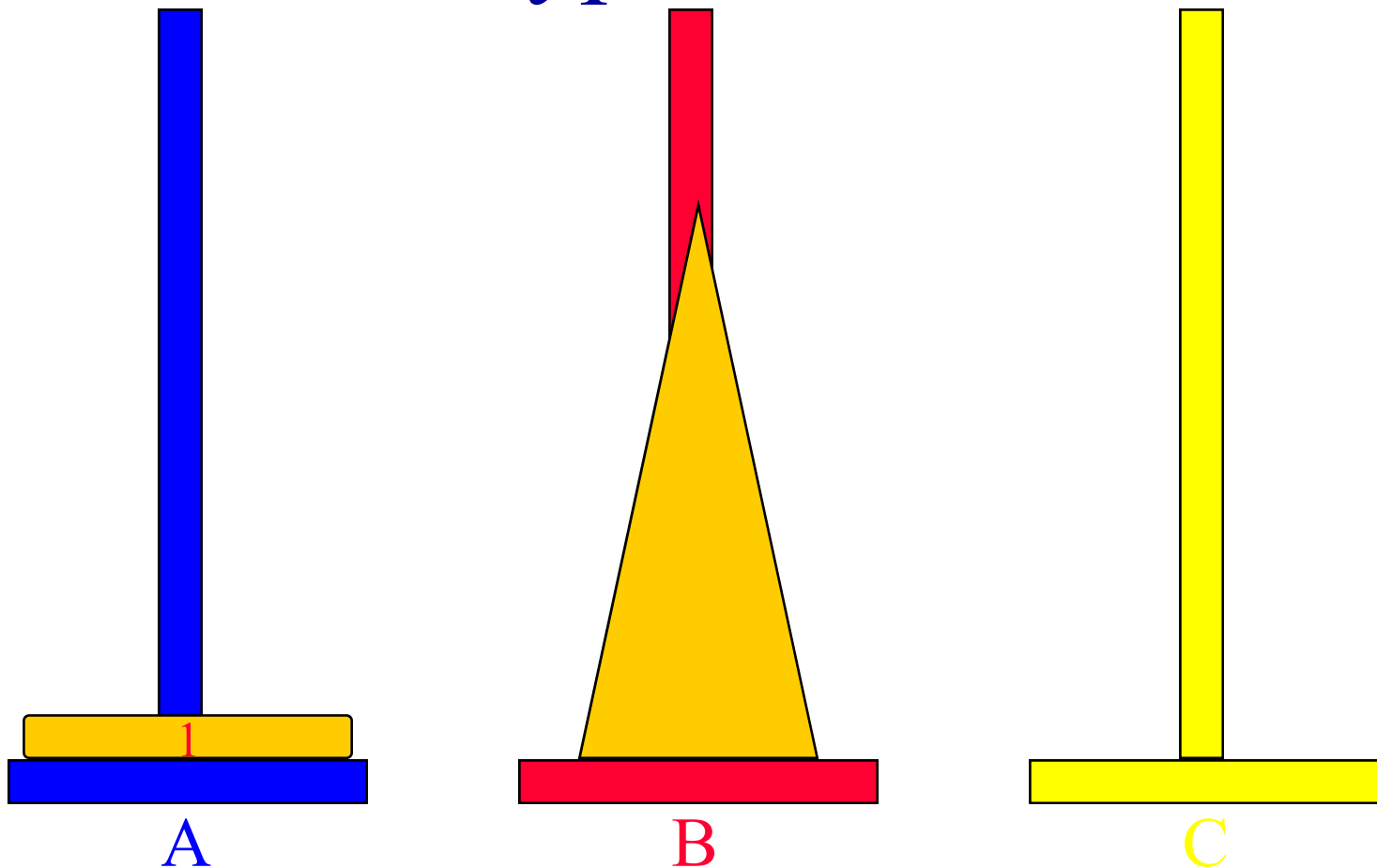
- 3 дисктэй Ханойн цамхаг
- 7 дискийг хөдөлгөлөө

# Рекурсив шийдэл



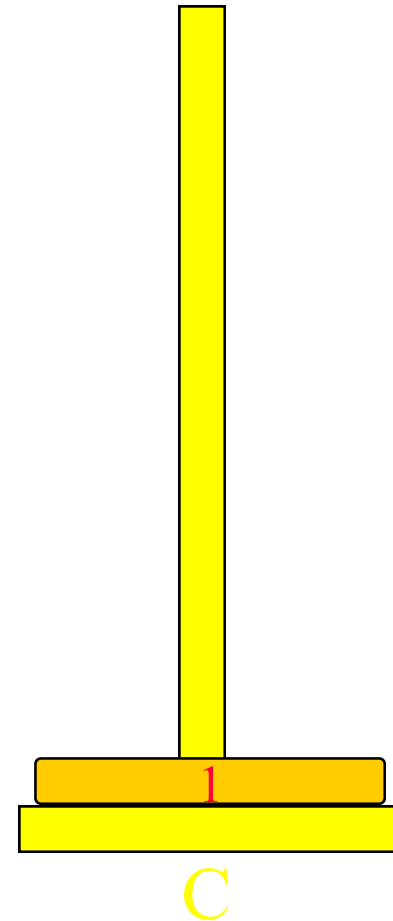
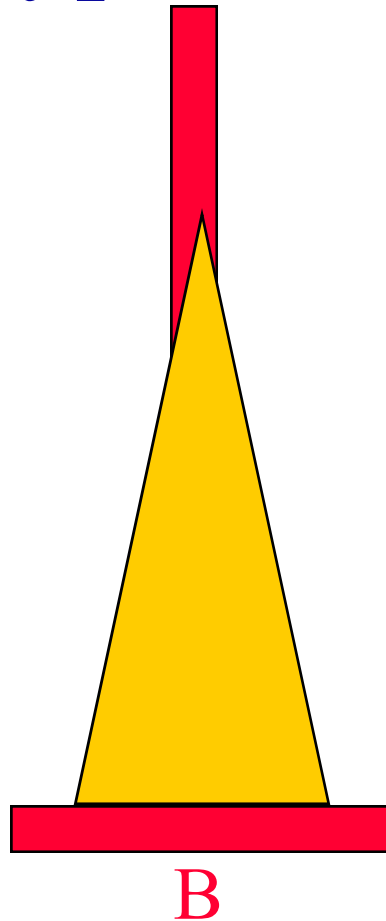
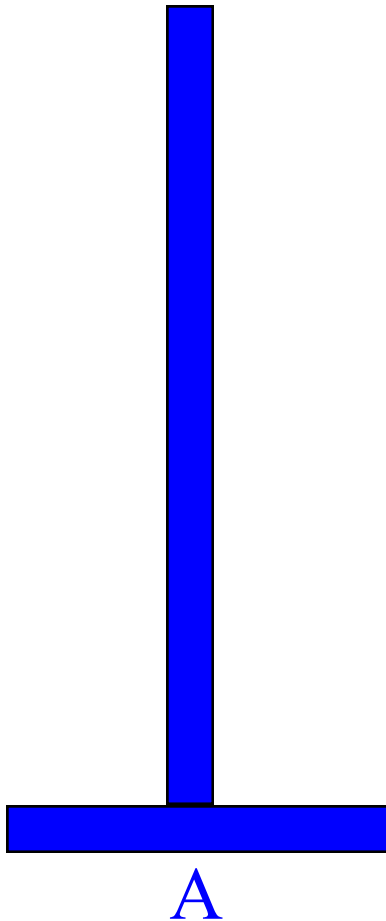
- $n > 0$  алтан дискийг A -аас C рүү B –г ашиглан шилжүүлнэ
- оройн  $n-1$  дискийг A -аас B рүү C –г ашиглан шилжүүлнэ

# Рекурсив шийдэл



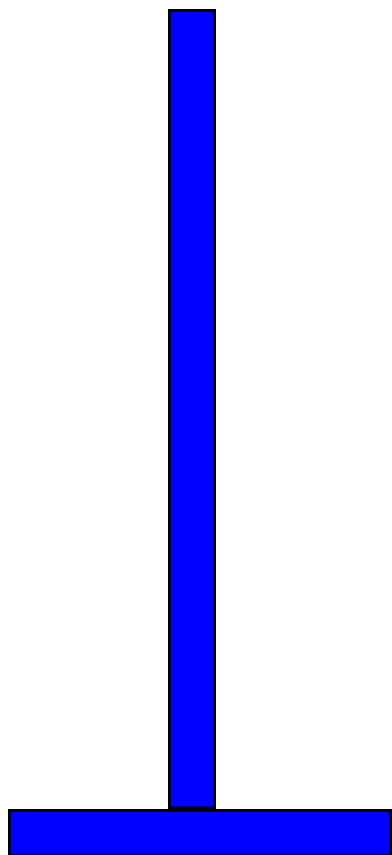
- оройн дискийг **A** -аас **C** рүү шилжүүлнэ

# Рекурсив шийдэл

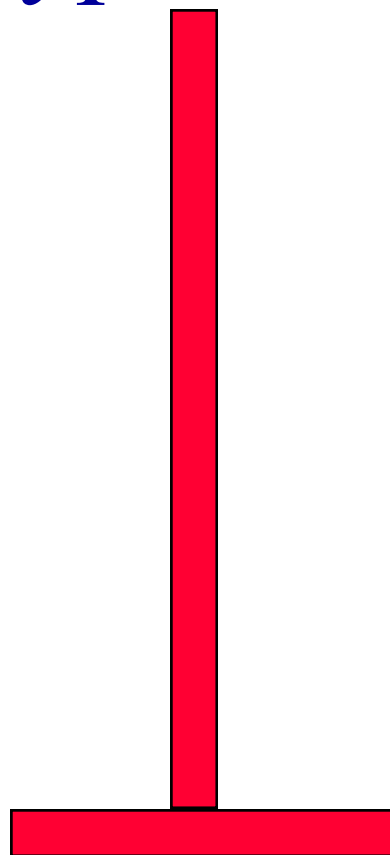


- оройн  $n-1$  дискийг  $B$  -ээс  $C$  рүү  $A$  –г ашиглан шилжүүлнэ

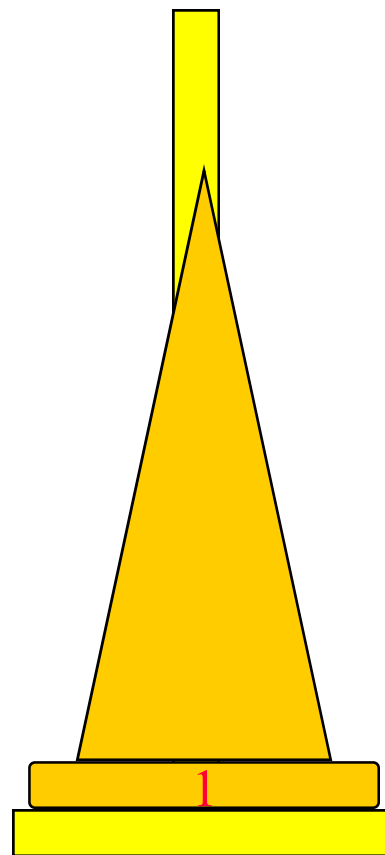
# Рекурсив шийдэл



A



B



C

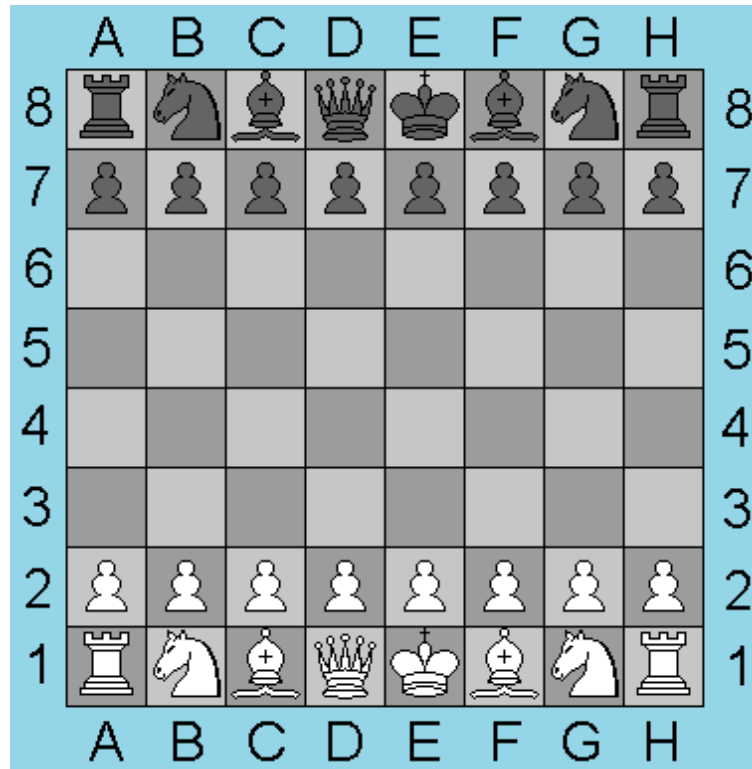
- $\text{moves}(n) = 0$ ,  $n = 0$  бол
- $\text{moves}(n) = 2 * \text{moves}(n-1) + 1 = 2^n - 1$ ,  $n > 0$  бол

# Ханойн цамхаг

- $\text{moves}(64) = 1.8 * 10^{19}$  (ойролцоогоор)
- $10^9$  шилжүүлэлт/сек хурдтай компьютер 570 орчим жил зарцуулна.
- 1 диск шилжүүлэлт/мин хийдэг лам  $3.4 * 10^{13}$  жил зарцуулна.

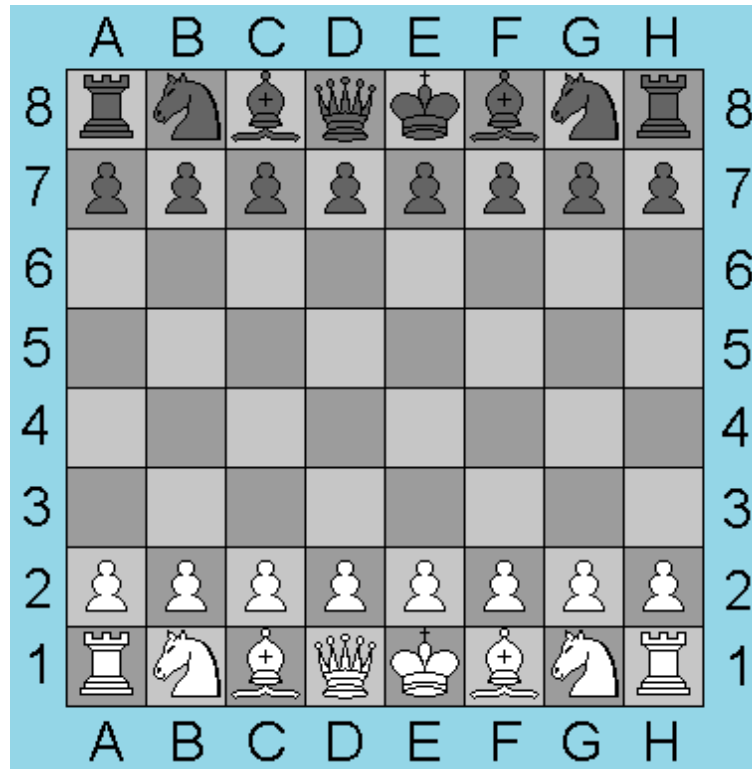


# Шатрын хөлөг



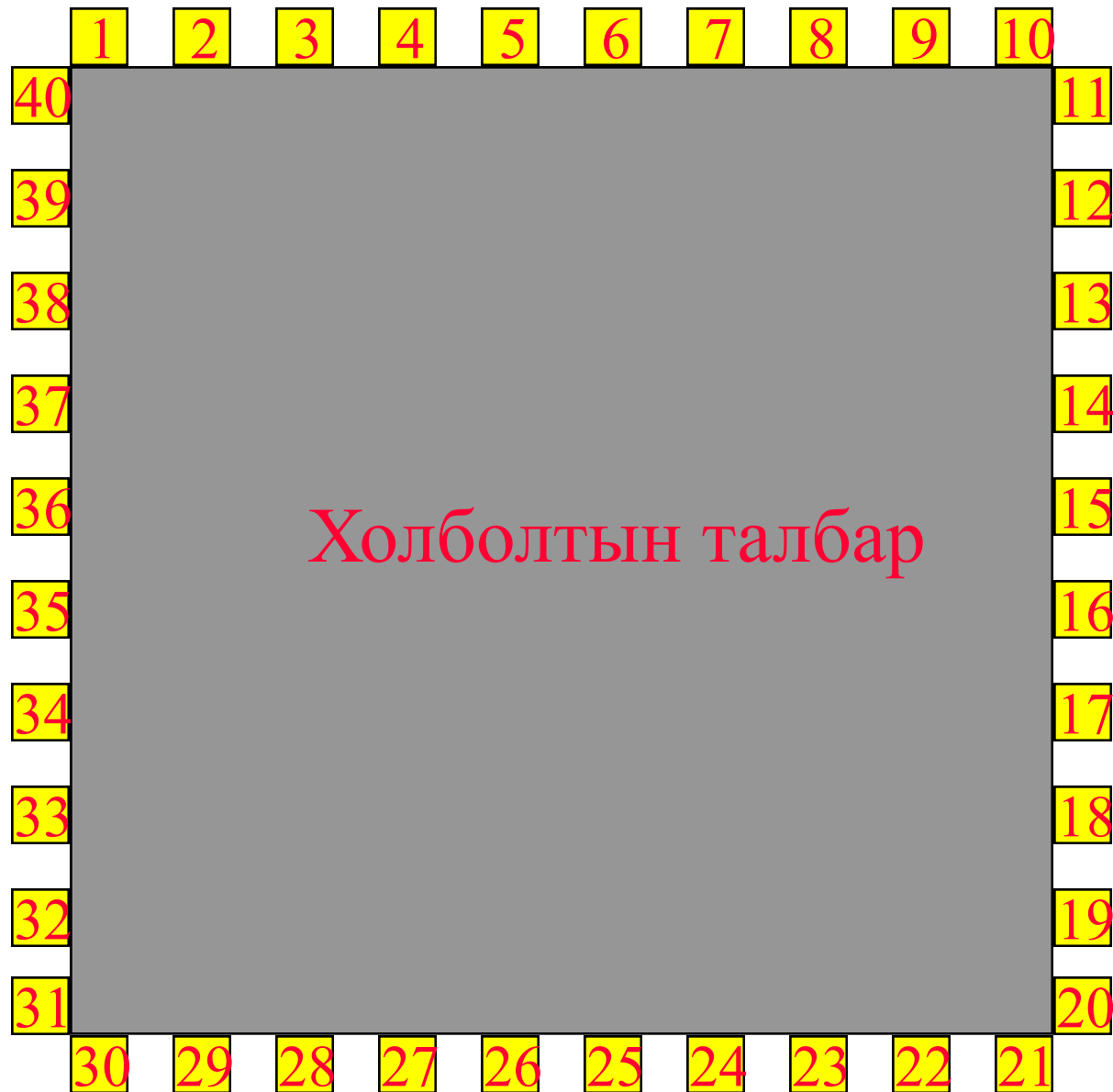
- Эхний нүдэнд 1 будаа, дараагийнхад 2, дараагийнхад 4, гэх мэт.
- Шаардлагатай талбай тэлхийг бүрхэнэ.

# Шатрын хөлөг

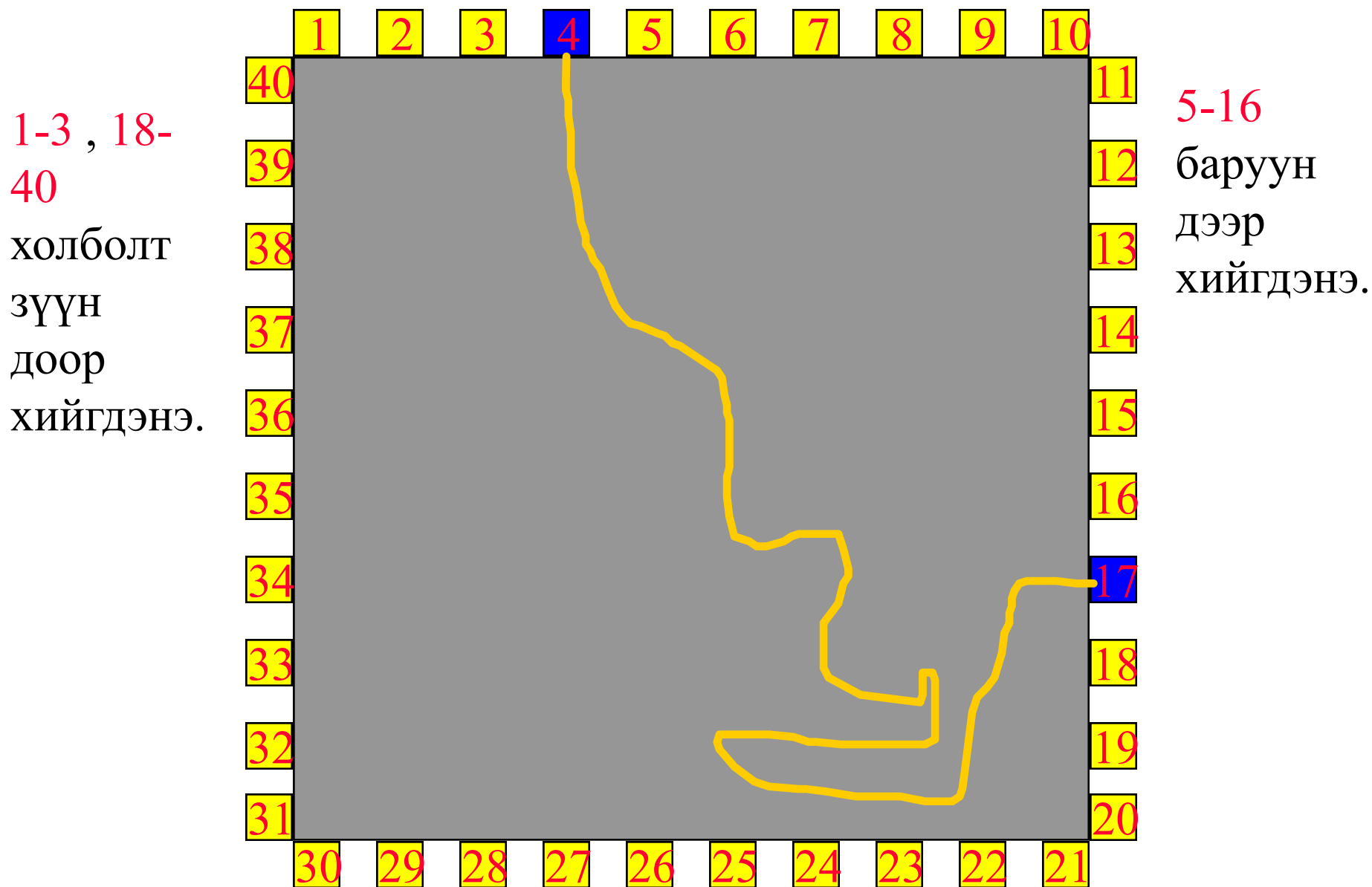


- Эхний нүдэнд 1 цент, дараагийнхад 2, дараагийнхад 4, гэх мэт.
- $\$3.6 * 10^{17}$  (холбооны төсөв  $\sim \$2 * 10^{12}$ ) .

# Холболтын хайрцаг



## 2 цэгийн холболт

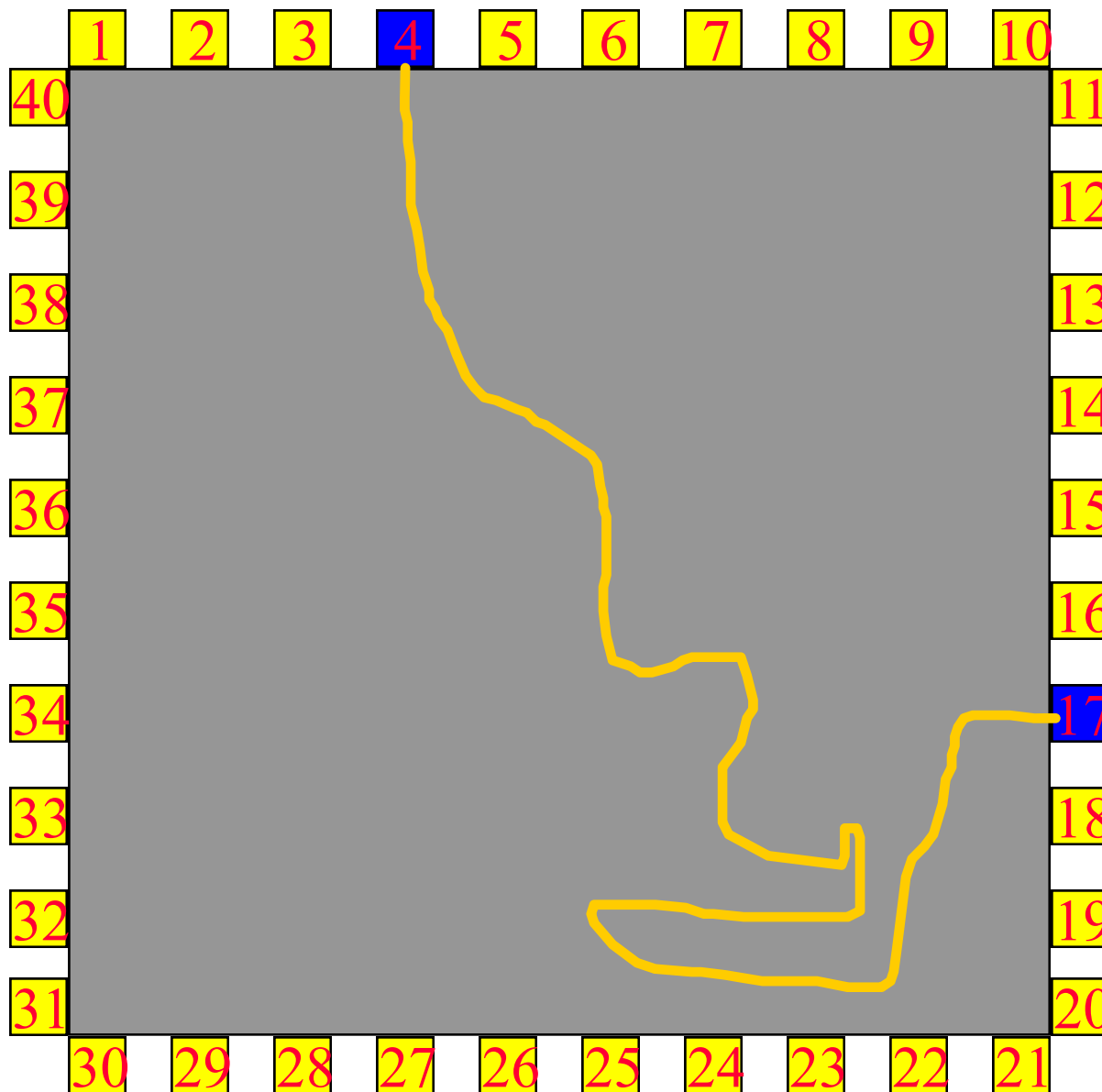


# 2 цэгийн холболт

$(u, v)$ ,  $u < v$   
бол 2-  
цэгийн  
холболт.

$u$  ЭХЛЭХ  
цэг.

$v$  ТӨГСӨХ  
цэг.

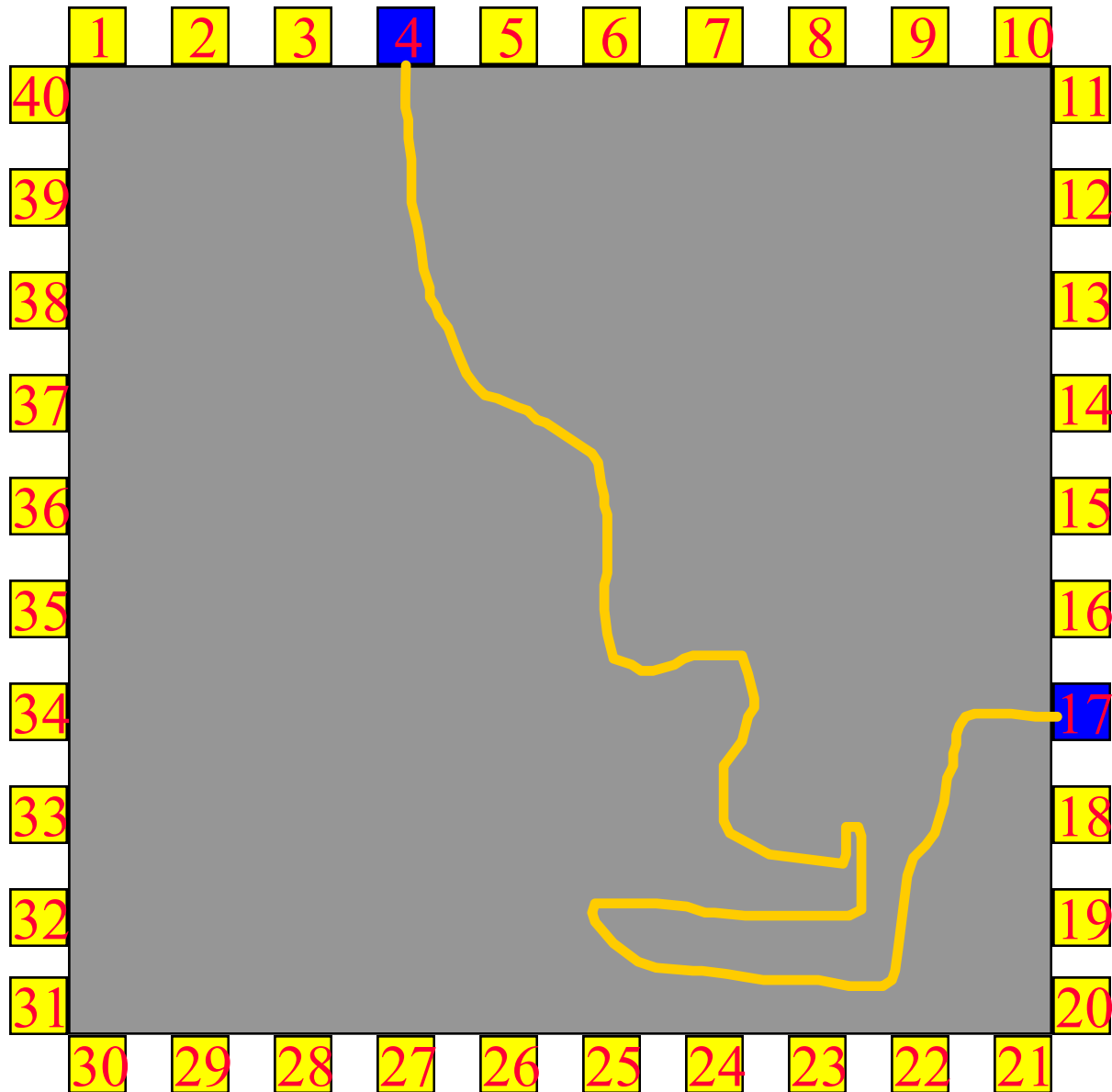


Цэгүүдий  
г цагын  
зүүний  
дагуу 1 -  
ээс эхлэн  
дугаарла  
в

## 2 цэгийн холболт

Эхлэх  
цэгийг  
=> стект  
хийнэ.

Төгсөх  
цэгт =>  
харгалзах  
ЭХЛЭХ ЦЭГ  
стекийн  
оройд  
байна.



# Аргыг Дуудах ба Буцах

```
public void a()
{ ...; b(); ...}
public void b()
{ ...; c(); ...}
public void c()
{ ...; d(); ...}
public void d()
{ ...; e(); ...}
public void e()
{ ...; c(); ...}
```

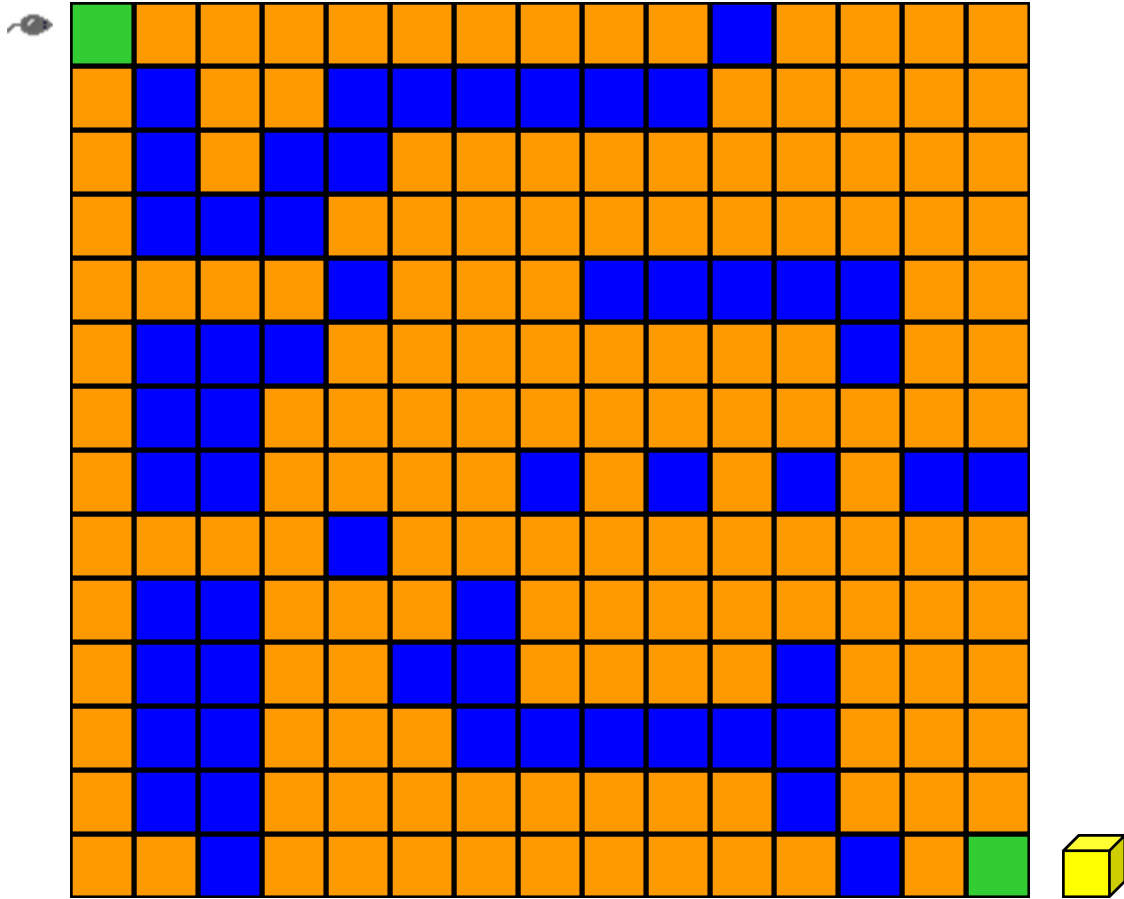
```
буцах хаяг d()
буцах хаяг c()
буцах хаяг e()
буцах хаяг d()
буцах хаяг c()
буцах хаяг b()
буцах хаяг a()
```

# Try-Throw-Catch

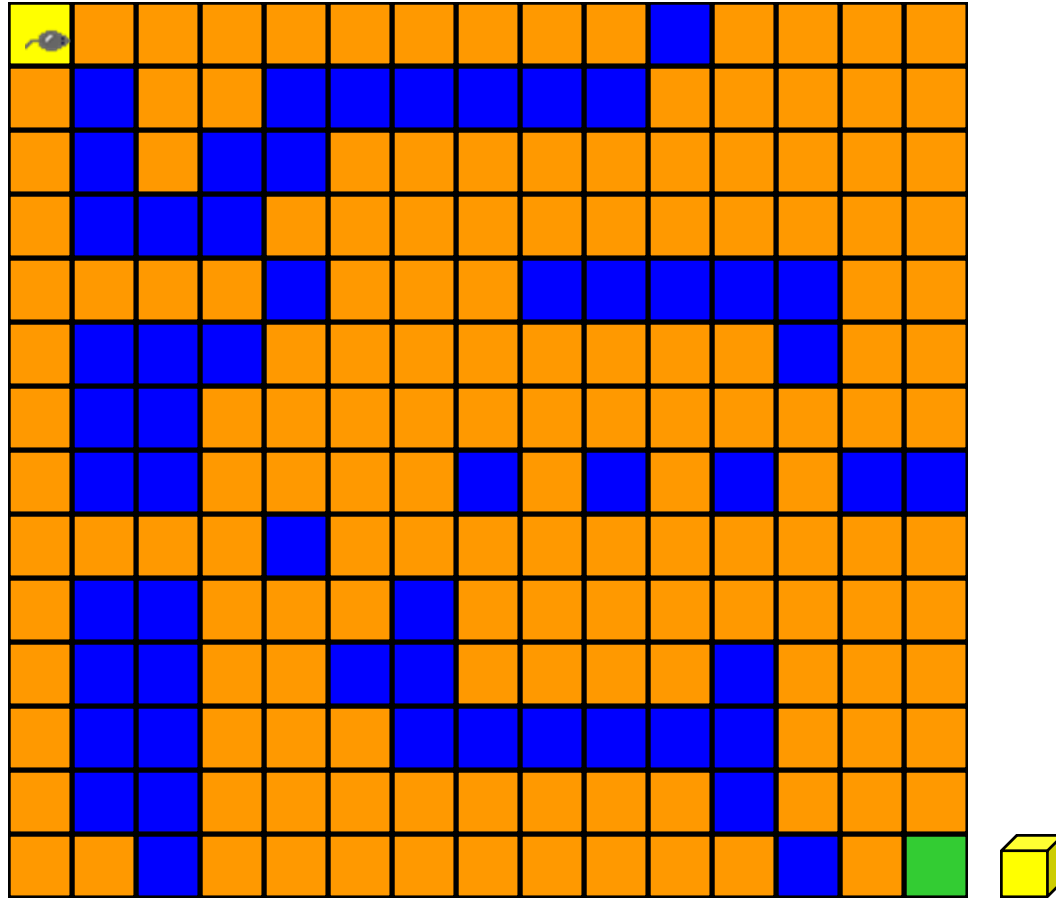
- `try` блокт ороход, энэ блокын хаягийг стект хийнэ.
- Онцгой тохиолдол унахад, стекийн оройд байгаа `try` блокын хаягийг авна(стек хоосон бол зогсоно).
- Гаргаж авсан `try` блокт харгалзах `catch` блок байхгүй бол, өмнөх алхам руу буцна.
- Гаргаж авсан `try` блокт харгалзах `catch` блок байгаа бол, тэр `catch` блок хэрэгжинэ.



# Төөрөлдсөн оготно

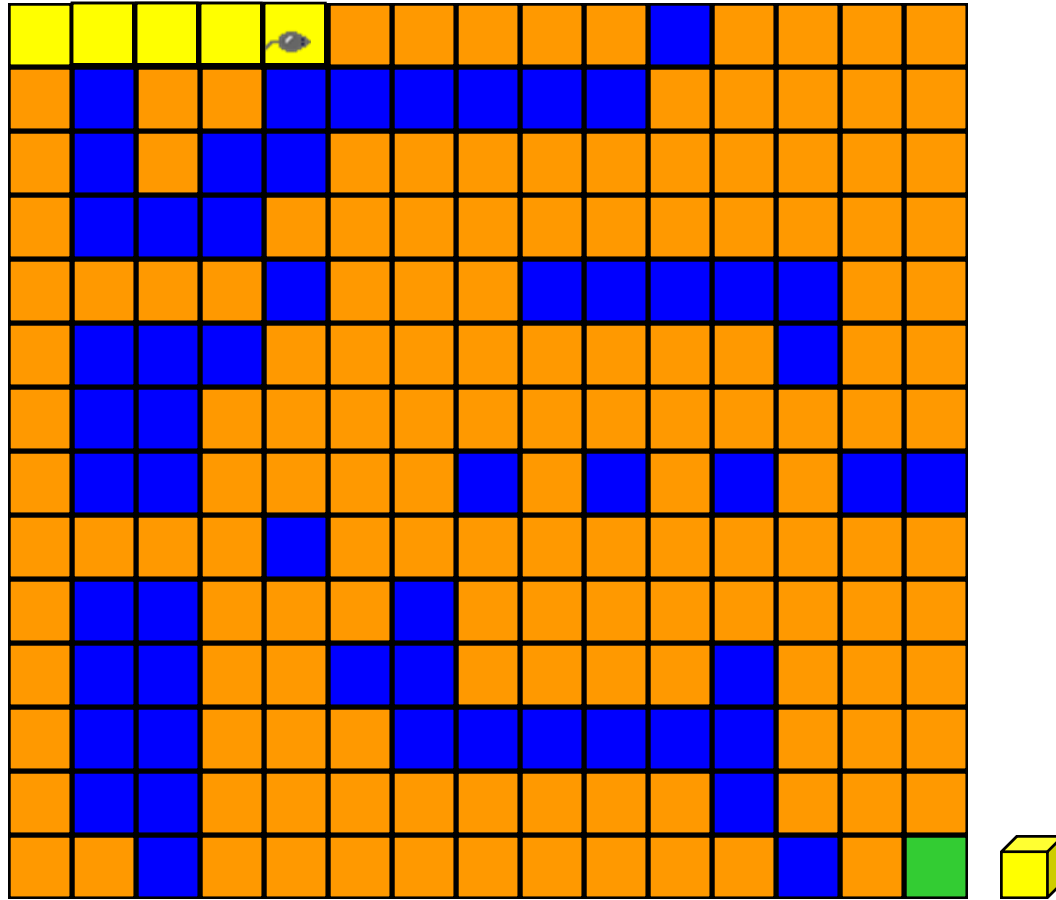


# Төөрөлдсөн оготно



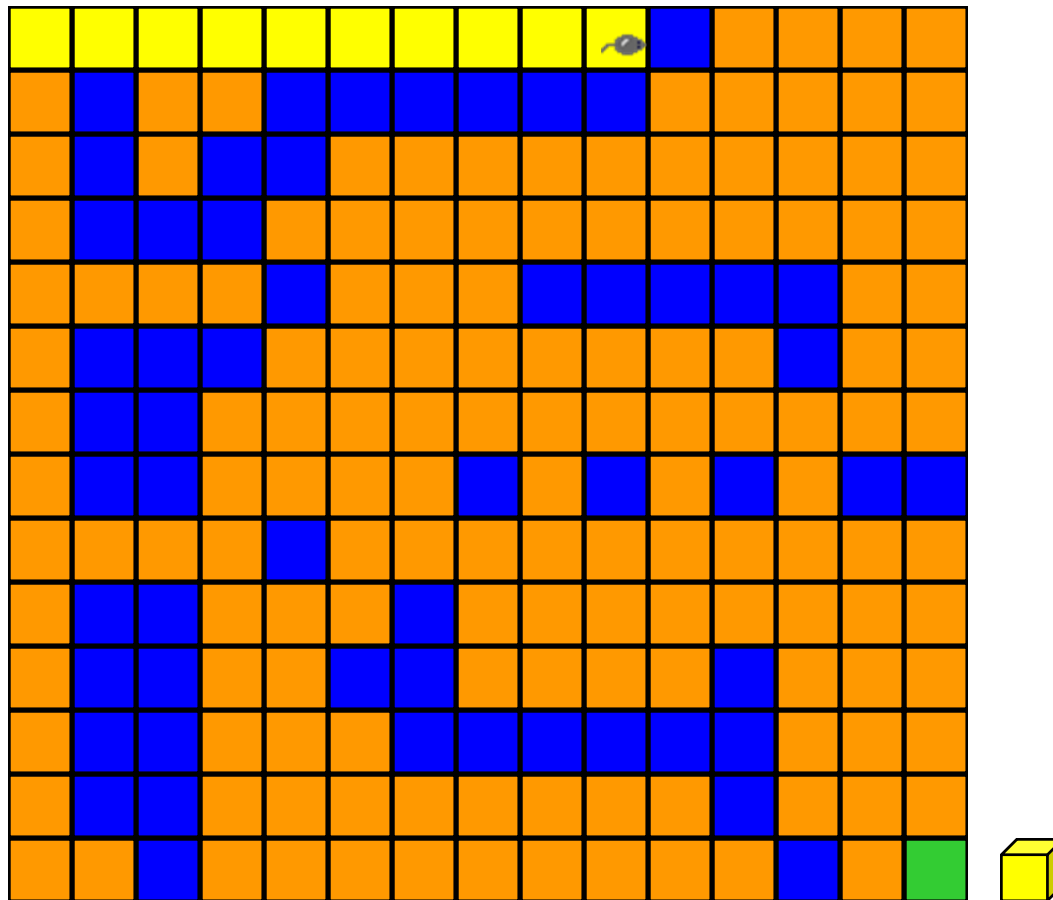
- Явах чиглэл: **right, down, left, up**
- Дахин орохгүйн тулд нүдийг хаана.

# Төөрөлдсөн оготно



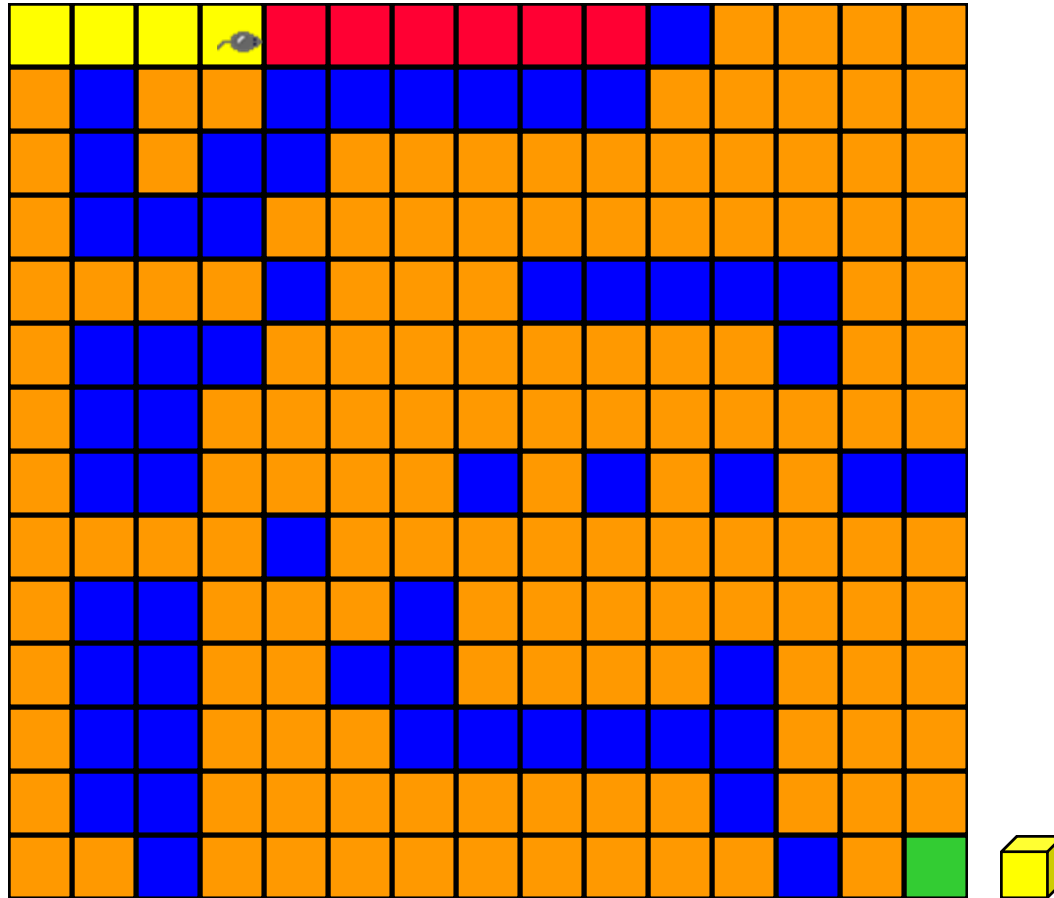
- Явах чиглэл: **right, down, left, up**
- Дахин орохгүйн тулд нүдийг хаана.

# Төөрөлдсөн оготно



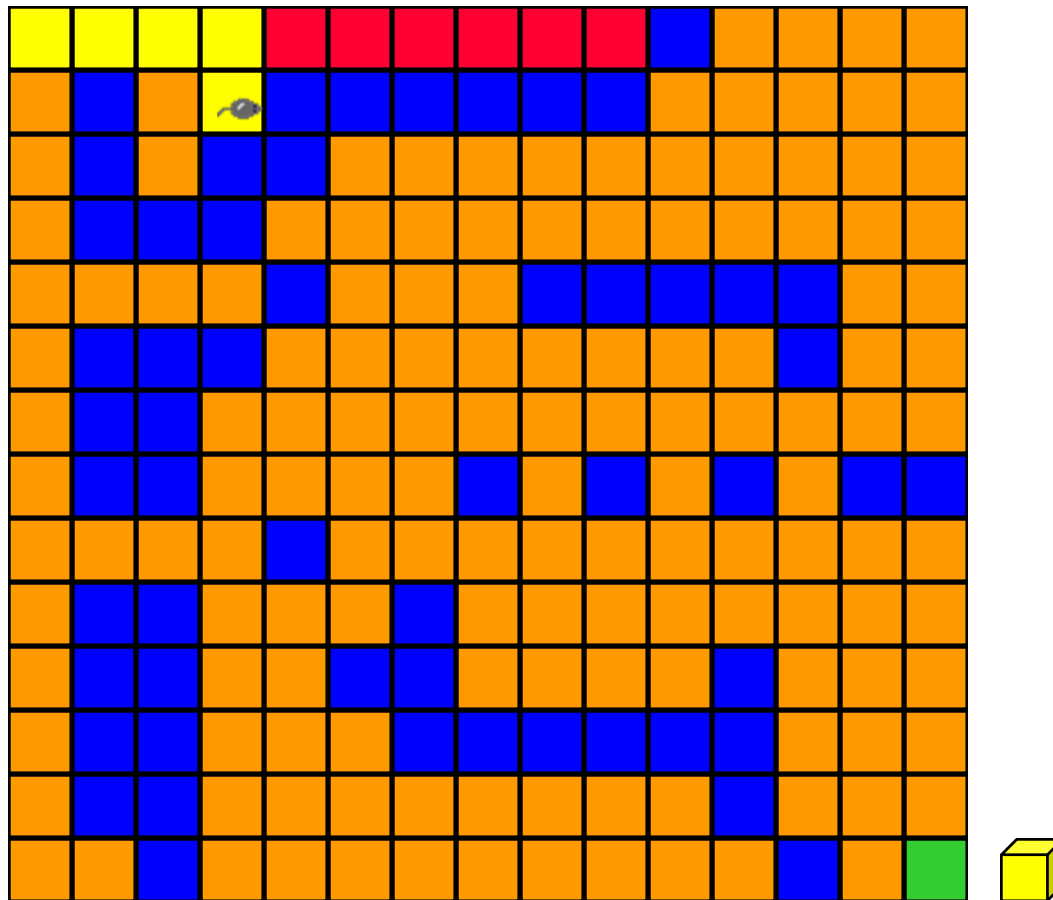
- Урагш явах боломжтой нүд хүртэл ухрах.

# Төөрөлдсөн оготно



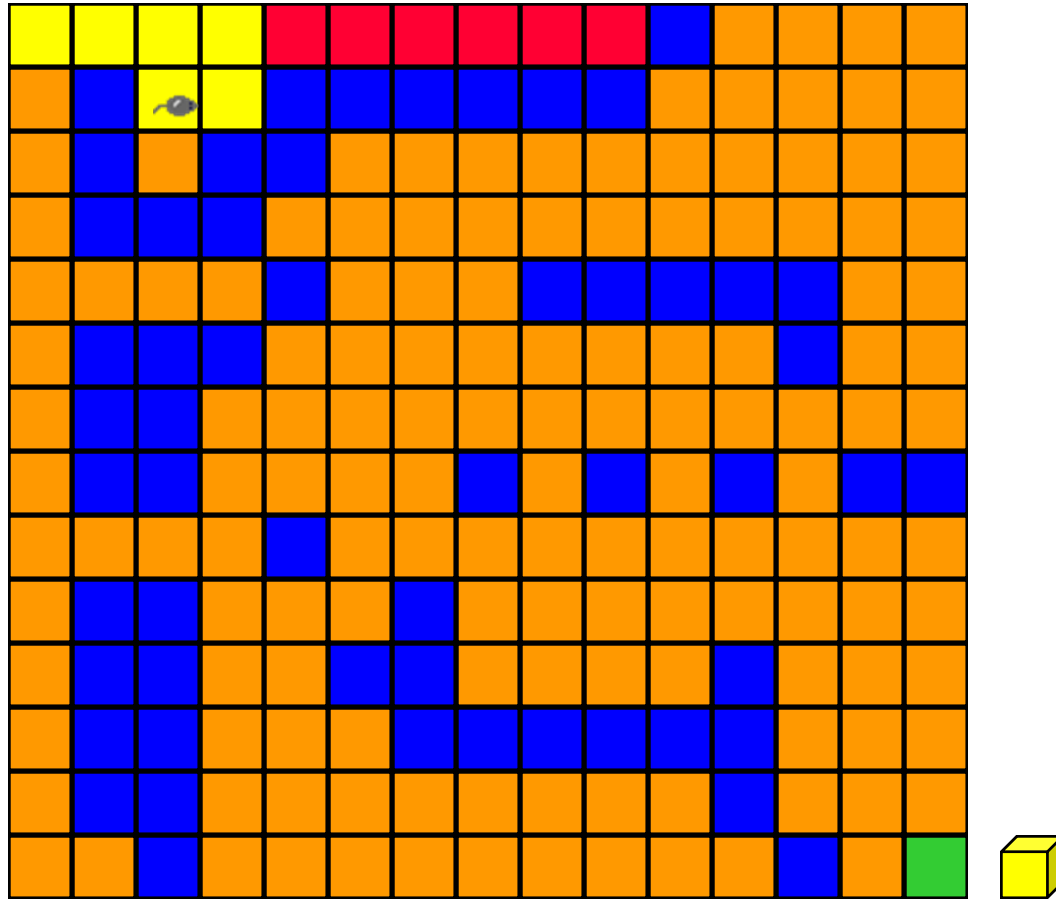
- Доош явах.

# Төөрөлдсөн оготно



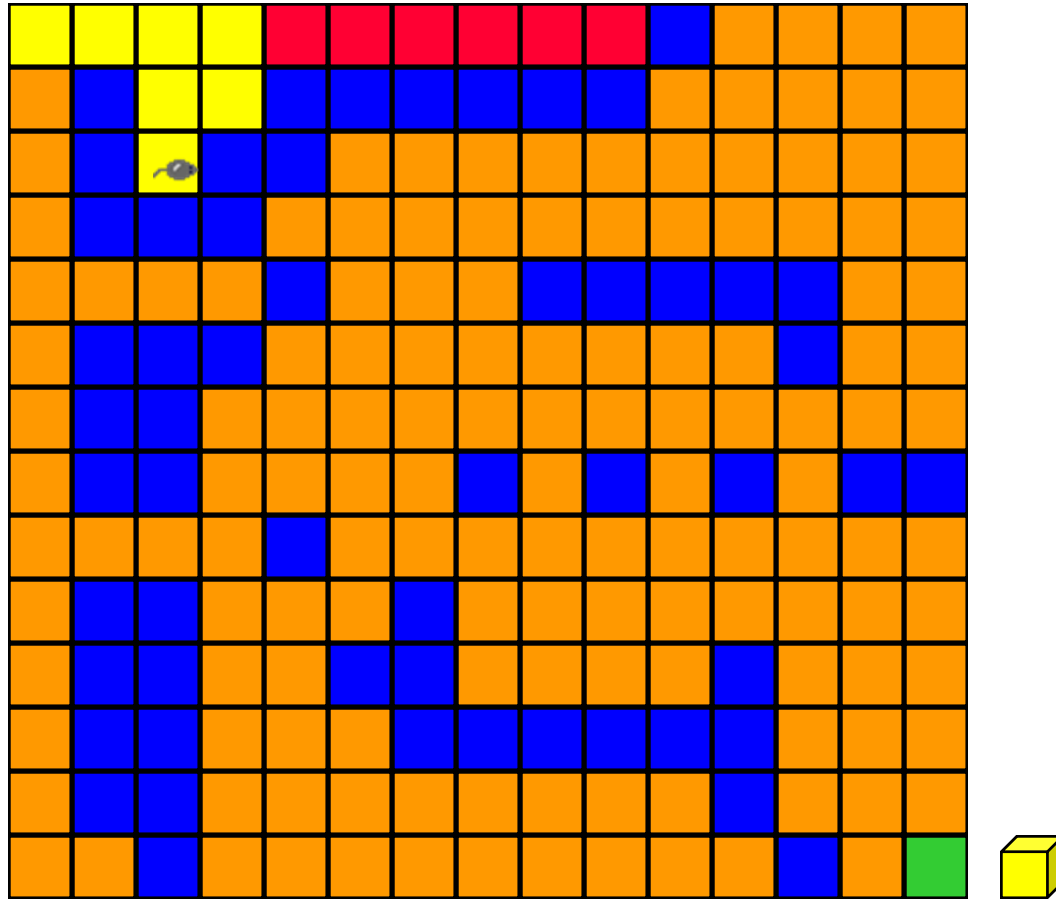
- Зүүн тийш явах.

# Төөрөлдсөн оготно



- Доош явах.

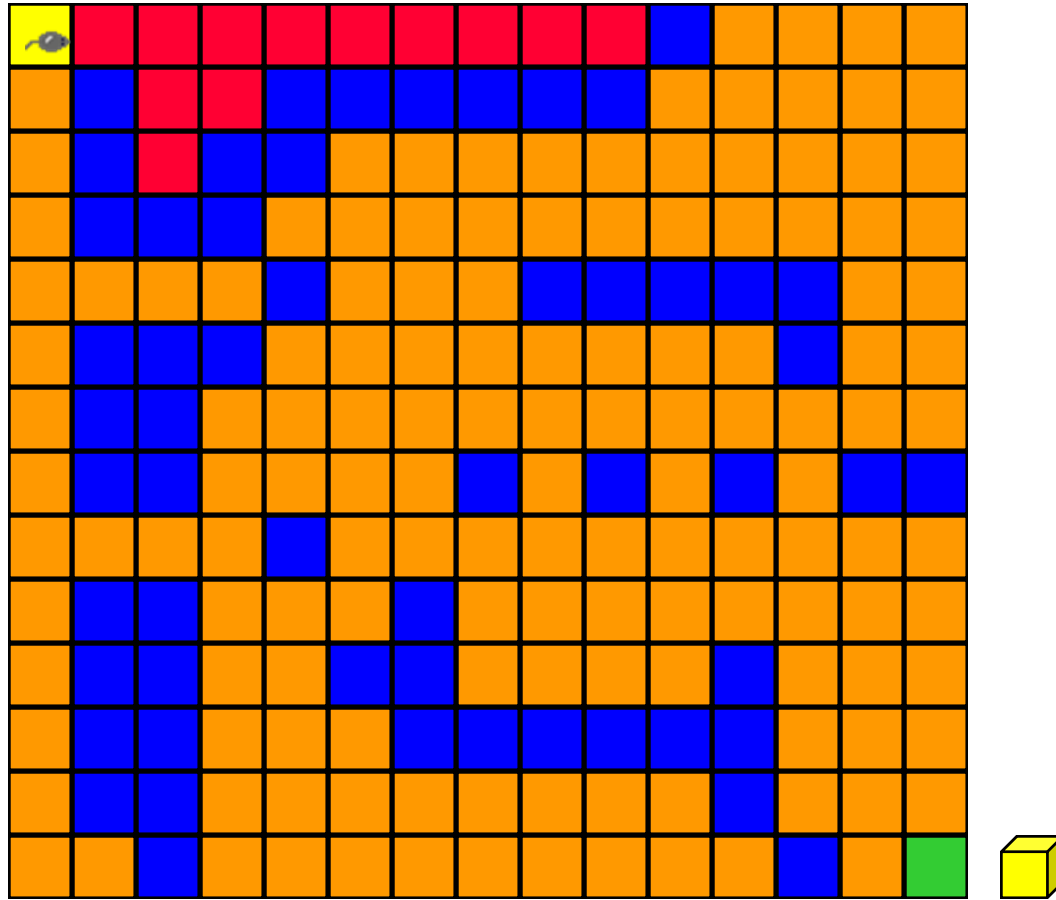
# Төөрөлдсөн оготно



- Урагш явах боломжтой нүд хүртэл ухрах.

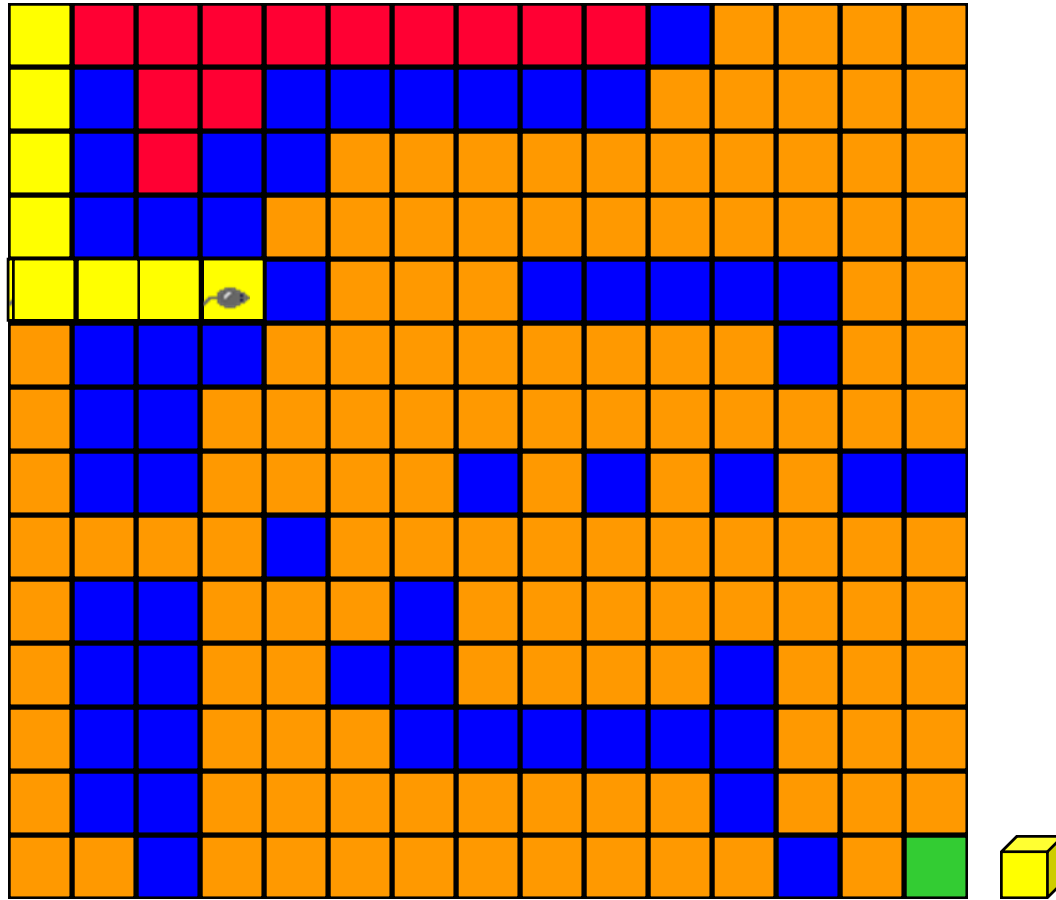


# Төөрөлдсөн оготно



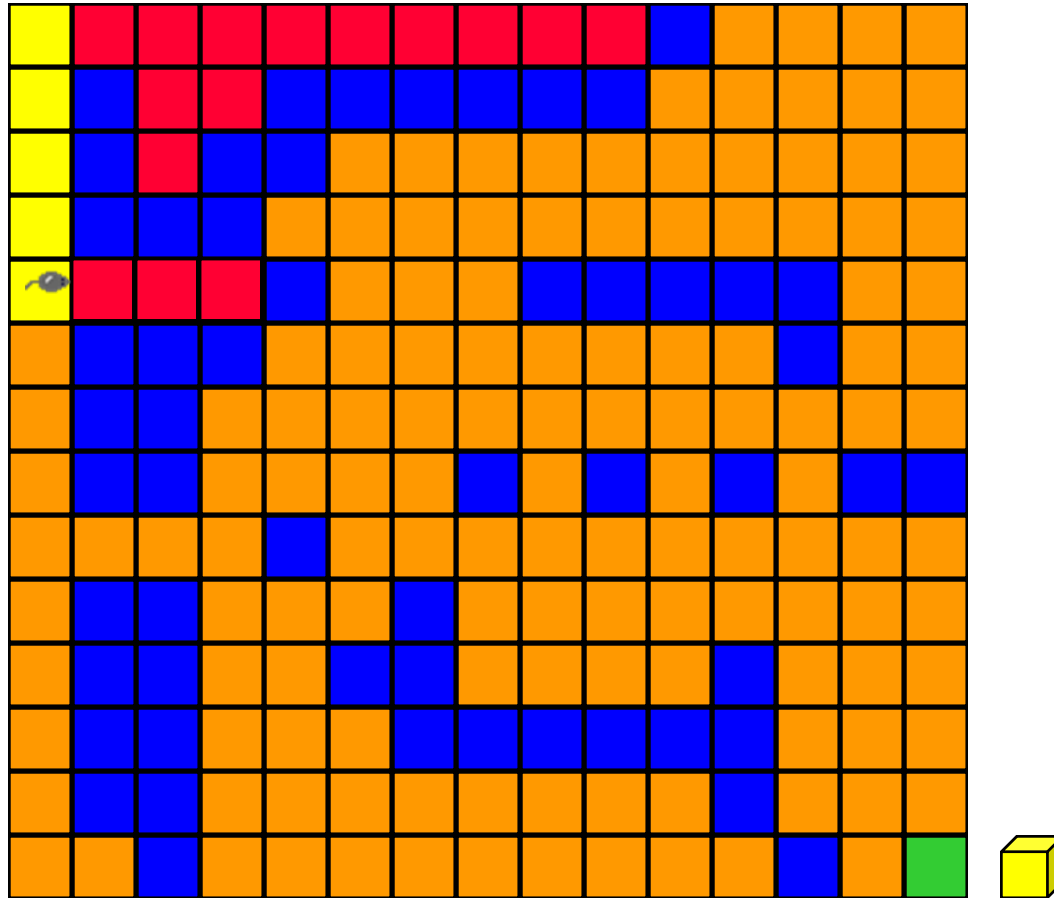
- Урагш явах боломжтой нүд хүртэл ухрах.
- Доош явах.

# Төөрөлдсөн оготно



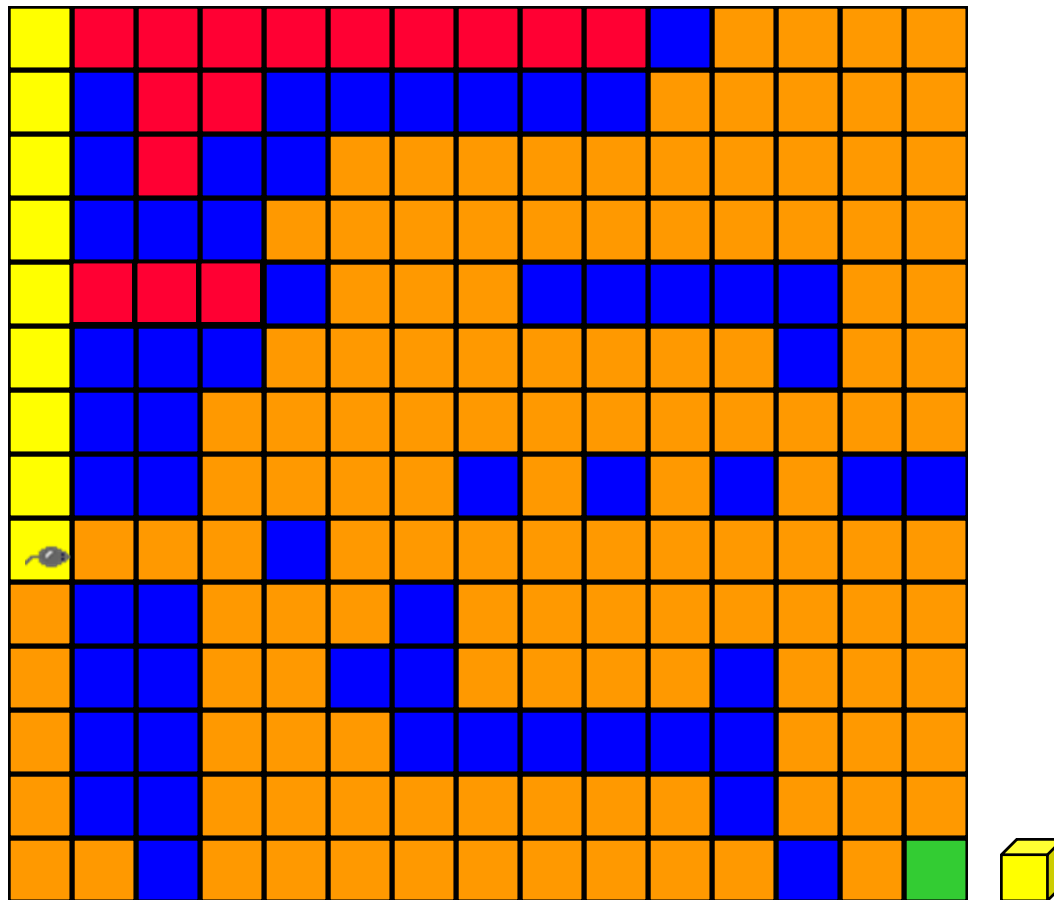
- Баруун тийш явах.
- Буцах.

# Төөрөлдсөн оготно



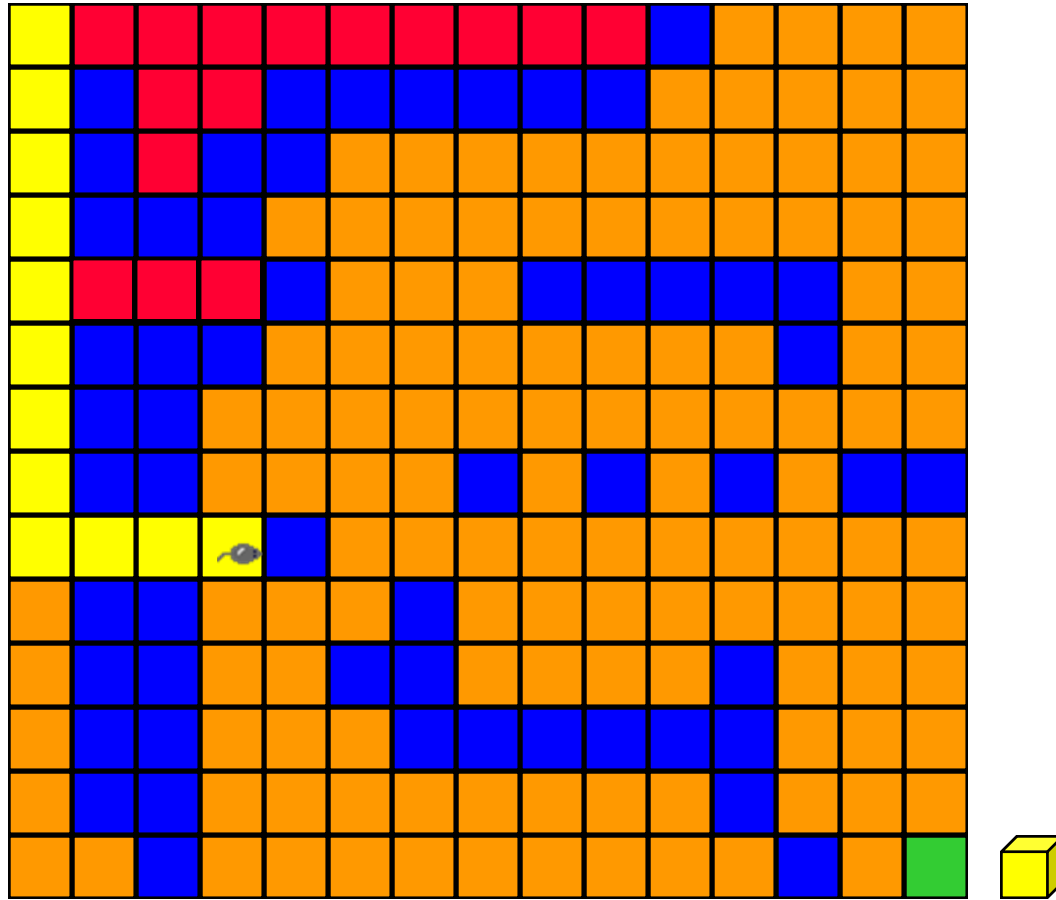
- Доош явах.

# Төөрөлдсөн оготно



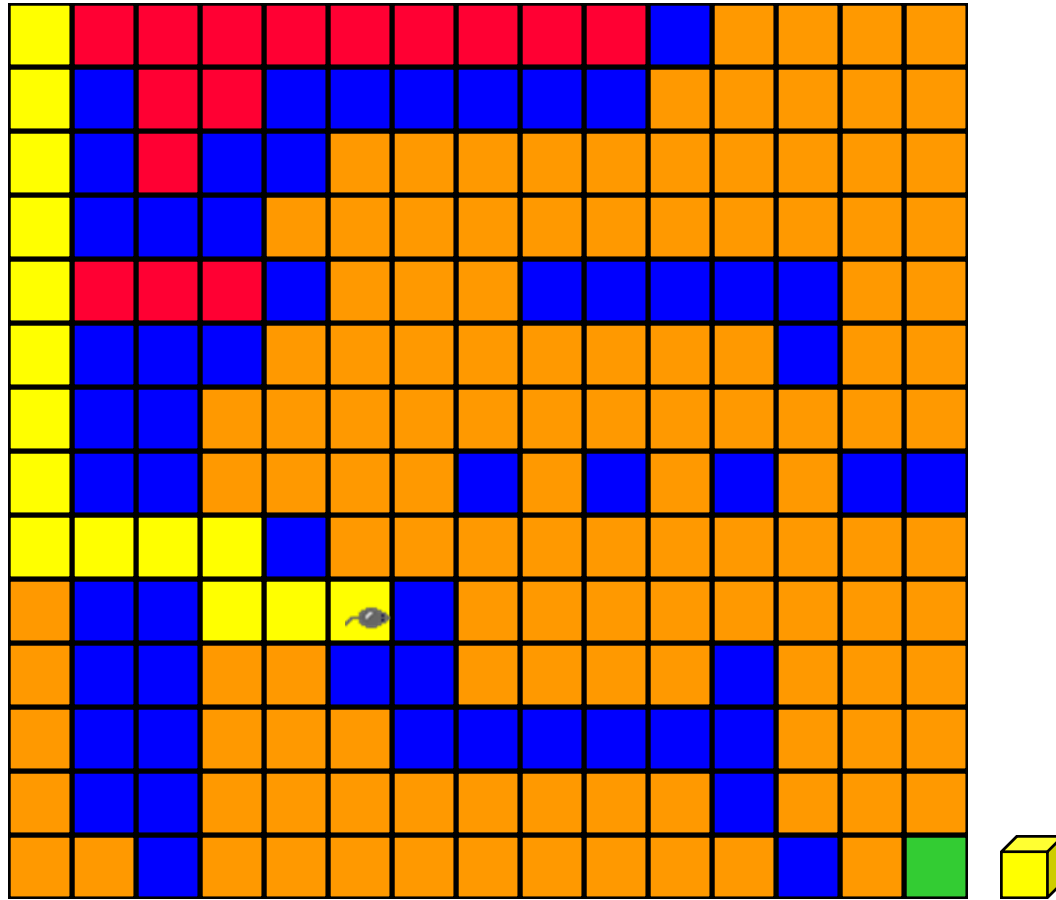
- Баруун тийш явах.

# Төөрөлдсөн оготно



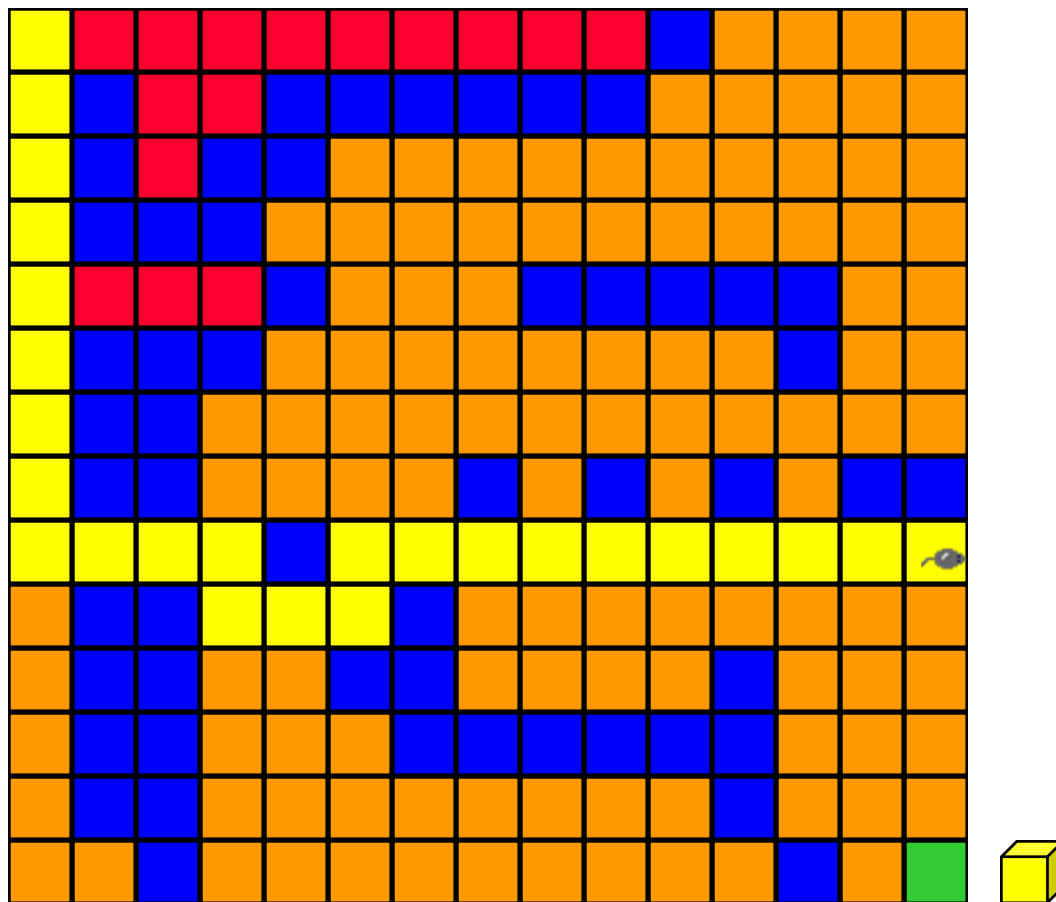
- Нэг доошлоод баруун тийш явах.

# Төөрөлдсөн хулгана



- Нэг дээшлээд баруун тийш явах.

# Төөрөлдсөн оготно



- Доош явж гарангаа бяслагийг идэх.
- Оготны орсон нүднээс тухайн байршил хүртэлх зам стектэй адилхан ажиллана.

# Стек

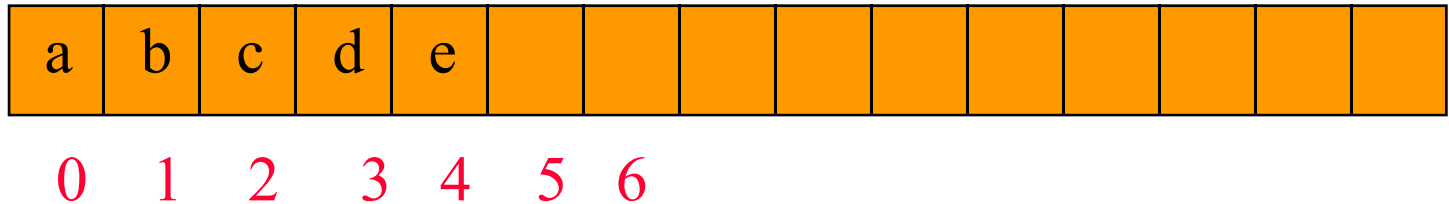
```
public interface Stack
{
    public boolean empty();
    public Object peek();
    public void push(Object theObject);
    public Object pop();
}
```



# Linear List классаас уламжлах

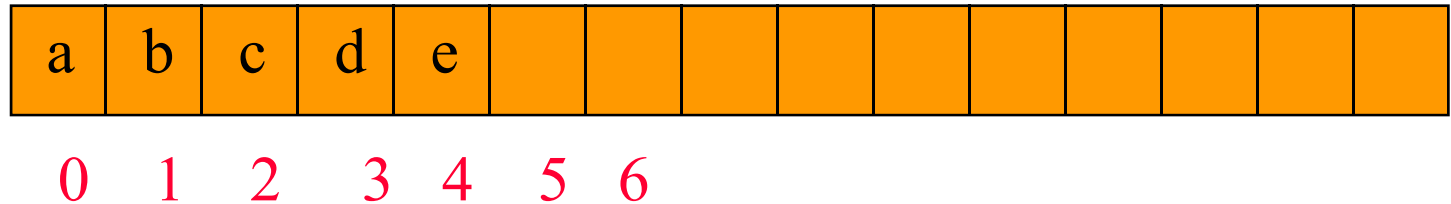
- *ArrayLinearList*
- *Chain*

# ArrayLinearList –ээс уламжлах



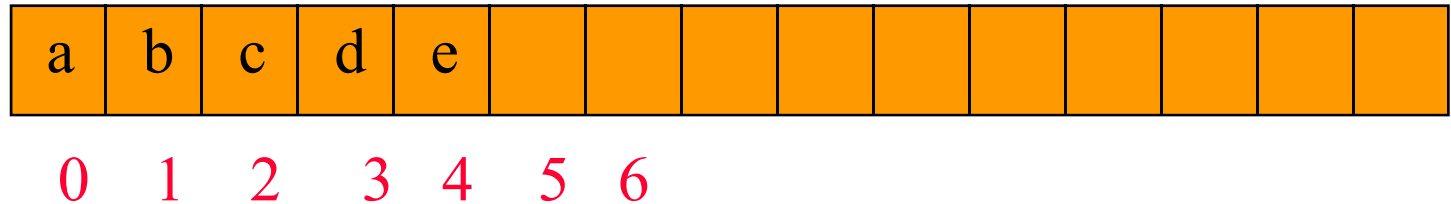
- стекын орой нь шугаман жагсаалтын зүүн, баруун төгсгөлийн нэг байна
- `empty()` => `isEmpty()`
- `peek()` => `get(0)` эсхүл `get(size() - 1)`

# ArrayList –ээс уламжлах



- шугаман жагсаалтын зүүн төгсгөл нь орой бол
  - `push(theObject)`  $\Rightarrow$  `add(0, theObject)`
  - `pop()`  $\Rightarrow$  `remove(0)`

# ArrayList –ээс уламжлах

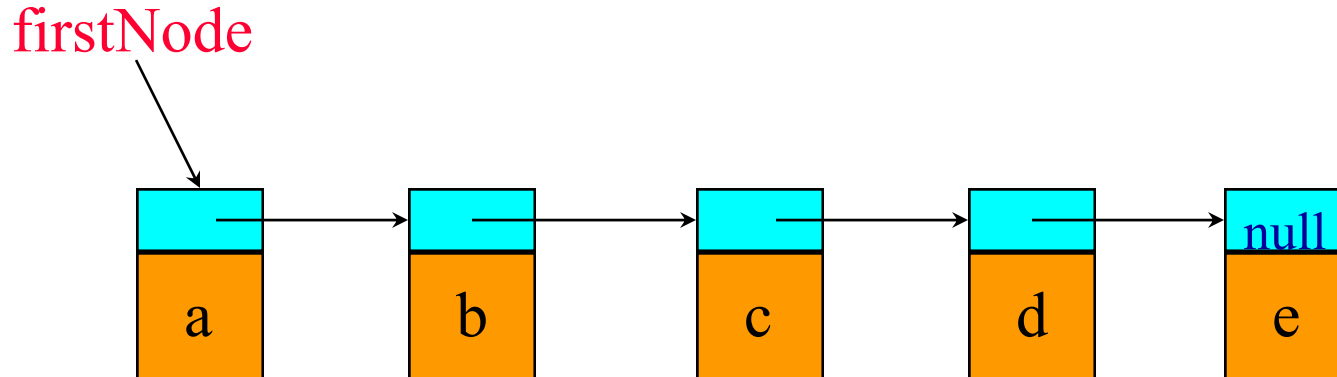


— шугаман жагсаалтын баруун төгсгөл орой бол

- `push(theObject) => add(size(), theObject)`
- `pop() => remove(size()-1)`

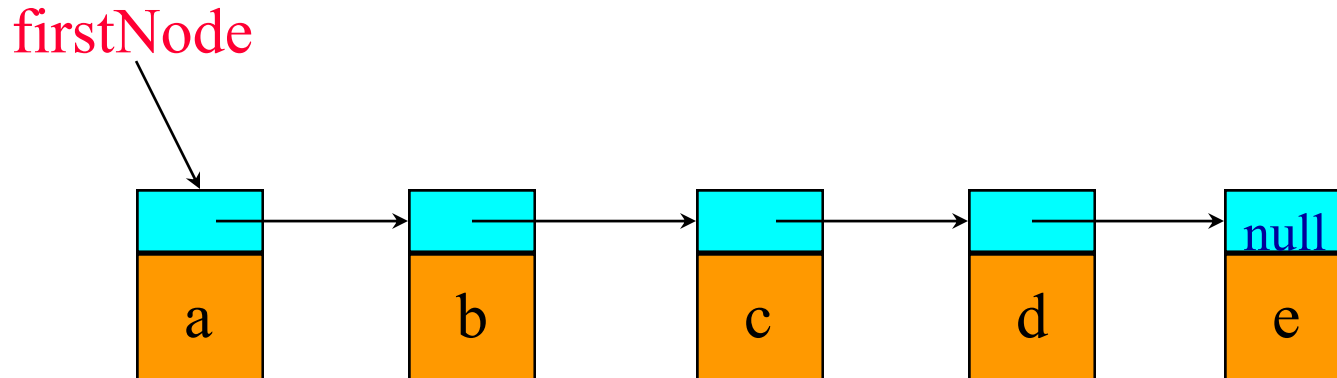
— жагсаалтын баруун төгсгөлийг стекын орой болгож ашиглая

# Chain –ээс уламжлах



- Шугаман жагсаалтын зүүн, баруун төгсгөлийн нэг стекын орой
- `empty() => isEmpty()`

# Chain –ээс уламжлах

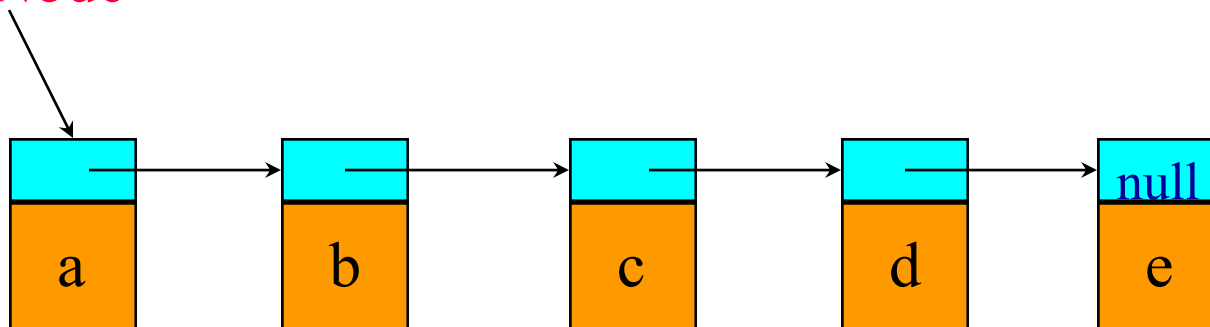


– шугаман жагсаалтын зүүн төгсгөл орой бол

- `peek() => get(0)`
- `push(theObject) => add(0, theObject)`
- `pop() => remove(0)`

# Chain –ээс уламжлах

firstNode



– шугаман жагсаалтын баруун төгсгөл орой бол

- `peek() => get(size() - 1)`
- `push(theObject) => add(size(), theObject)`
- `pop() => remove(size()-1)`
- **жагсаалтын зүүн төгсгөлийг стекын орой болгож ашиглая**

# ArrayLinearList –ээс уламжлах

```
package dataStructures;
```

```
import java.util.*; // онцгой тохиолдолд
```

```
public class DerivedArrayStack  
    extends ArrayLinearList  
    implements Stack
```

```
{
```

```
    // байгуулагчид
```

```
    // Stack интерфэйсийн аргууд
```

```
}
```





# Байгуулагчид



/\*\* тодорхой багтаамжтай стек

байгуулах \*/

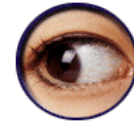
```
public DerivedArrayStack(int initialCapacity)  
{super(initialCapacity);}
```

/\*\* 10 –ын багтаамжтай стек байгуулах\*/

```
public DerivedArrayStack()  
{this(10);}
```



# empty() ба peek()

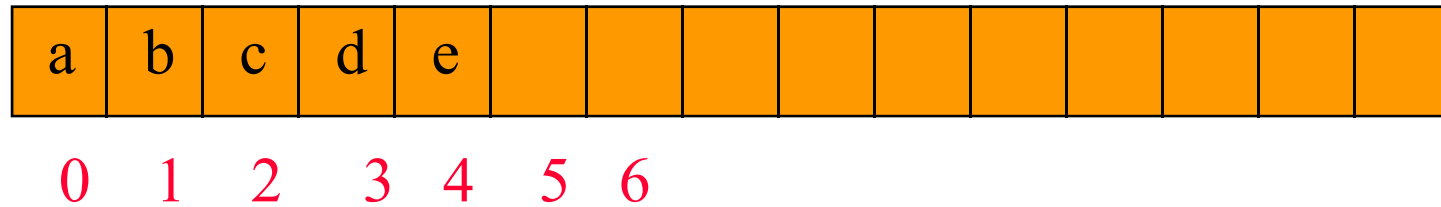


0 1 2 3 4 5 6

```
public boolean empty()  
{return isEmpty();}
```

```
public Object peek()  
{  
    if (empty())  
        throw new EmptyStackException();  
    return get(size() - 1)  
}
```

# push(theObject) ба pop()



```
public void push(Object theElement)
{add(size(), theElement);}
```

```
public Object pop()
{
    if (empty())
        throw new EmptyStackException();
    return remove(size() - 1);
}
```

# Дүгнэлт

- **ArrayLinearList** –ээс уламжлахын давуу тал
  - Уламжлагдсан классын код маш энгийн, хөгжүүлэхэд амар.
  - Кодыг бага зэрэг зүгшрүүлэх.
  - Стекыг холбоосоор хэрэгжүүлэх кодыг амархан гарган авч болно.
    - **extends ArrayLinearList** -г **extends Chain** –ээр солино
    - Бүтээмжийг дээшлүүлэх үүднээс жагсаалтын зүүн төгсгөлийг орой болгохоор өөрчлөлт оруулах хэрэгтэй.

# Дутагдал

- **ArrayList** -ийн бүх public аргуудыг стект хэрэглэж болно.
  - **get(0)** ... ёроолын элементийг авах
  - **remove(5)**
  - **add(3, x)**
  - Тэгэхээр жинхэнэ стекийн хэрэгжилт биш болж байна.
  - Хэрэглэгдэхгүй аргуудыг давхар ачаалах ёстой.

```
public Object get(int theIndex)
```

```
{throw new UnsupportedOperationException();}
```

Өмнө ашигласан **get(i)** -г **super.get(i)** болгож өөрчилнө

# Дутагдал

- Код шаардлагагүй ажлыг хийдэг.
  - `peek()` стек хоосон эсэхийг `get` дуудахаас өмнө шалгадаг. Тэгэхээр индексийг шалгах `get` шаардлагагүй болж байна.
  - `add(size(), theElement)` индексийг шалгадаг, орохгүй `for` давталт орсон. Аль нь ч хэрэггүй.
  - `pop()` болохоор `remove` –г дуудахаас өмнө стек хоосон эсэхийг шалгадаг. `remove` индекс шалгаж, орохгүй `for` тавталттай. Аль нь ч хэрэггүй.
  - Иймд код хэрэгцээгүй удаан ажиллана.

# Дүгнэлт

- Кодыг шинээр бичсэнээр хурдан ажиллах боловч хөгжүүлэх гэж хугацаа зарна.
- Програм хангамж хөгжүүлэх зардал, чанарын үзүүлэлт хоёроос сонгох хэрэгтэй.
- Зах зээл гарах хугацаа, чанарын үзүүлэлт хоёроос сонгох хэрэгтэй.
- Анхны кодыг амархан хийгээд, дараа нь чанарын үзүүлэлтийг сайжруулах.

# Хурдан pop()



```
if (empty())  
    throw new EmptyStackException();  
return remove(size() - 1);
```

оронд нь

```
try {return remove(size() - 1);}  
catch(IndexOutOfBoundsException e)  
    {throw new EmptyStackException();}
```



# ЭХНЭЭС НЬ КОДЧИЛОХ

- 1D `stack` массив ашиглах, төрөл нь `Object`.
  - `ArrayLinearList` –д массив `element` ашигласан шиг
- `int top` хувьсагч ашиглах
  - Стекын элементүүд `stack[0:top]` -д
  - Оройн элемент нь `stack[top]`.
  - Ёроолын элемент нь `stack[0]`.
  - `top = -1` бол стек хоосон
  - Стек дэх элементийн тоо `top+1`.



# ЭХНЭЭС НЬ КОДЧИЛОХ

```
package dataStructures;  
import java.util.EmptyStackException;  
import utilities.*; // ChangeArrayLength  
public class ArrayStack implements Stack  
{  
    // өгөгдөл гишүүд  
    int top;           // Стекын орой  
    Object [] stack; // Элементийн массив  
    // байгуулагчид  
    // Stack интерфэйсийн аргууд  
}
```



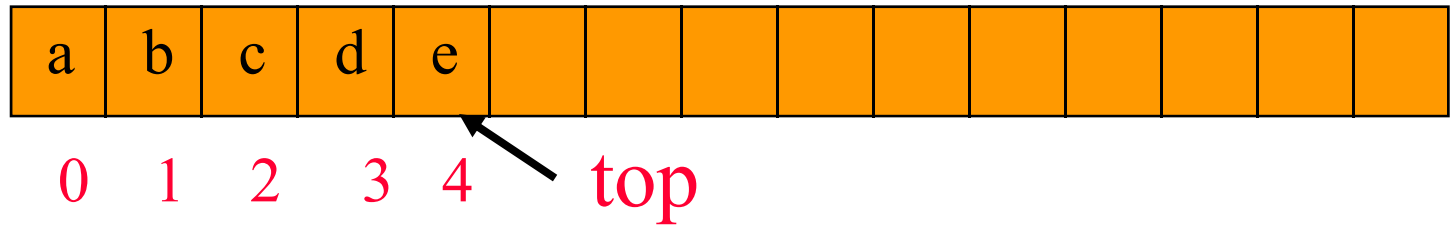
# Байгуулагчид



```
public ArrayStack(int initialCapacity)
{
    if (initialCapacity < 1)
        throw new IllegalArgumentException
            ("initialCapacity must be >= 1");
    stack = new Object [initialCapacity];
    top = -1;
}

public ArrayStack()
{this(10);}
```

# push(...)



```
public void push(Object theElement)
```

```
{
```

```
// хэрэгтэй бол массивын хэмжээг ихэсгэх
```

```
if (top == stack.length - 1)
```

```
    stack = ChangeArrayLength.changeLength1D
```

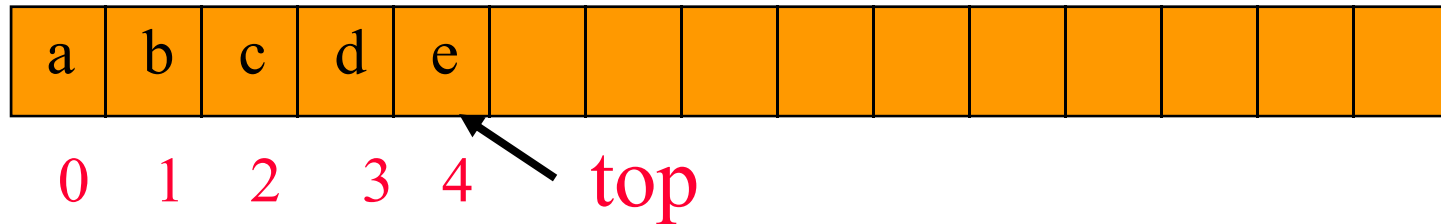
```
        (stack, 2 * stack.length);
```

```
// theElement стекын оройд хийх
```

```
stack[++top] = theElement;
```

```
}
```

# pop()



```
public Object pop()
{
    if (empty())
        throw new EmptyStackException();
    Object topElement = stack[top];
    stack[top--] = null; // хаягдалд өгөх
    return topElement;
}
```

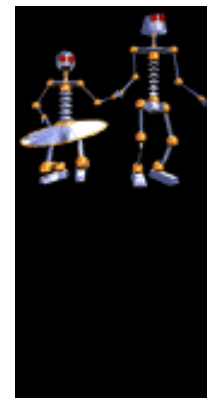
# Холбоосон стекыг эхнээс нь кодчилох

- Сурах бичгээс харна уу.

# java.util.Stack

- `java.util.Vector` -аас уламжлагдсан
- `java.util.Vector` бол шугаман жагсаалтын массив хэрэгжүүлэлт.

# Чанарын үзүүлэлт



500,000 **pop**, **push**, **peek** үйлдэл

Класс	анхны багтаамж	
	10	500,000
ArrayStack	0.44s	0.22s
DerivedArrayStack	0.60s	0.38s
DerivedArrayStackWithCatch	0.55s	0.33s
java.util.Stack	1.15s	-
DerivedLinkedStack	3.20s	3.20s
LinkedStack	2.96s	2.96s





- Шугаман жагсаалт.
- Нэг төгсгөл нь **front** - нүүр.
- Нөгөө төгсгөл нь **rear** - сүүл.
- Нэмэхдээ зөвхөн **сүүлд** нь.
- Устгахдаа зөвхөн **нүүрнийх** нь.

# Автобусны буудлын дараалал



# Автобусны буудлын дараалал



front



rear



# Автобусны буудлын дараалал



front



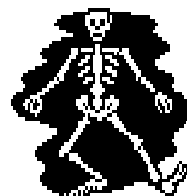
rear



# Автобусны буудлын дараалал



front



rear



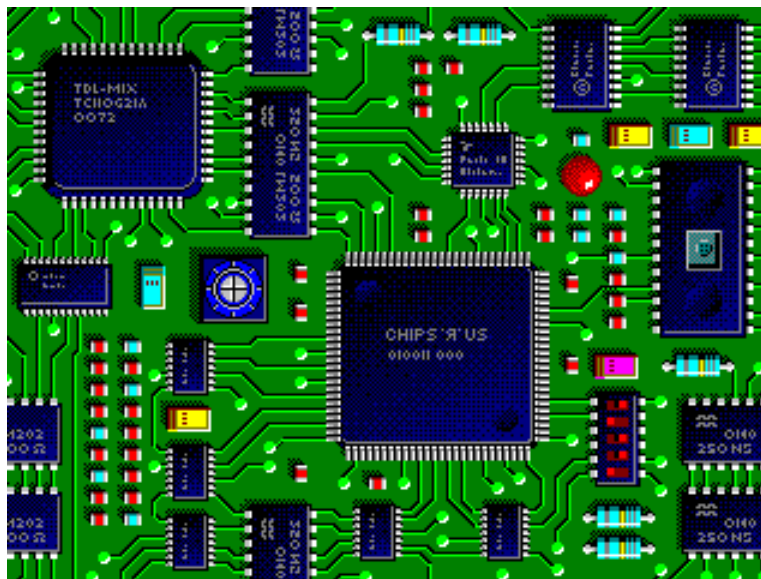
# Интерфейс - Queue

```
public interface Queue
{
    public boolean isEmpty();
    public Object getFrontEelement();
    public Object getRearEelement();
    public void put(Object theObject);
    public Object remove();
}
```

# Стек хэрэглэсэн жишээг эргэж харья



- Стекыг дарааллаар сольж болохгүй жишээ.
  - Хаалт хослох.
  - Ханойн цамхаг.
  - Switchbox routing.
  - Аргыг дуудах ба буцах.
  - Try-catch-throw хэрэгжүүлэлт.
- Стекыг дарааллаар сольж болох жишээ.
  - Төөрөлдсөн оготно.
    - Үр дүн нь гарах дөт замыг олох.

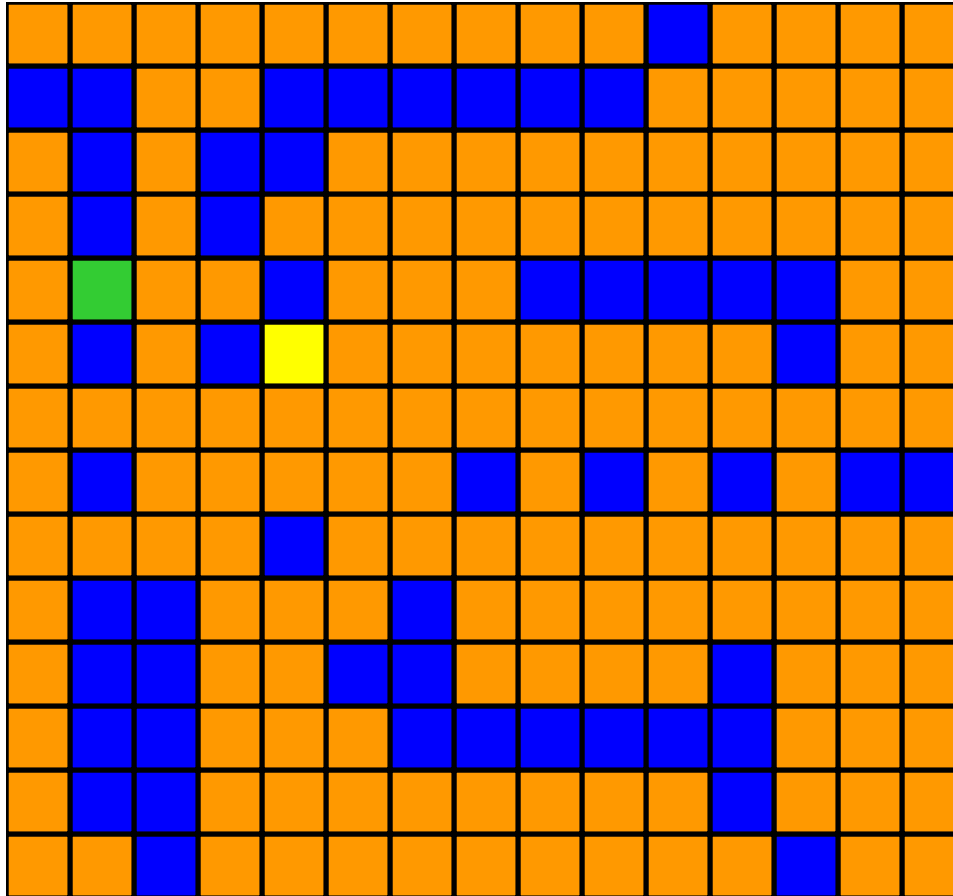
# Зам гаргах







# Lee'ын зам гаргагч

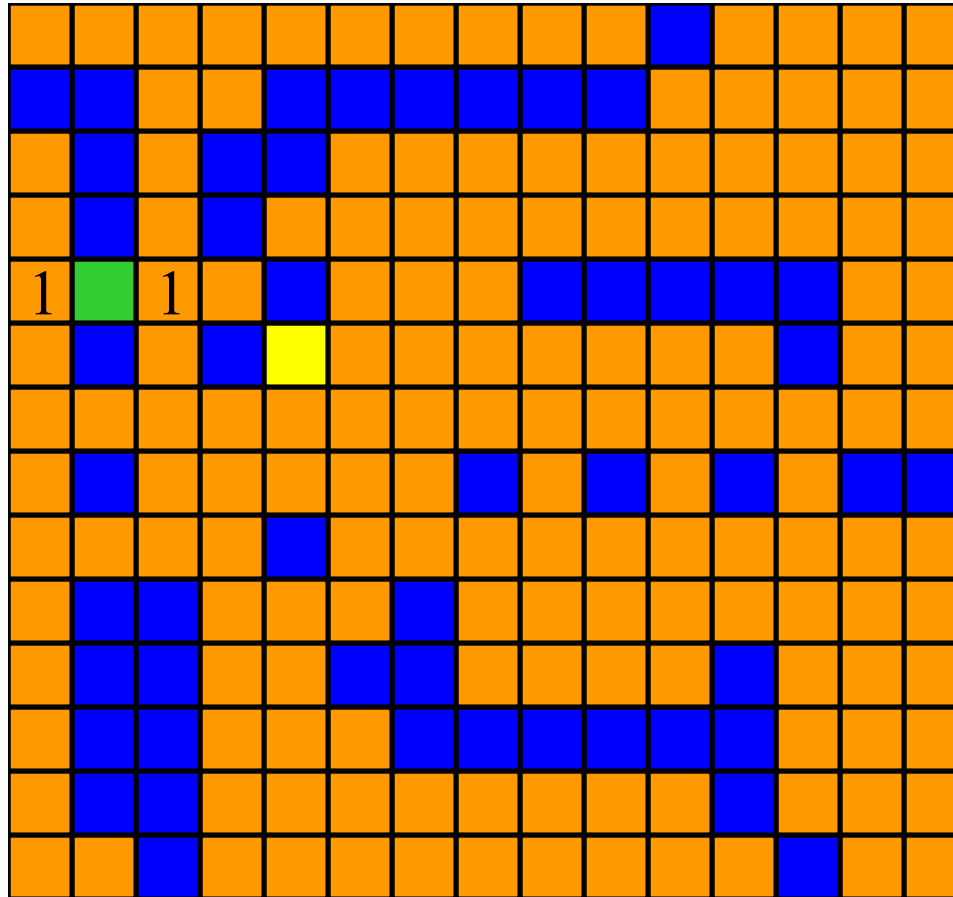
 start pin  
 end pin



Гараанаас хүрч болох нүдийг **1** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin  
 end pin

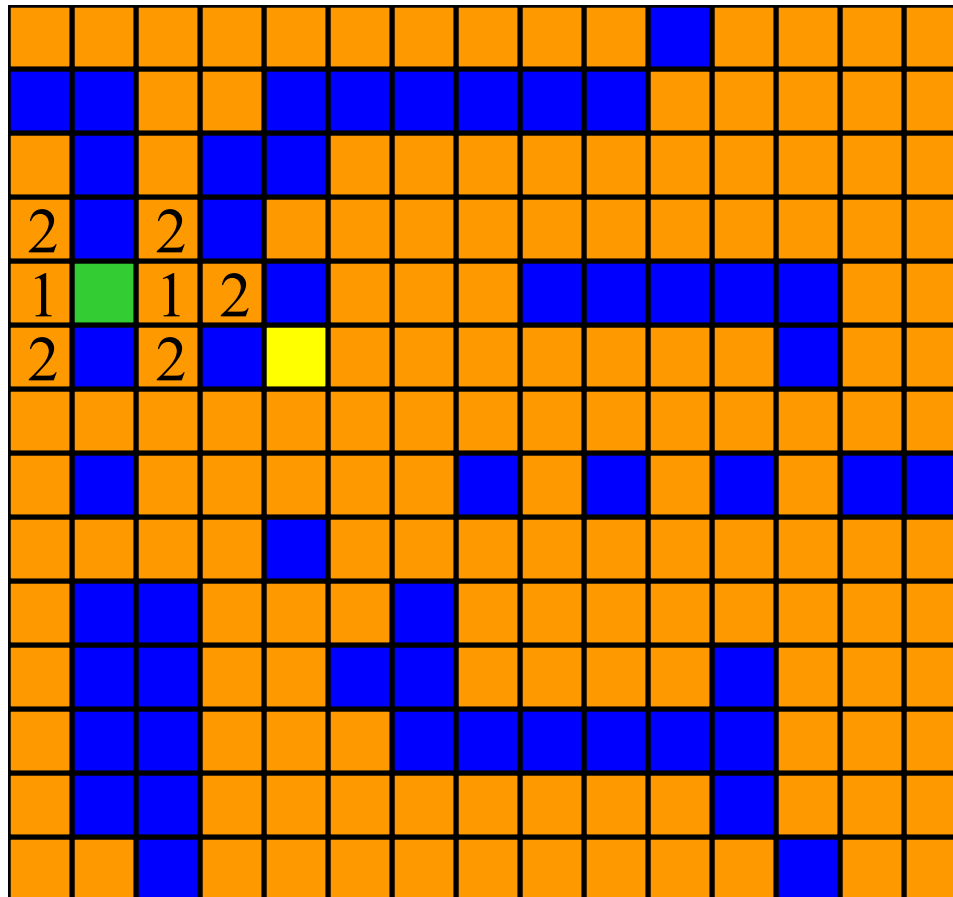


Гараанаас хүрч болох нүдийг 2 гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin

 end pin

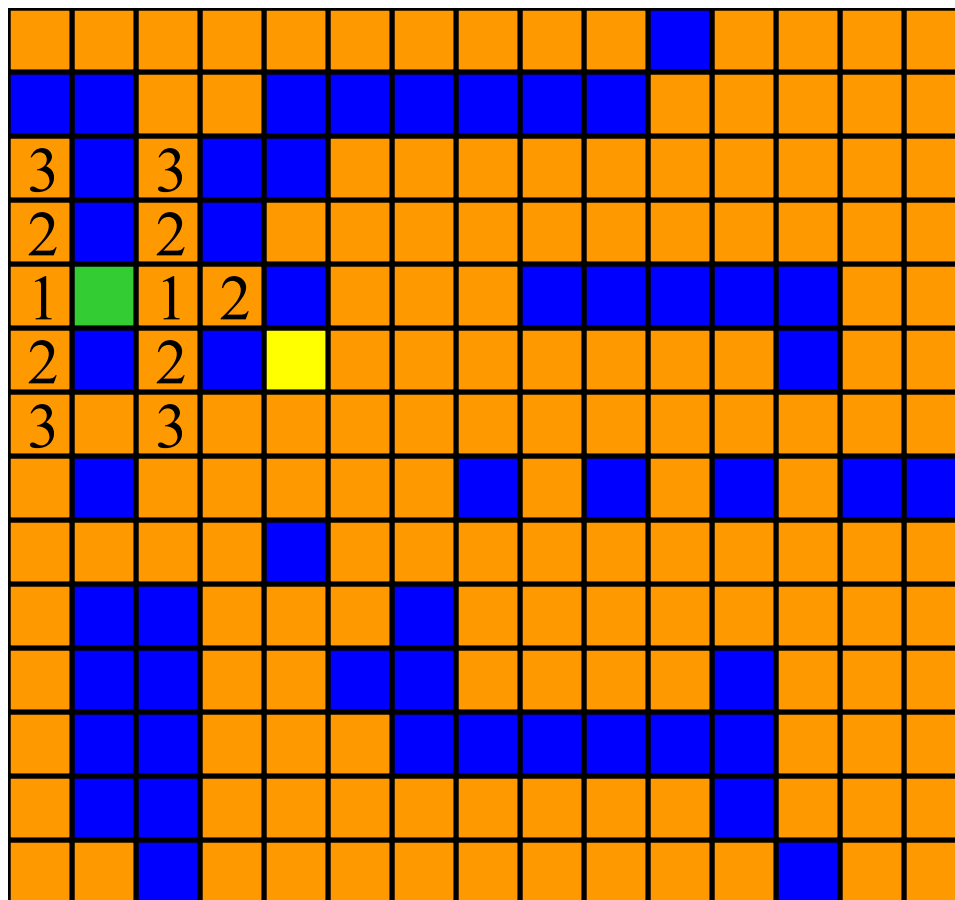


Гараанаас хүрч болох нүдийг 3 гэж  
тэмдэглэе.

# Lee'ын зам гаргагч



 start pin

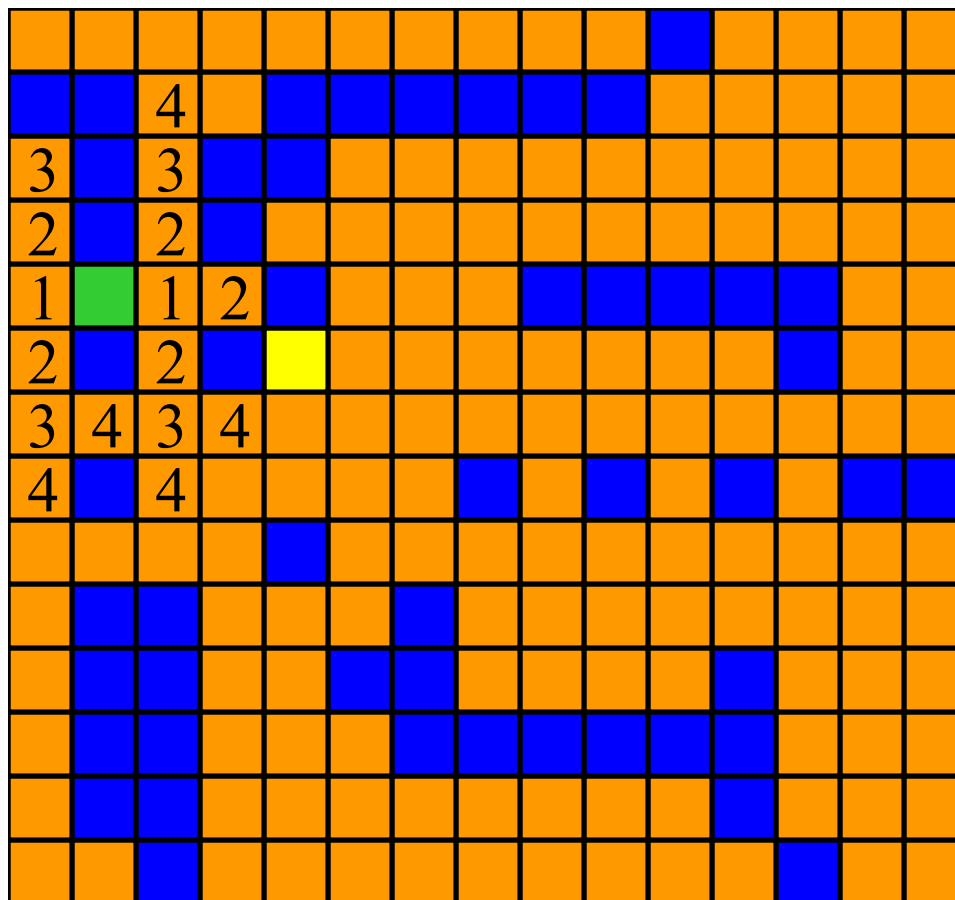
 end pin



Гараанаас хүрч болох нүдийг 4 гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin  
 end pin

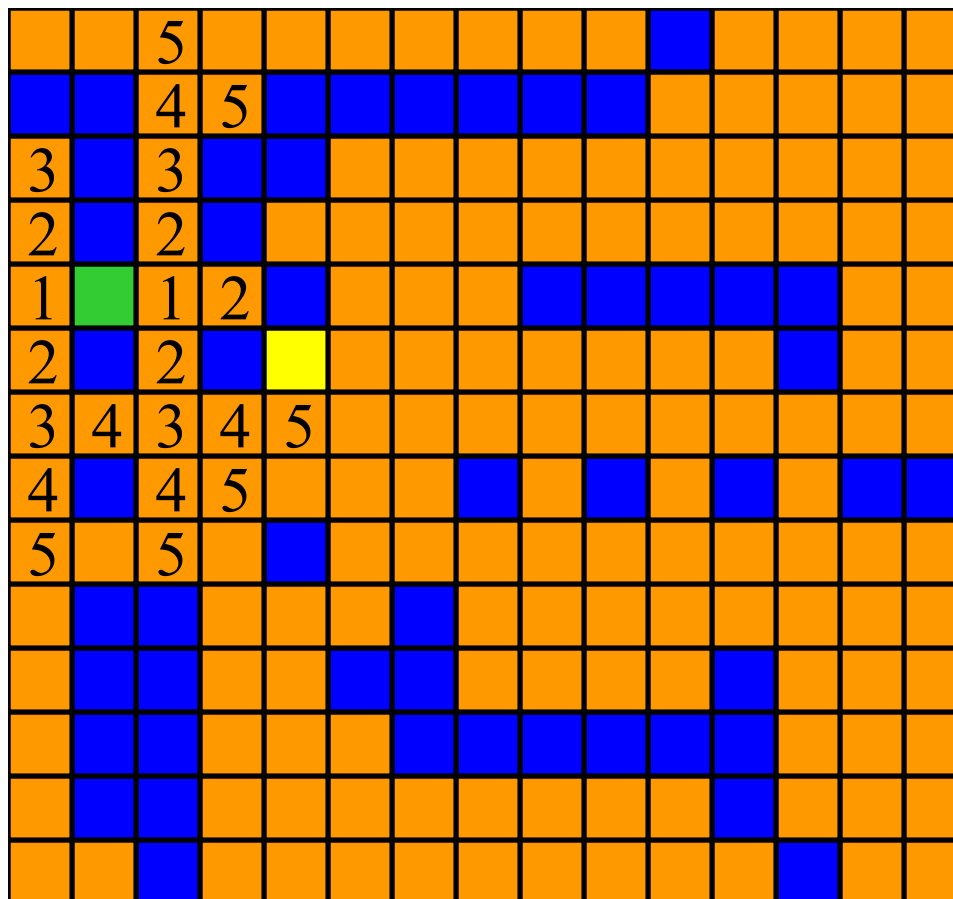


Гараанаас хүрч болох нүдийг **5** гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin

 end pin

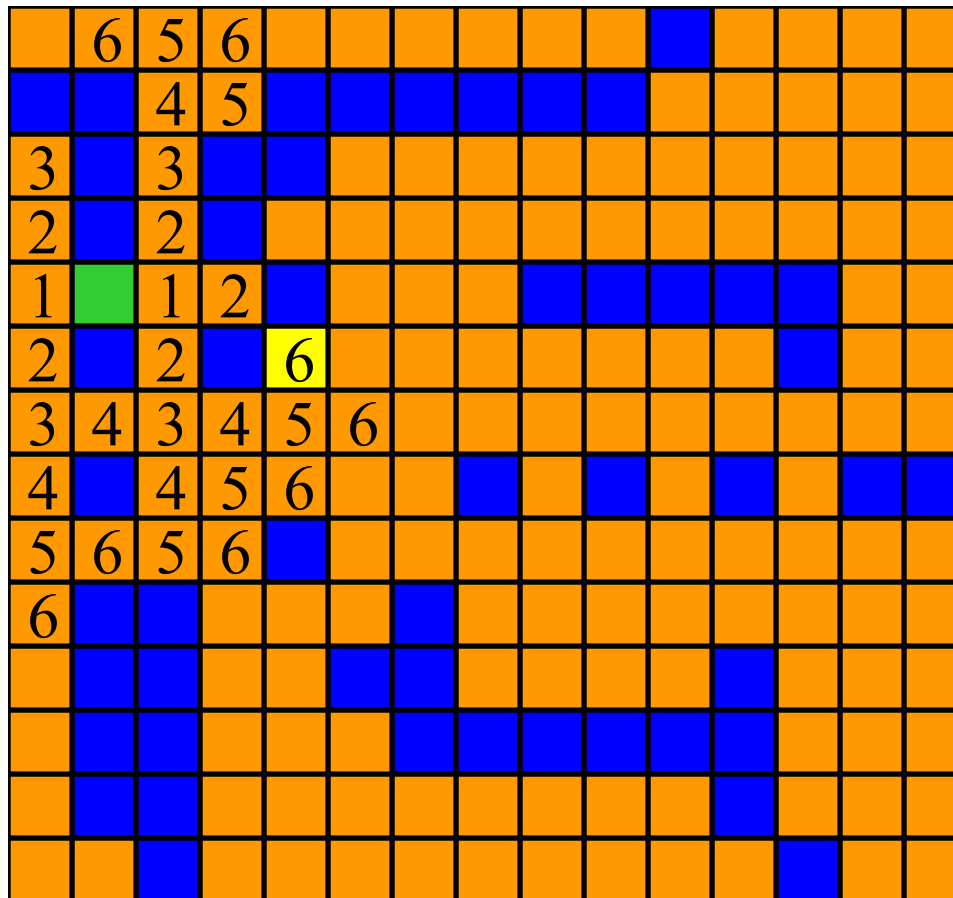


Гараанаас хүрч болох нүдийг 6 гэж  
тэмдэглэе.

# Lee'ын зам гаргагч

 start pin

 end pin

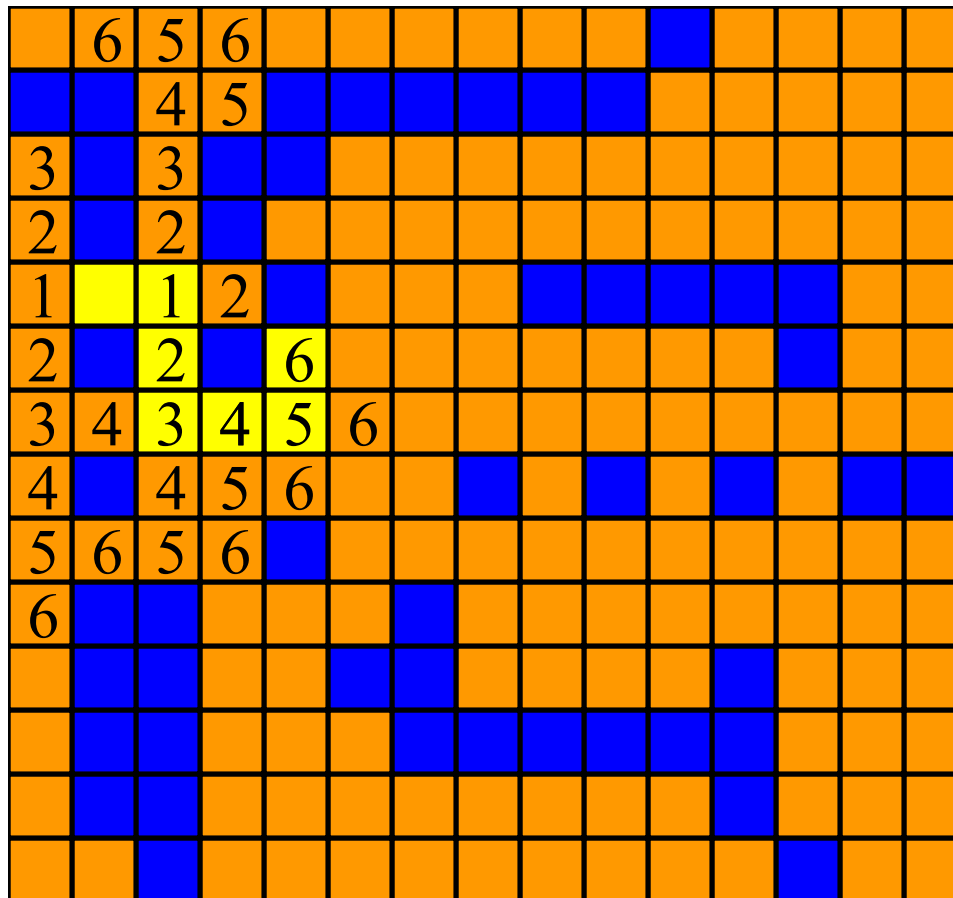


Төгсгөлд хүрлээ. Буцья.

# Lee'ын зам гаргагч

 start pin

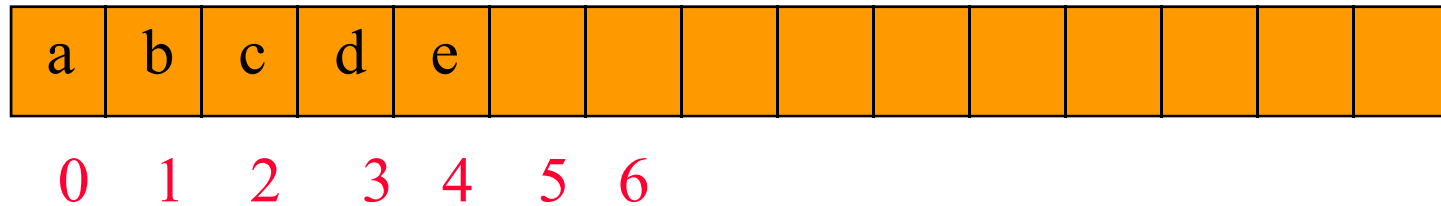
 end pin



Төгсгөлд хүрлээ. Буцья.



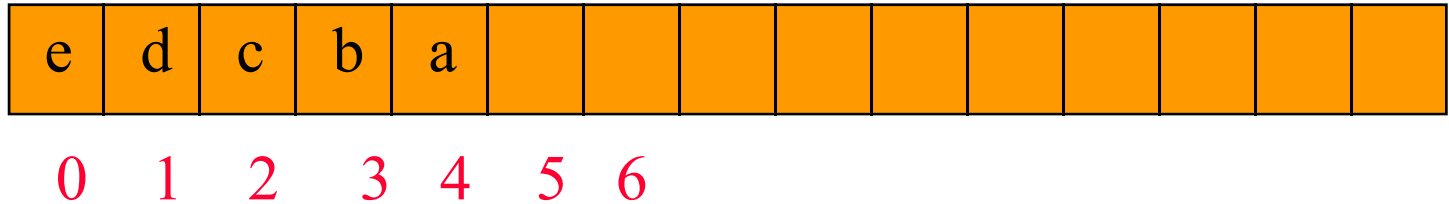
# ArrayList –ээс уламжлах



➤ Жагсаалтын зүүн төгсгөл нүүр, баруун төгсгөл нь сүүл бол

- `Queue.isEmpty() => super.isEmpty()`
- `getFrontElement() => get(0)`
- `getRearElement() => get(size() - 1)`
- `put(theObject) => add(size(), theObject)`
- `remove() => remove(0)`

# ArrayList –ээс уламжлах



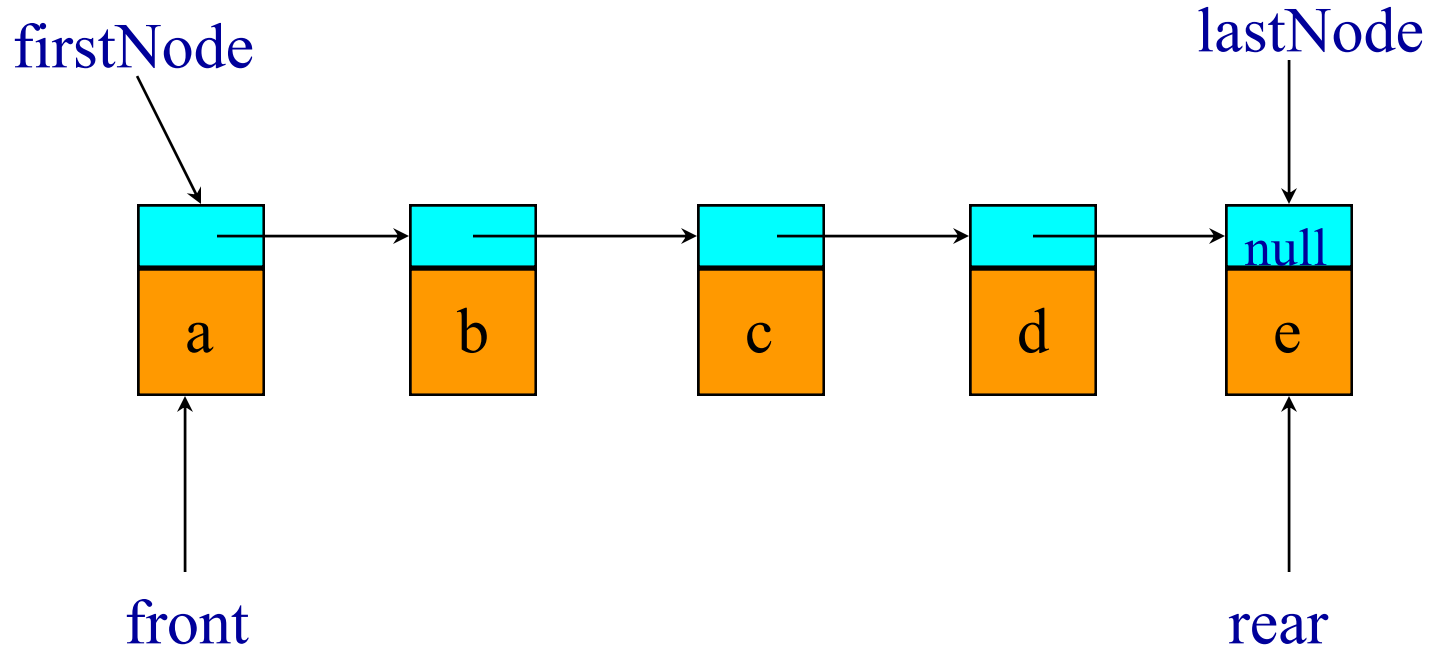
— Жагсаалтын зүүн төгсгөл сүүл, баруун төгсгөл нь нүүр бол

- `Queue.isEmpty() => super.isEmpty()`
- `getFrontElement() => get(size() - 1)`
- `getRearElement() => get(0)`
- `put(theObject) => add(0, theObject)`
- `remove() => remove(size() - 1)`

# ArrayList –ээс уламжлах

- Үйлдлүүдийг гүйцэтгэх хугацаа богино (массивыг хоёр дахин ихэсгэхээс бусад), бидэнд сайжруулсан массив дүрслэл хэрэгтэй

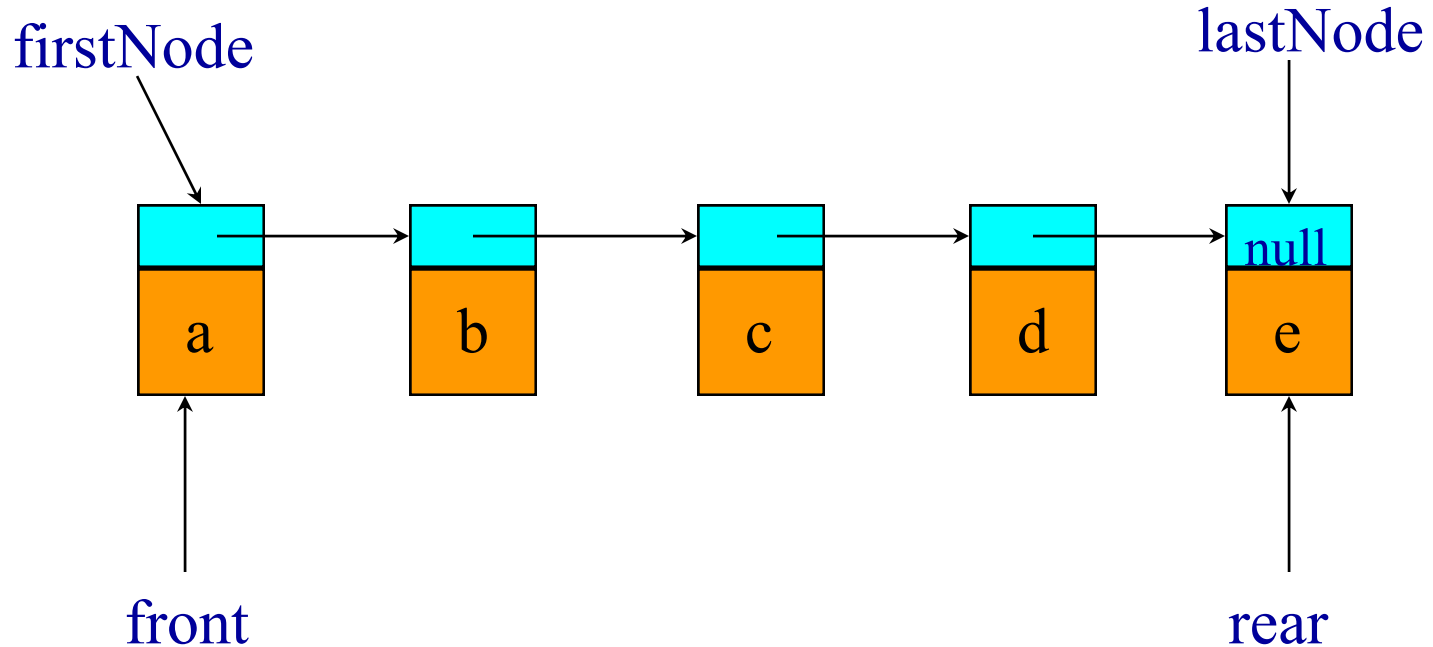
# ExtendedChain –ээс уламжлах



➤ жагсаалтын зүүн төгсгөл нь нүүр, баруун төгсгөл нь сүүл бол

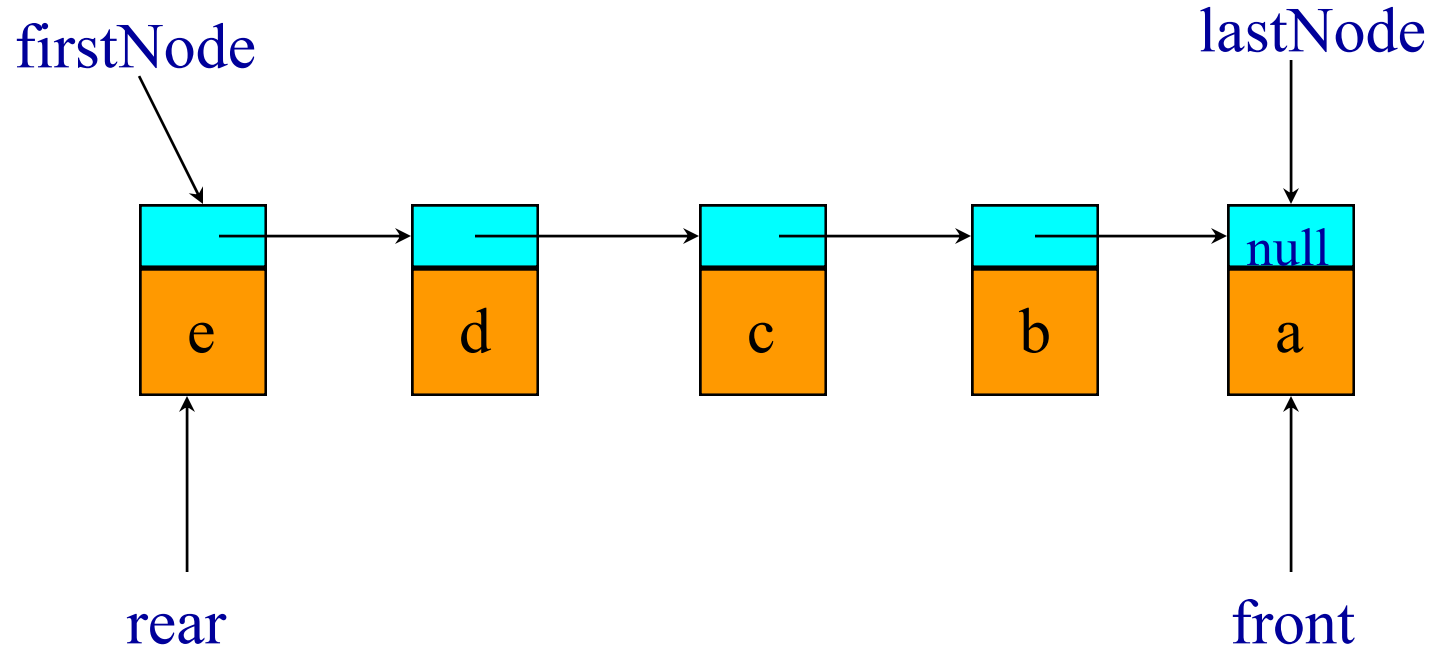
- `Queue.isEmpty() => super.isEmpty()`  
– `getFrontElement() => get(0)`

# ExtendedChain –ээс уламжлах



- `getRearElement()` => `getLast()` ... шинэ арга
  - `put(theObject)` => `append(theObject)`
  - `remove()` => `remove(0)`

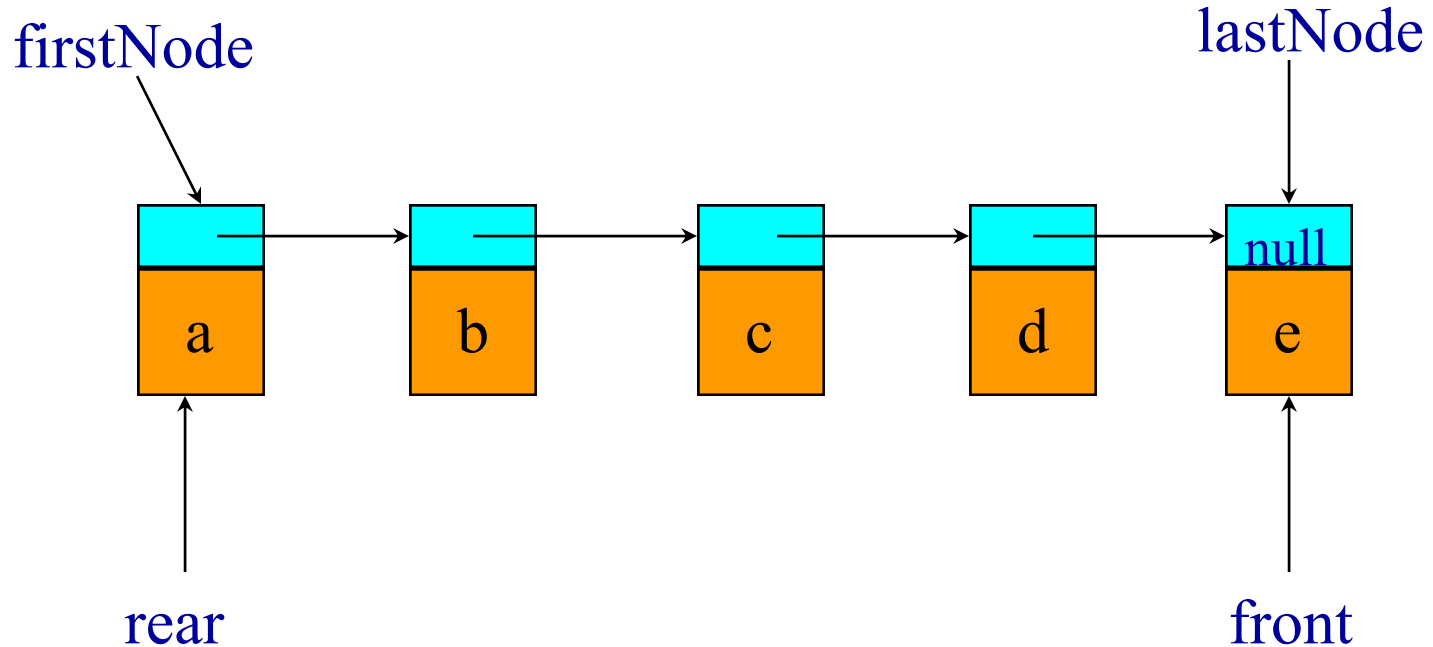
# Derive From ExtendedChain



➤ Жагсаалтын зүүн төгсгөл сүүл, баруун төгсгөл нь нүүр бол

- `Queue.isEmpty() => super.isEmpty()`  
– `getFrontElement() => getLast()`

# ExtendedChain –ээс уламжлах



- `getRearElement()`  $\Rightarrow$  `get(0)`
  - `put(theObject)`  $\Rightarrow$  `add(0, theObject)`
  - `remove()`  $\Rightarrow$  `remove(size-1)`

# Өөрчилсөн холбоост код

- **ExtendedChain** —ээс уламжилснаас сайн үзүүлэлт хэрэгтэй бол **Queue** —д зориулсан холбоост классыг эхнээс нь кодчилох хэрэгтэй

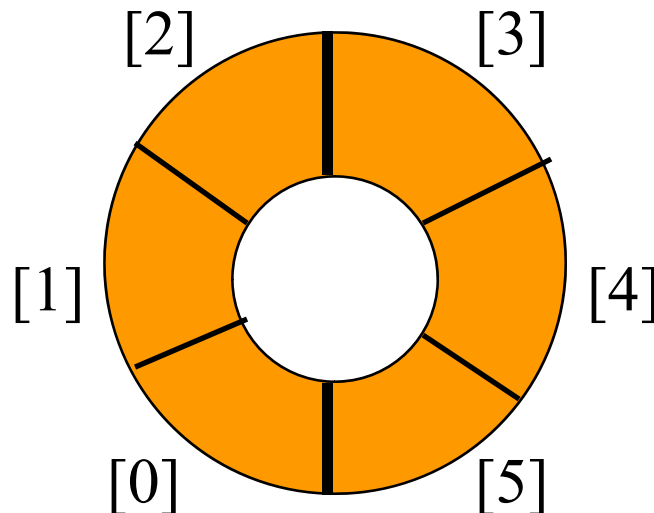


# Өөрчилсөн массив дараалал

- 1D массив **queue** -г ашиглая

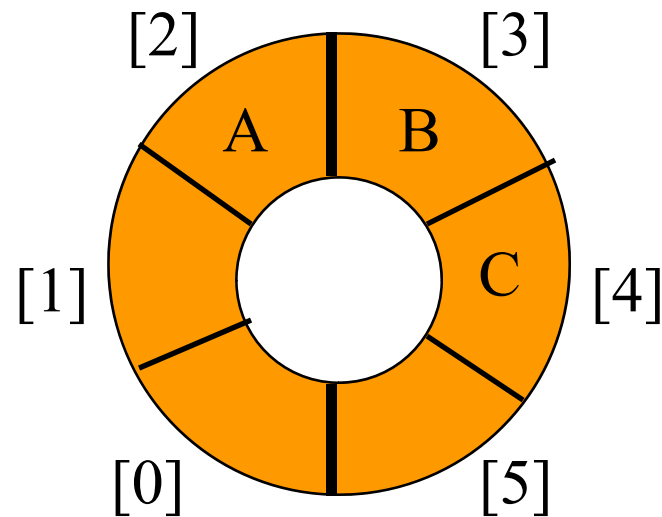
**queue[]** 

- Массивын цагираг харагдац.



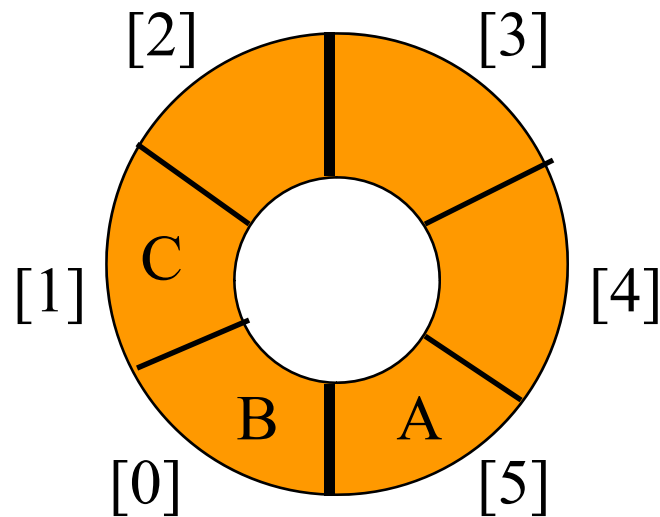
# Өөрчилсөн массив дараалал

- 3 элементтэй хувилбар.



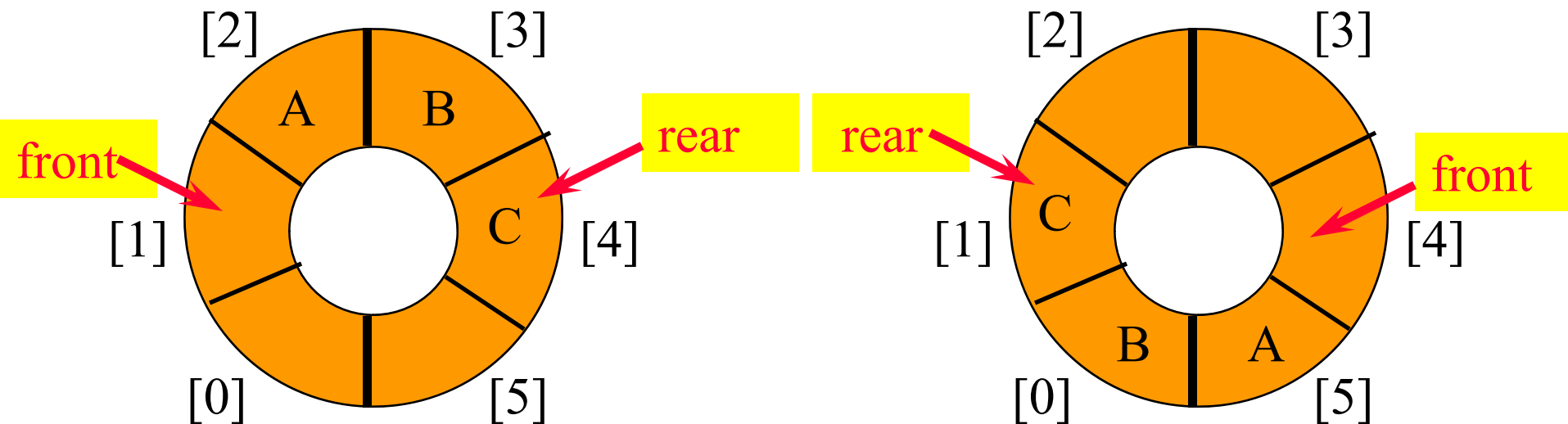
# Өөрчилсөн массив дараалал

- 3 элементтэй өөр нэг хувилбар.



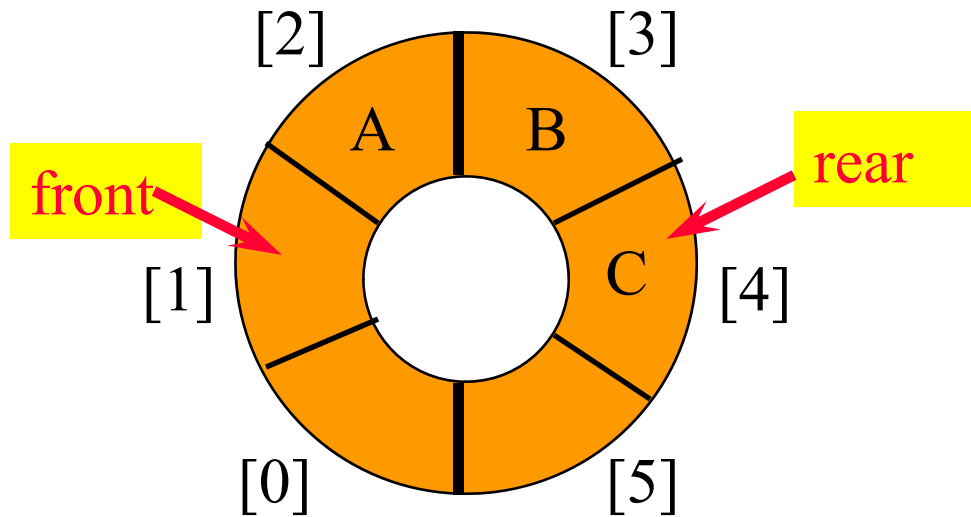
# Өөрчилсөн массив дараалал

- Бүхэл хувьсагч **front** , **rear** -г ашиглая
  - **front** эхний элементээс цагийн зүүний дагуу явсан нэг байршил
  - **rear** сүүлийн элементийн байршлыг өгнө



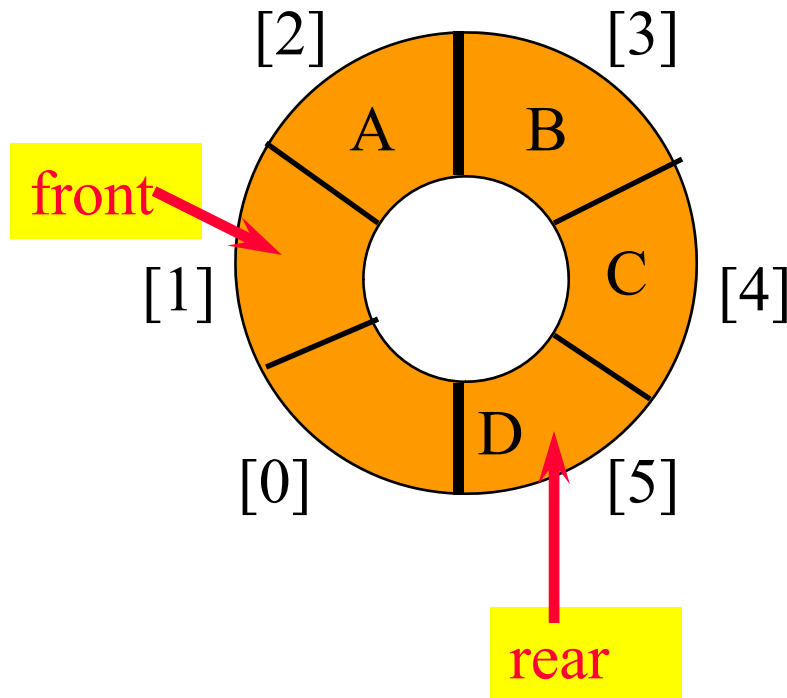
# Элемент нэмэх

- **rear** цагийн дагуу нэгээр хөдөлгөнө.



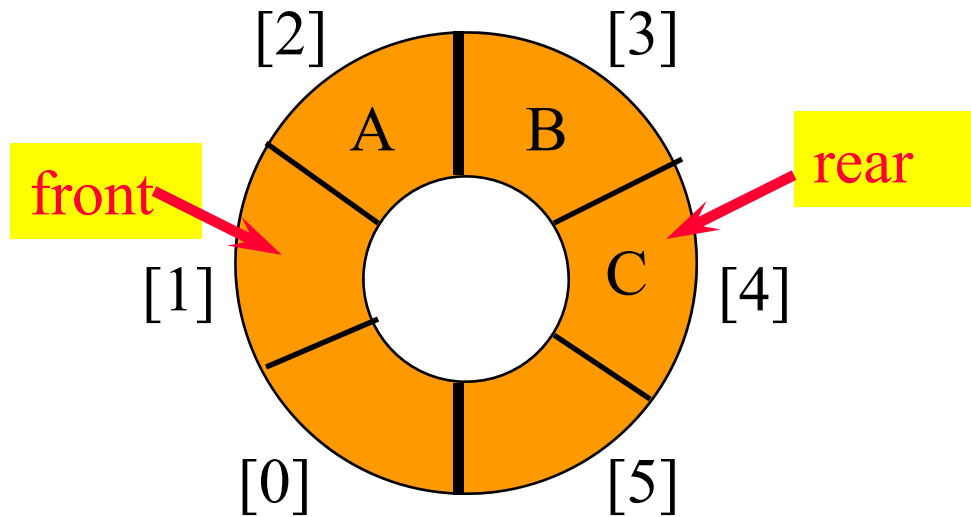
# Элемент нэмэх

- **rear** цагийн дагуу нэгээр хөдөлгөнө.
- **queue[rear]** -д элементийг хийнэ



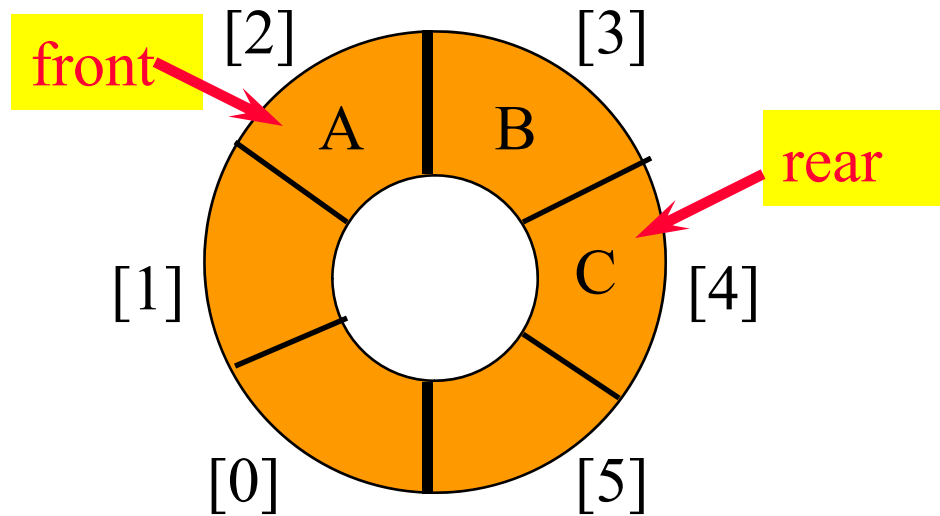
# Элементийг устгах

- **front** цагийн дагуу нэгээр хөдөлгөнө.



# Элемент устгах

- **front** цагийн дагуу нэгээр хөдөлгөнө.
- **queue[front]** -аас элемент авна.

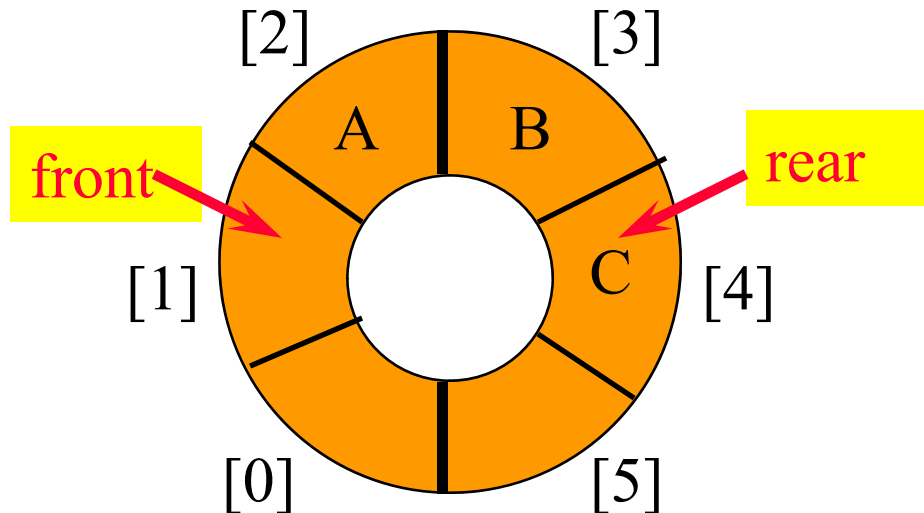




# rear –г цагийн дагуу хөдөлгөх

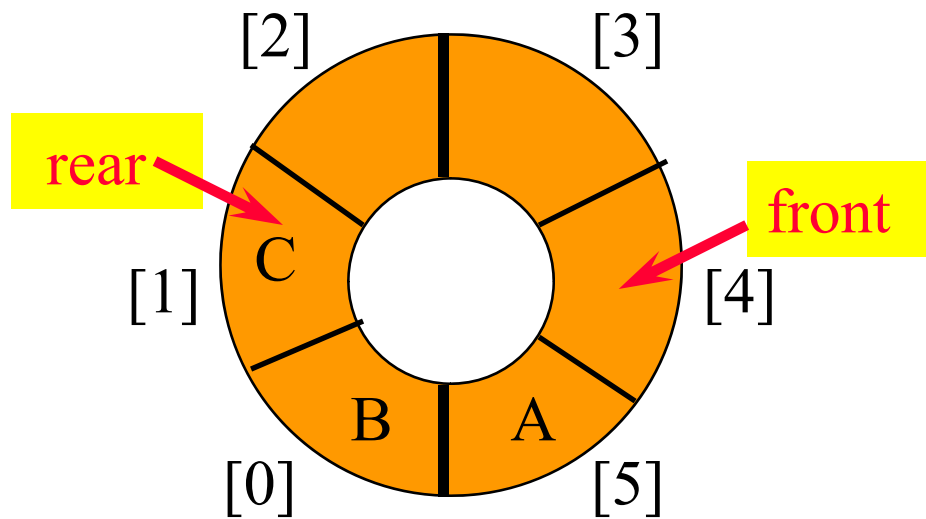
- `rear++;`

`if (rear == queue.length) rear = 0;`

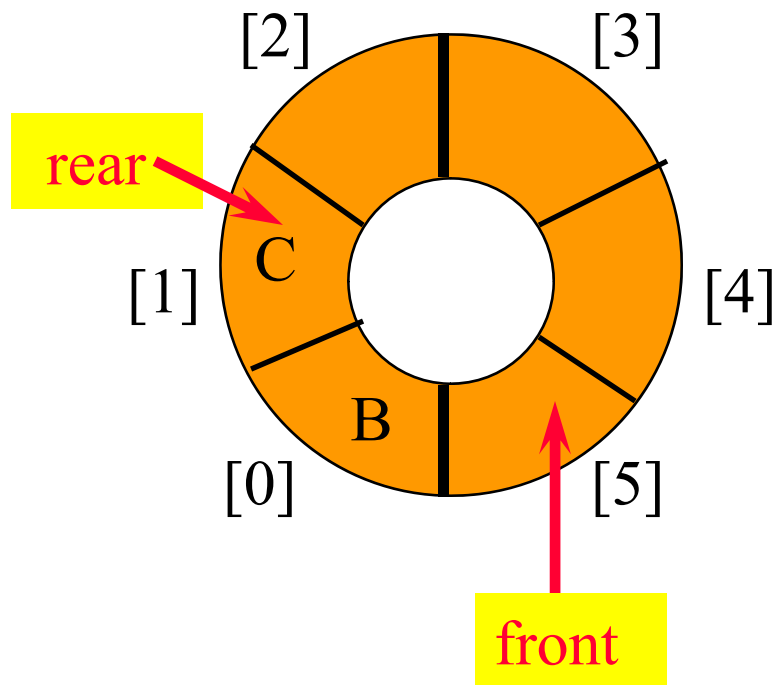


- `rear = (rear + 1) % queue.length;`

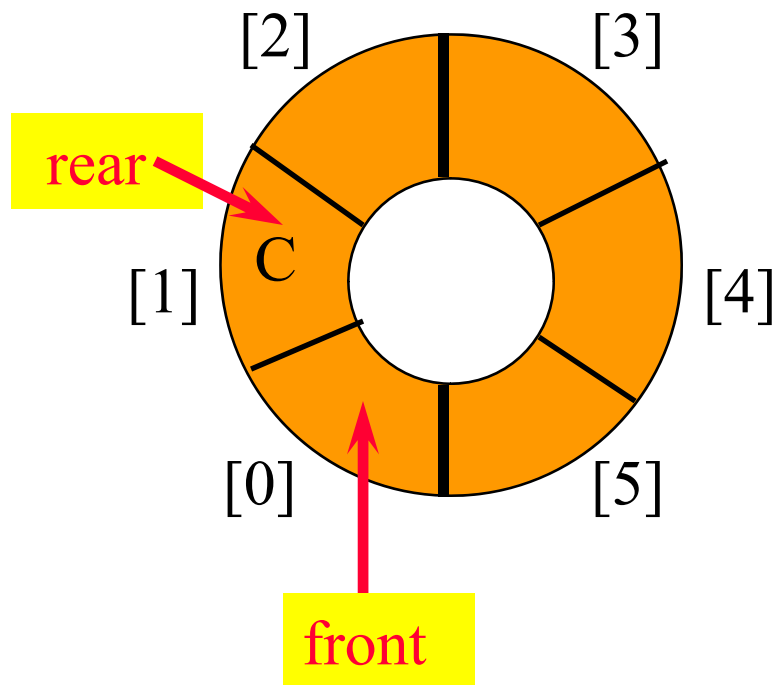
# Дарааллыг хоослох



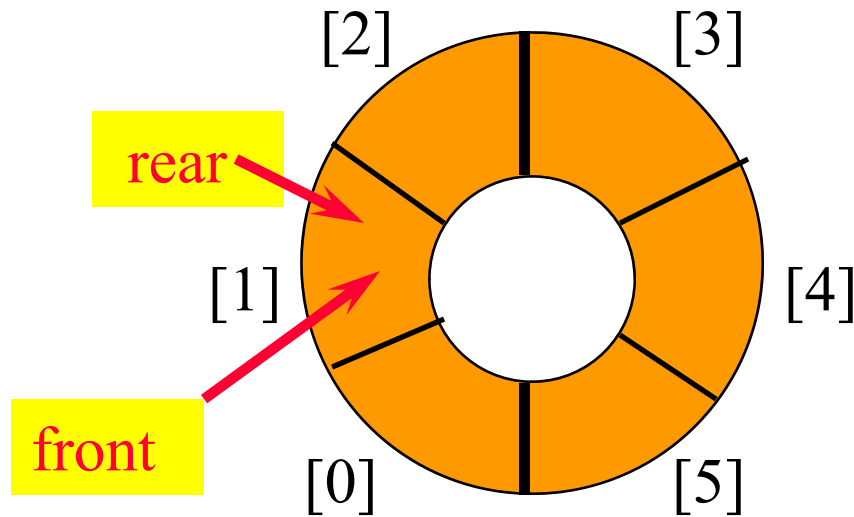
# Дарааллыг хоослох



# Дарааллыг хоослох

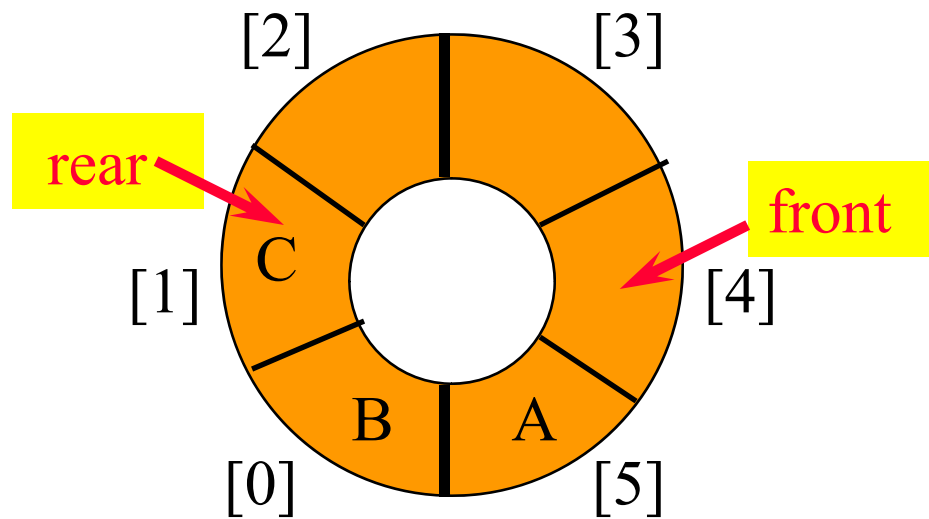


# Дарааллыг хоослох

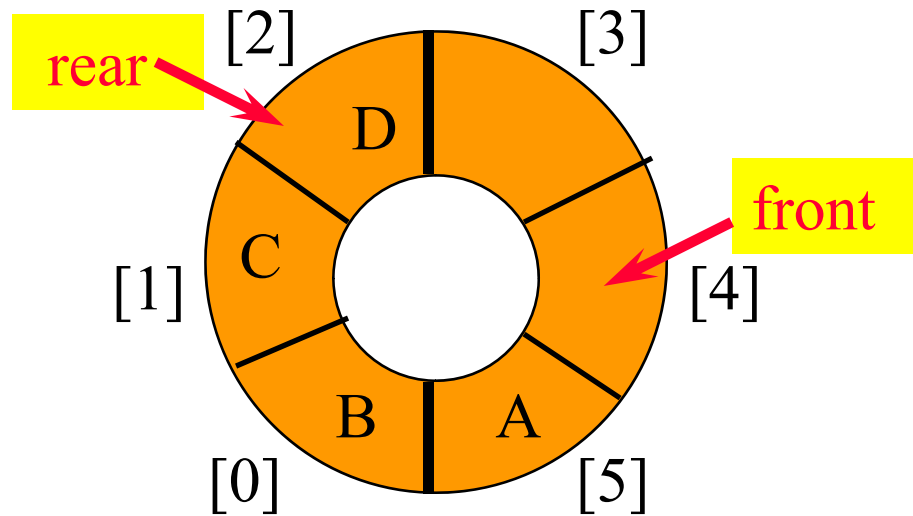


- Дараалсан устгалтуудаар  $front = rear$  болоход дараалал хоослогдоно
- Анх байгуулагдахад хоосон байна.
- Иймд эхэндээ  $front = rear = 0$ .

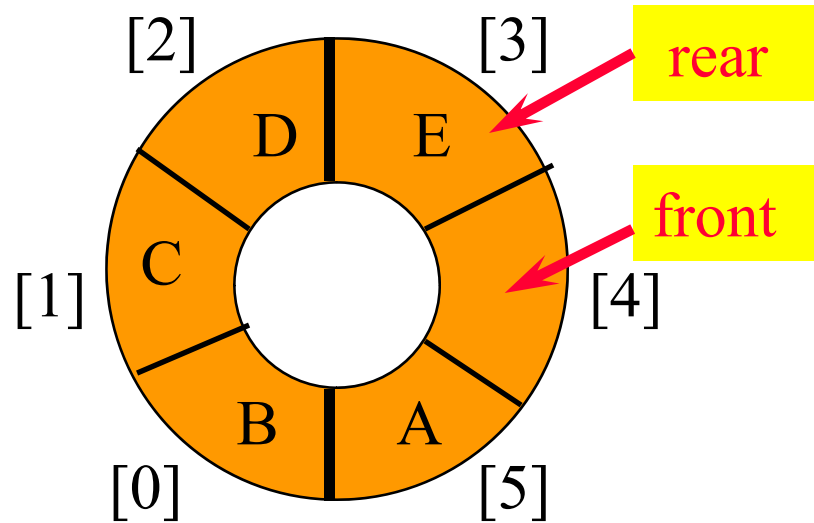
# Дарааллын савыг дүүргэх



# Дарааллын савыг дүүргэх

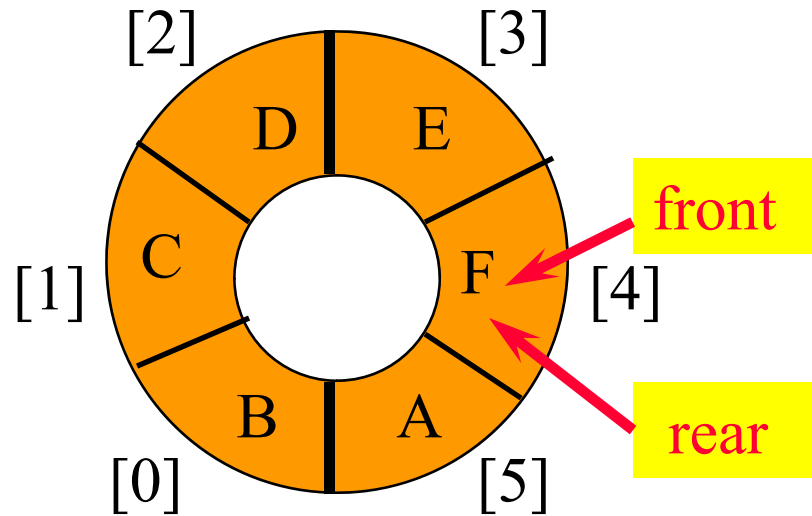


# Дарааллын савыг дүүргэх





# Дарааллын савыг дүүргэх



- Дараалсан нэмэлтүүдээр  $front = rear$  болоход сав дүүрнэ
- Сав дүүрэн үү, хоосон уу гэдгийг ялгах хэзүү боллоо!

# Анхаар!!!!

- Засвар.
  - Дарааллыг бүү дүүргэ.
    - Нэмэгдэх элемент дарааллыг дүүргэх бол массивын хэмжээг нэмэгдүүл.
    - Үүнийг сурах бичигт үзүүлсэн.
  - Булын хувьсагч **lastOperationIsPut** -г ашигла
    - **put** үйлдэл бүрийн дараа **true** болго
    - **remove** үйлдэл бүрийн дараа **false** болго
    - дараалал хоосон -> **(front == rear) && !lastOperationIsPut**
    - Дараалал дүүрэн -> **(front == rear) && lastOperationIsPut**

# Анхаар!!!!

- Засвар (үргэлжлэл).
  - Бүхэл хувьсагч **size** -г ашигла
    - **put** үйлдэл бүрийн дараа **size++**.
    - **remove** үйлдэл бүрийн дараа **size--**.
    - Дараалал хоосон -> **(size == 0)**
    - Дараалал дүүрэн -> **(size == queue.length)**
  - Эхний хувилбараар чанарын үзүүлэлт арай дээр байх болно.