алгоритмы и структуры данных

02

абстрактный тип данных • контейнер • структура данных

Нестеров P.A., PhD, доцент
департамента программной инженерии

сентябрь 2024

2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

план лекции

01

абстрактный тип данных: отношения между объектами

02

структура данных: размещение и эффективность

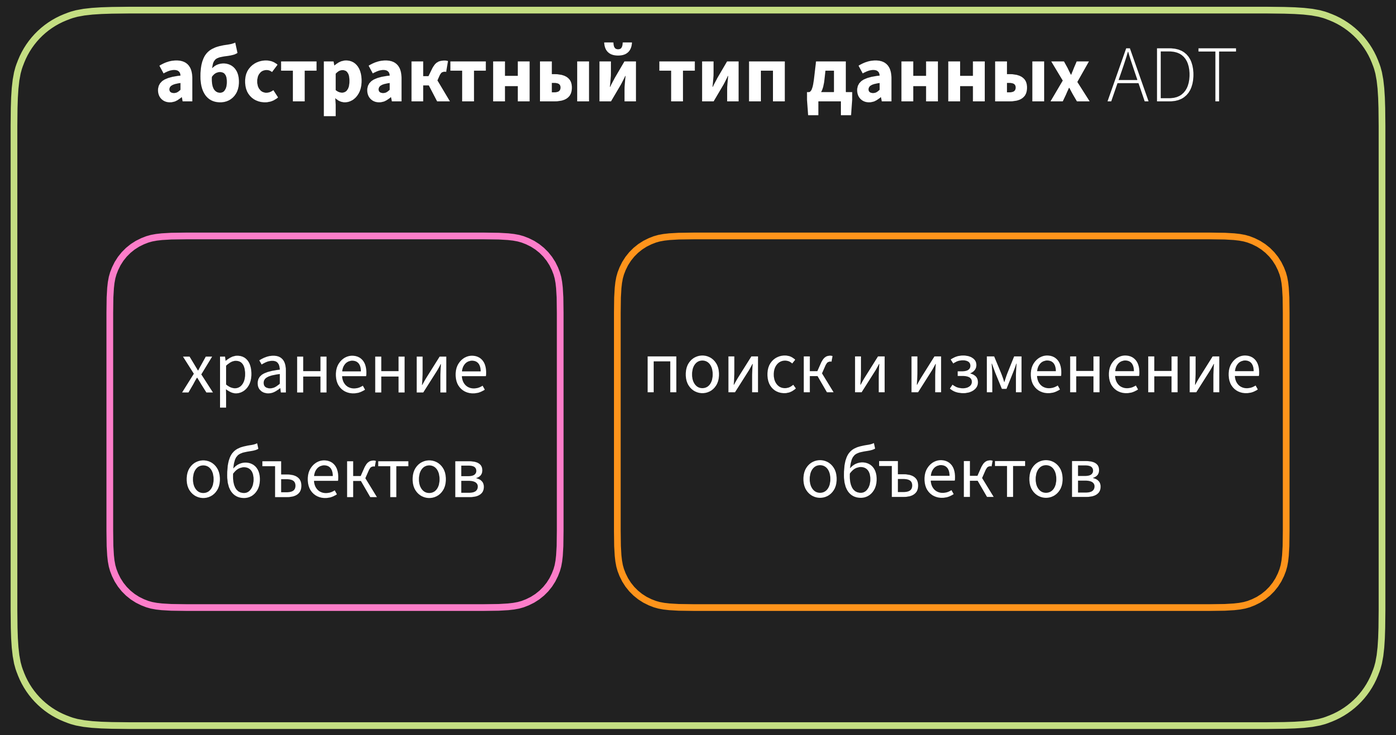
03

XOR–связный список: эмуляция двусвязности

абстрактный тип данных ADT

хранение объектов

поиск и изменение объектов



💡 ADT Контейнер — наиболее общая модель хранения и предоставления доступа к объектам

`template<typename T> class Container {…}`

операция	реализация
создание	`Container()`
копирование	`Container(const Container&)`
удаление	`~Container()`
очистка	`void clear()`
объединение	`void insert()`
пересечение	`. . .`

операции
с контейнером

запросы
к контейнеру

операции
с объектами

`template<typename T> class Container {…}`

операция	реализация
проверка на пустоту	`bool empty() const`
количество объектов	`size_type size() const`
максимальная вместимость	`size_type max_size() const`

операции с контейнером

запросы к контейнеру

операции с объектами

`template<typename T> class Container {…}`

операция	реализация
вставка	`void insert(const T&)`
удаление	`void erase(const T&)`
доступ	`iterator find(const T&) const`
количество копий объекта	`size_type count(const T&) const`
итерация	`iterator begin() const` `iterator end() const`

операции с контейнером

запросы к контейнеру

операции с объектами

стандартные контейнеры • STL

уникальные объекты	дублирующиеся объекты
`std::set<Key>`	`std::multiset<Key>`
`std::map<Key, T>`	`std::multimap<Key, T>`

STL_SET.cpp

```cpp
#include <iostream>
#include <set>

int main() {
    std::set<int> ints;

    for (int i = -100; i <= 100; ++i) {
        ints.insert(i * i);
    }
    std::cout << ints.size() << std::endl;

    ints.erase(50);
    ints.erase(9);
    std::cout << ints.size() << std::endl;

    return 0;
}
```

`insert` проигнорирует дублирующиеся объекты

`std::set` не порождает самостоятельных исключений

стандартные контейнеры • STL

уникальные объекты	дублирующиеся объекты
`std::set<Key>`	`std::multiset<Key>`
`std::map<Key, T>`	`std::multimap<Key, T>`

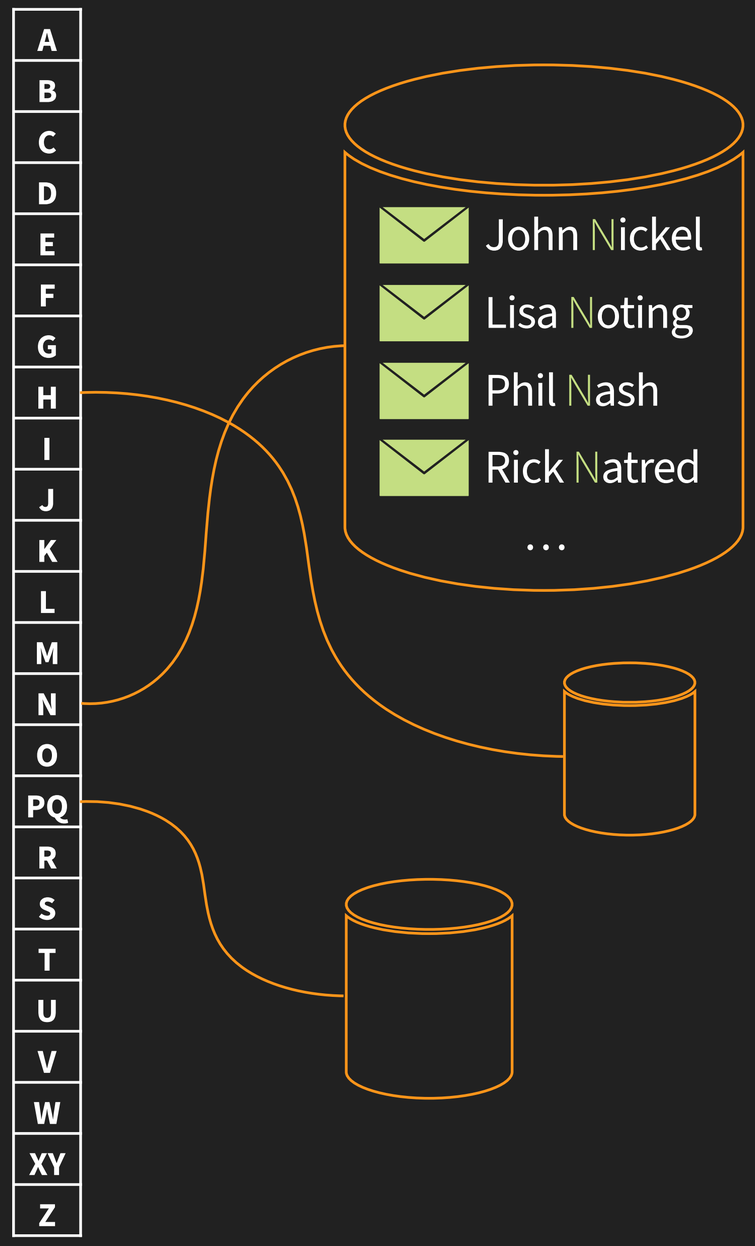
помимо хранения объектов, контейнер также хранит отношение порядка между ними

беспорядок • хеш–таблица

- поиск объекта за одно и то же время
  вне зависимости от их количества
- на 30% процентов больше памяти,
  чем для хранения самих объектов

организация e–mail для отдела из 24 человек,
в котором часто меняются сотрудники:

1. сотрудники идентифицируются по первой букве своей фамилии — хеш (John Nickel)
2. письма помещаются в одну из 24 корзин, соответствующую фамилии сотрудника



X — множество объектов

$R \subseteq X \times X$ — отношение

- рефлексивность: $\forall x \in X\colon\ xRx$
- симметричность: $\forall x, y \in X\colon\ xRy \Rightarrow yRx$
- транзитивность: $\forall x, y, z \in X\colon\ (xRy, yRz) \Rightarrow xRz$

X — множество объектов

$\leq \subseteq X \times X$ — отношение

➡ рефлексивность: $x \leq x$

➡ антисимметричность: $(x \leq y, y \leq x) \Rightarrow x = y$

➡ транзитивность: $(x \leq y, y \leq z) \Rightarrow x \leq z$

линейный порядок

ADT List, Sorted List, Stack, Queue, Deque, String, PriorityQueue

числа	$\ldots \leq -9 \leq \ldots \leq 1 \leq 2 \leq \ldots$ $1.2 \leq \ldots \leq 1.21 \leq 2.65 \leq \ldots$
символы	$A \leq B \leq \ldots \leq a \leq b \leq \ldots$
адреса в памяти	`0x0000, …, 0xFFFF`

$$(x_1, x_2) \leq (x_3, x_4) \Leftrightarrow \begin{bmatrix} x_1 < x_3 \\ x_1 = x_3, x_2 \leq x_4 \end{bmatrix}$$

1. найти k–ую порядковую статистику контейнера
2. найти объекты, которые лежат в интервале $[a, b]$
3. найти предыдущий и следующий объект

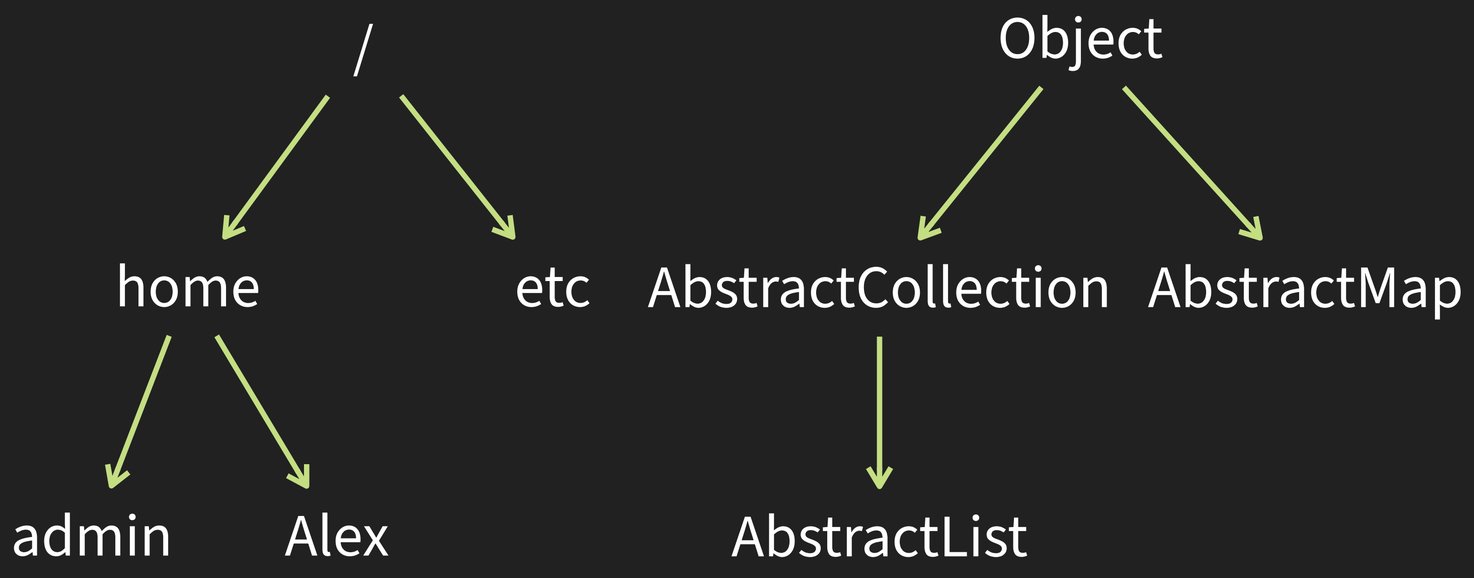
…

X — множество объектов

$\prec \subseteq X \times X$ — отношение

- антирефлексивность: $\neg(x \prec x)$
- асимметричность: $x \prec y \Rightarrow \neg(y \prec x)$
- транзитивность: $(x \prec y, y \prec z) \Rightarrow x \prec z$

иерархия



SCOPES.cpp

```cpp
int func() {
    int a;
    {
        int b;
    }
    {
        int d;
    }
    return a;
}
```

X — множество объектов

$\prec \subseteq X \times X$ — отношение

- ➡ антирефлексивность: $\neg(x \prec x)$
- ➡ асимметричность: $x \prec y \Rightarrow \neg(y \prec x)$
- ➡ транзитивность: $(x \prec y, y \prec z) \Rightarrow x \prec z$

иерархия

ADT Tree

SCOPES.cpp

```cpp
int func() {
    int a;
    {
        int b;
    }
    {
        int d;
    }
    return a;
}
```

1. проверить, что два объекта x_1 и x_2 связаны отношением иерархии
2. проверить, что два объекта x_1 и x_2 находятся на одном уровне иерархии
3. найти ближайшего общего предка для двух объектов x_1 и x_2

…

X — множество объектов

$\sqsubset \subseteq X \times X$ — отношение

частичный порядок

- ➡ антирефлексивность: $\neg(x \sqsubset x)$
- ➡ асимметричность: $x \sqsubset y \Rightarrow \neg(y \sqsubset x)$
- ➡ транзитивность: $(x \sqsubset y, y \sqsubset z) \Rightarrow x \sqsubset z$

ADT DirectedAcyclicGraph

1. проверить, предшествует ли объект x_1 объекту x_2
2. найти объекты, которым не предшествует ни один объект
3. найти всех предков [потомков] объекта x

…

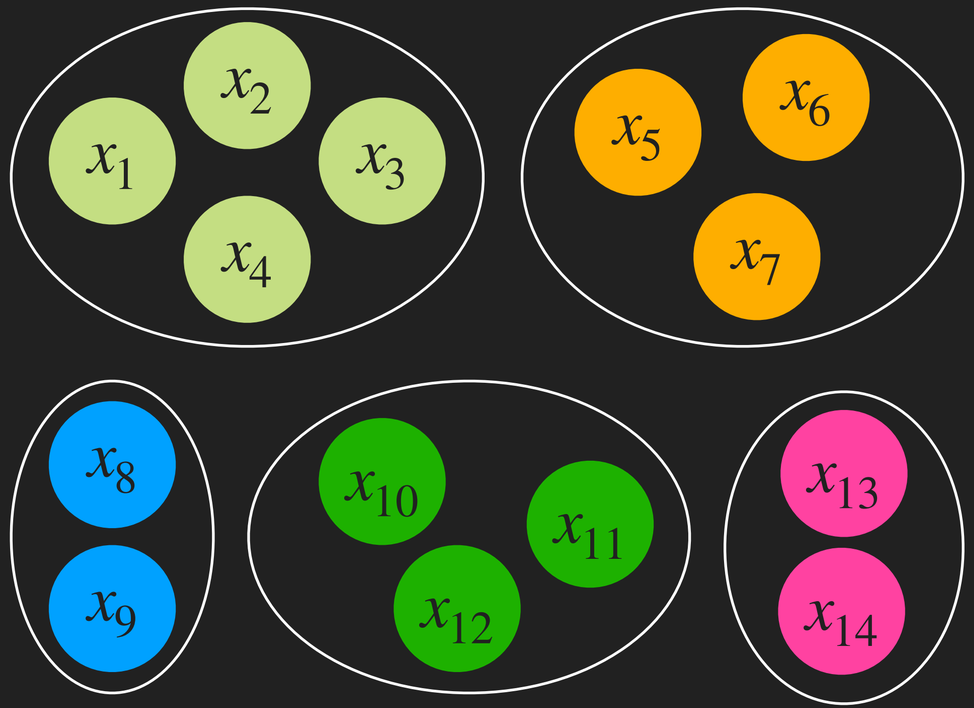
X — множество объектов

$\sim \subseteq X \times X$ — отношение

эквивалентность

- рефлексивность: $x \sim x$
- симметричность: $x \sim y \Rightarrow y \sim x$
- транзитивность: $(x \sim y, y \sim z) \Rightarrow x \sim z$

ADT Partition — Система Непересекающихся Множеств



1. проверить, принадлежат ли объекты x_1 и x_2 одному классу эквивалентности
2. найти/подсчитать объекты, эквивалентные объекту x
3. установить отношение между двумя объектами x_1 и x_2

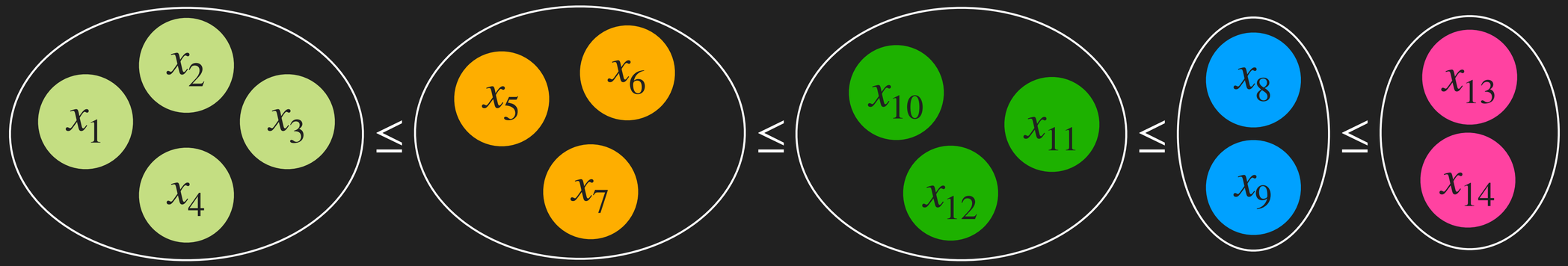
…

X — множество объектов

$\leq \subseteq X \times X$ — отношение

- рефлексивность: $x \leq x$
- антисимметричность: $(x \leq y, y \leq x) \Rightarrow x \sim y$
- транзитивность: $(x \leq y, y \leq z) \Rightarrow x \leq z$

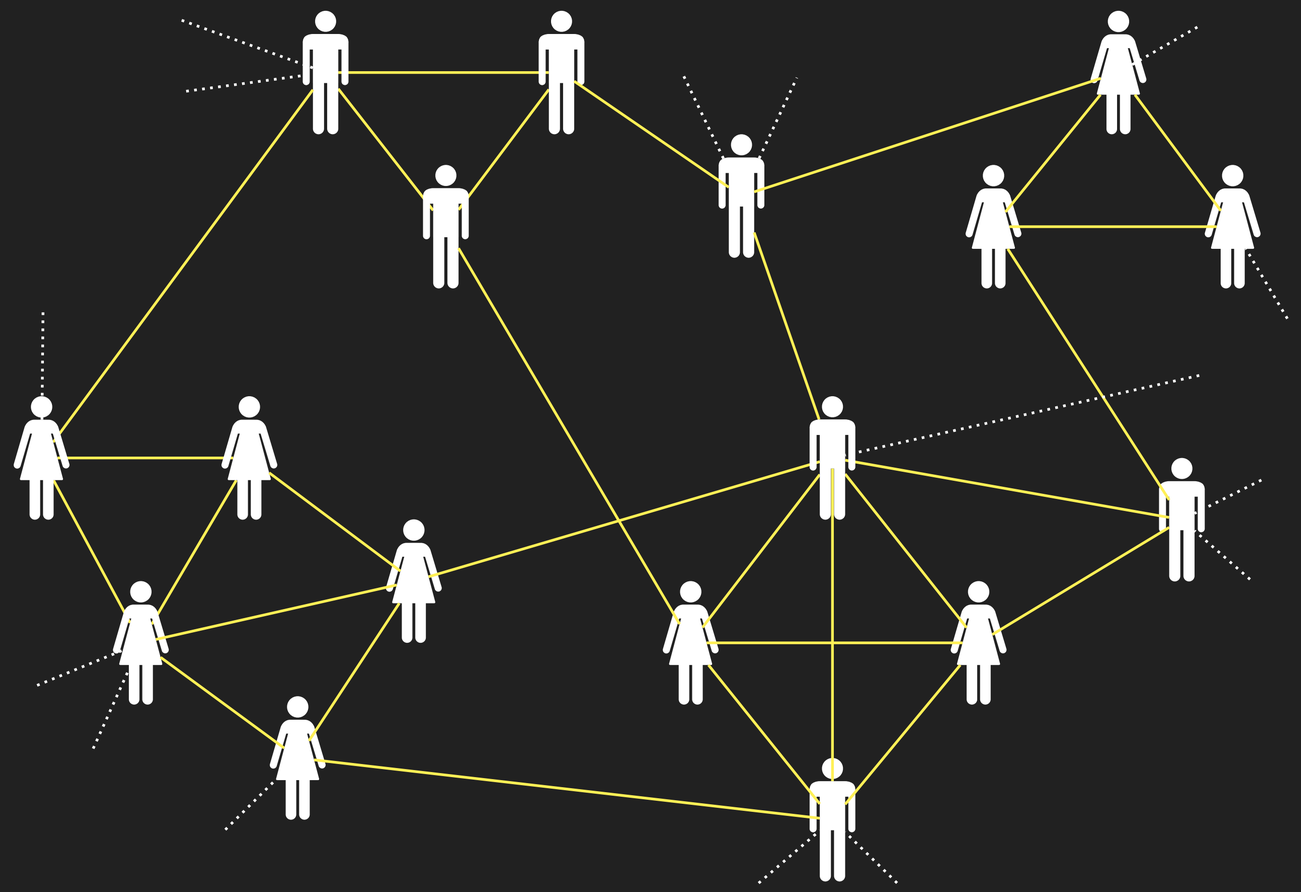
слабый порядок



слабый порядок лежит в основе реализации стандартных контейнеров `set`, `map`, `multiset`, `multumap`

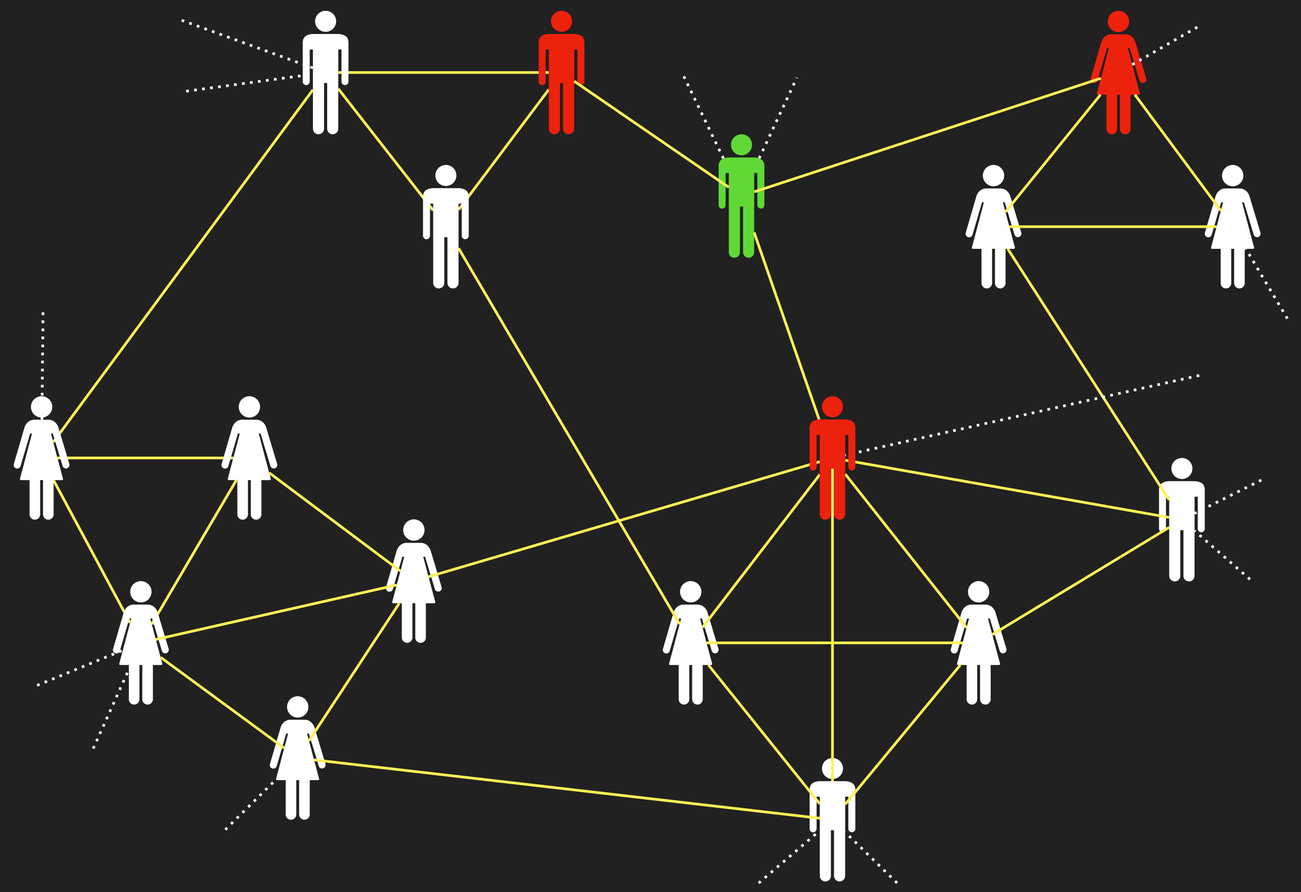
как упорядочиваются эквивалентные объекты — вопрос реализации

отношение смежности • $a \leftrightarrow b$



графовое представление произвольного бинарного отношения на множестве объектов

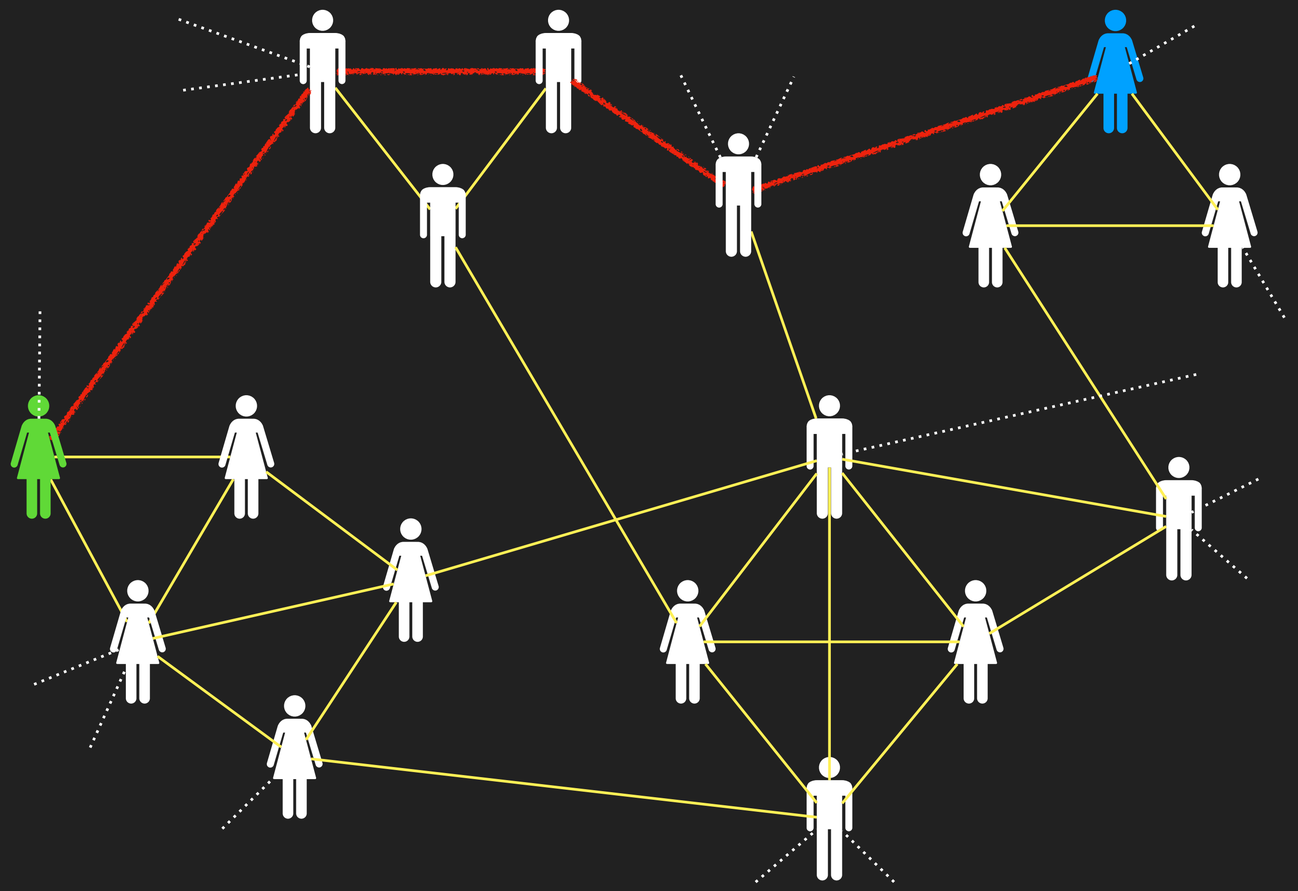
отношение смежности • $a \leftrightarrow b$



графовое представление произвольного бинарного отношения на множестве объектов

1. найти всех соседей для заданного объекта x

отношение смежности • $a \leftrightarrow b$



графовое представление произвольного бинарного отношения на множестве объектов

1. найти всех соседей для заданного объекта x
2. проверить достижимость вершины y из вершины x

…

как размещается основная структура данных в памяти?

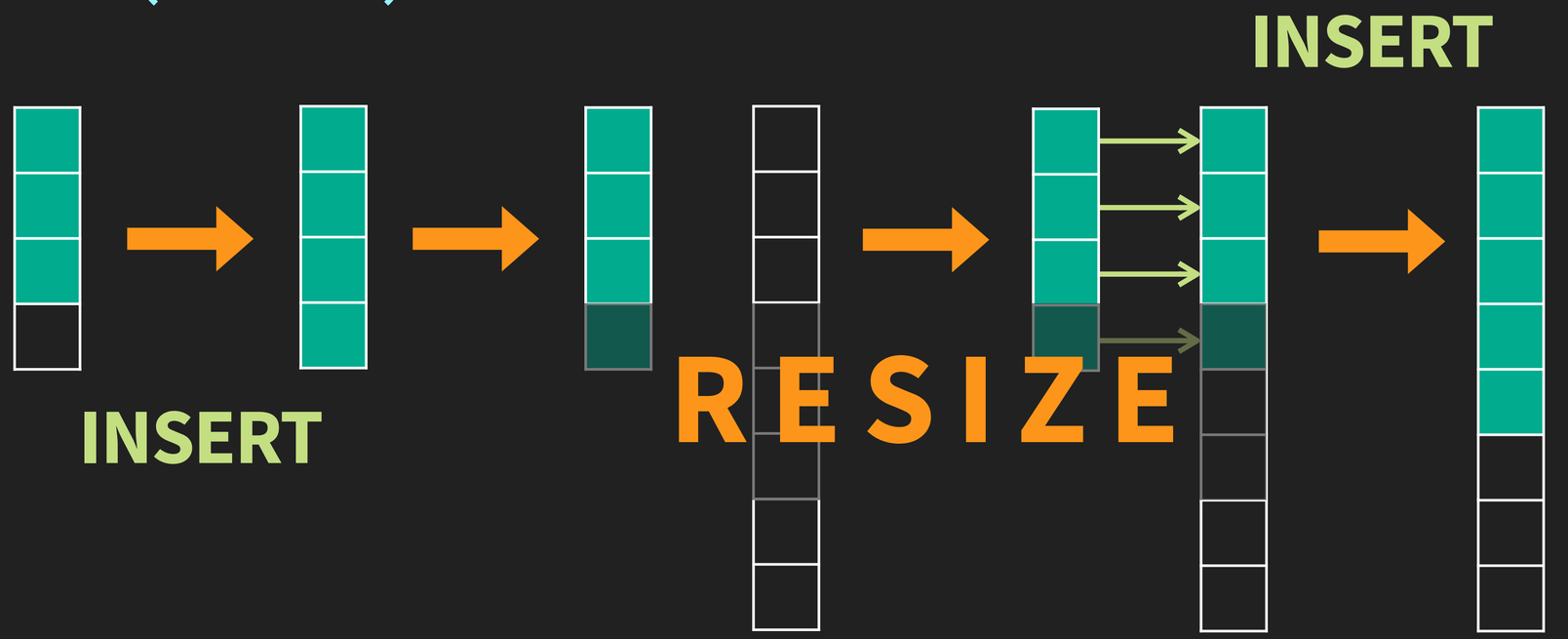
ABSTRACT_DATA_TYPE.cpp

```cpp
template<typename T>
class AbstractDataType {
public:
    AbstractDataType();
    ~AbstractDataType();

    // методы доступа
    // методы модификации

private:
    DataStructure<T> *storage;

    size_t count;
    size_t max_size;
    // ...
}
```

непрерывное [contiguous] размещение

1. фиксированные и динамические массивы
   `T[N]` • `std::array<T, N>` • `T*` • `std::vector<T>`

2. индексация — произвольный доступ
   `A[i] = *(A + i)`



связное [linked] размещение

NODE.cpp

```cpp
template<typename T>
class Node {
public:
    Node(const T& = T(), Node* = nullptr);
    T retrieve() const;
    Node *next() const;

private:
    T element;
    Node *next_node;
};
```

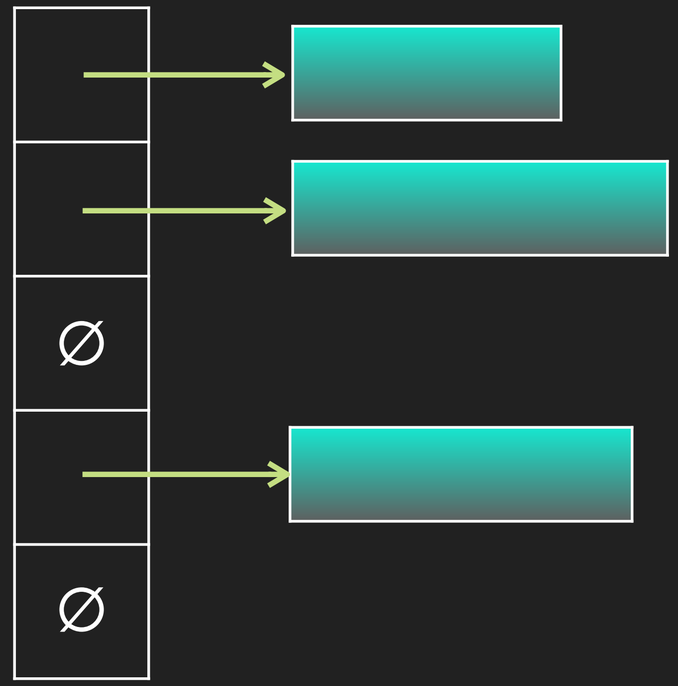
ADT_LIST.cpp

```cpp
template<typename T>
class List {
public:
    // конструкторы
    // методы доступа
    // методы модификации

private:
    Node<T> *head;
    Node<T> *tail;
};
```

индексированное [indexed] размещение

? массивы указателей на выделенные области памяти

`T**` • `std::list<T> *arr` • `...`

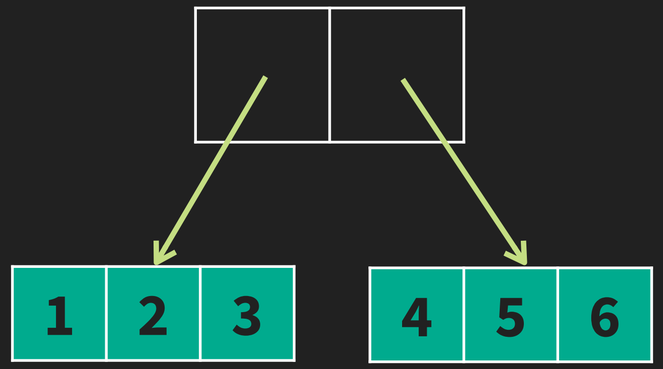


- многомерные **массивы**
- **графы** — списки смежности
- STL — `std::deque`

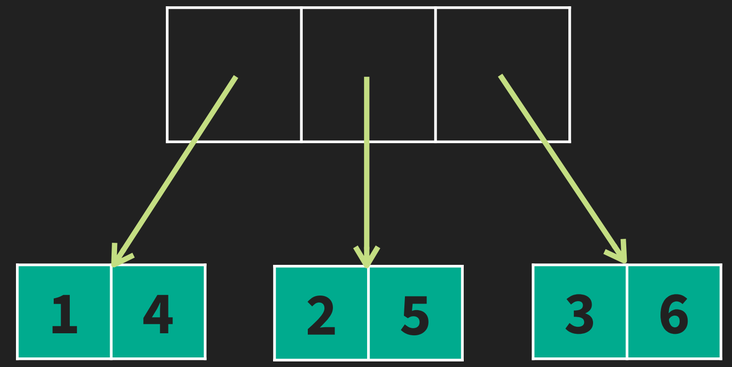
и другие…

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

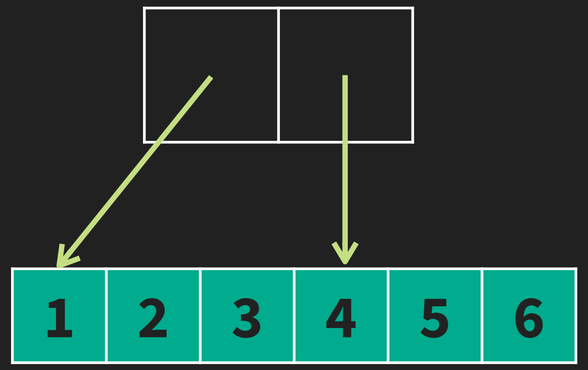
$A_{2\times3}$

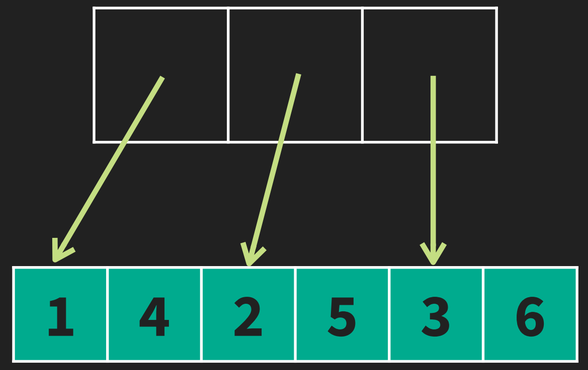


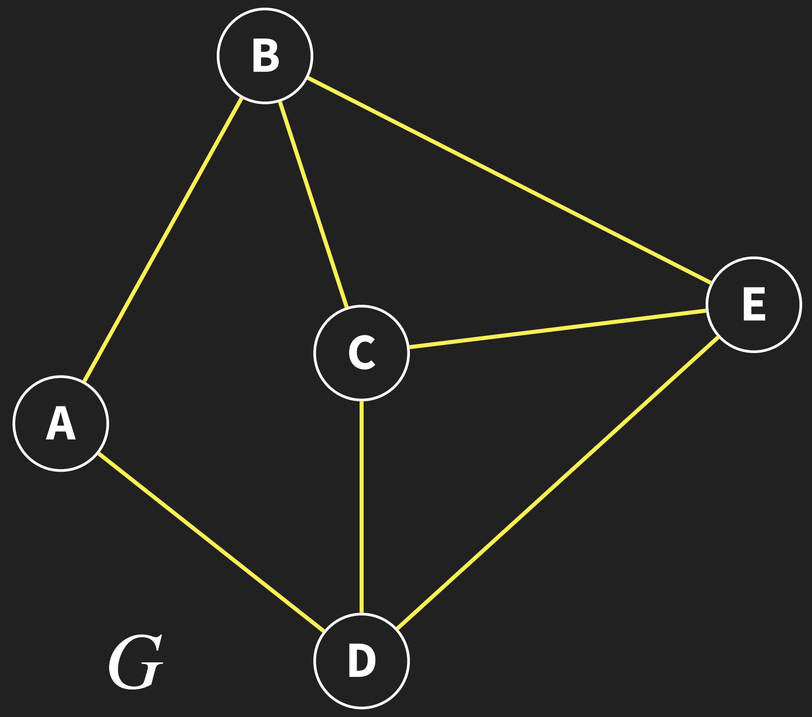
row–major order

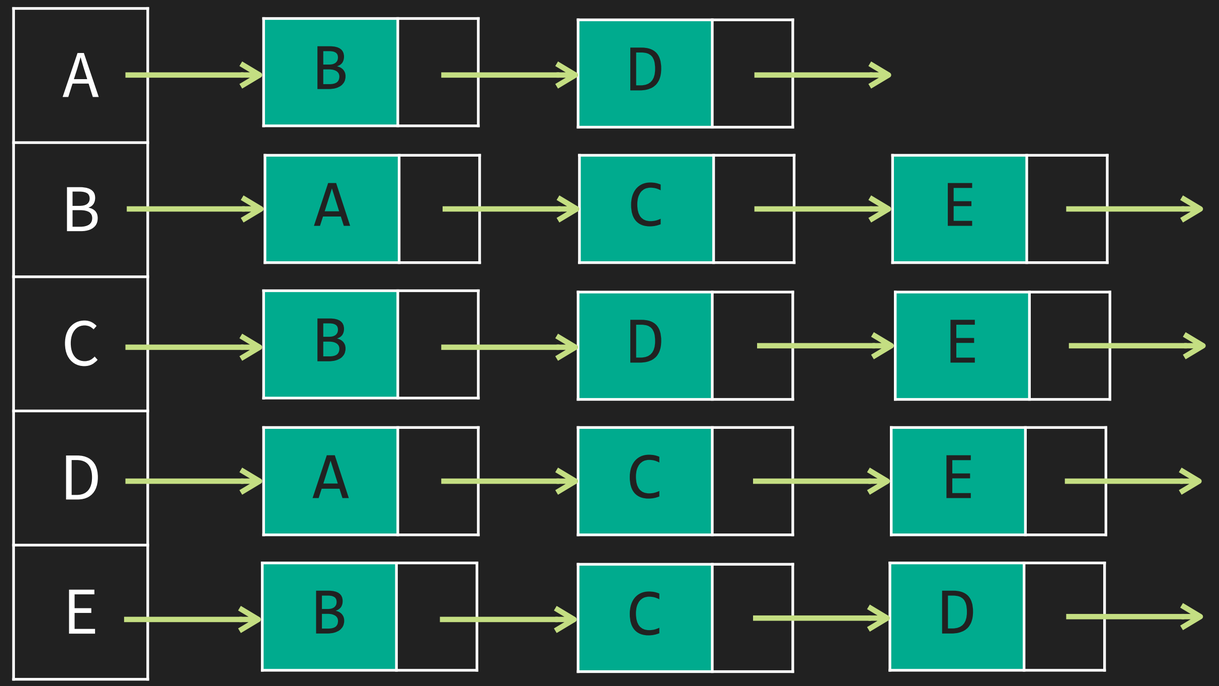


column–major order

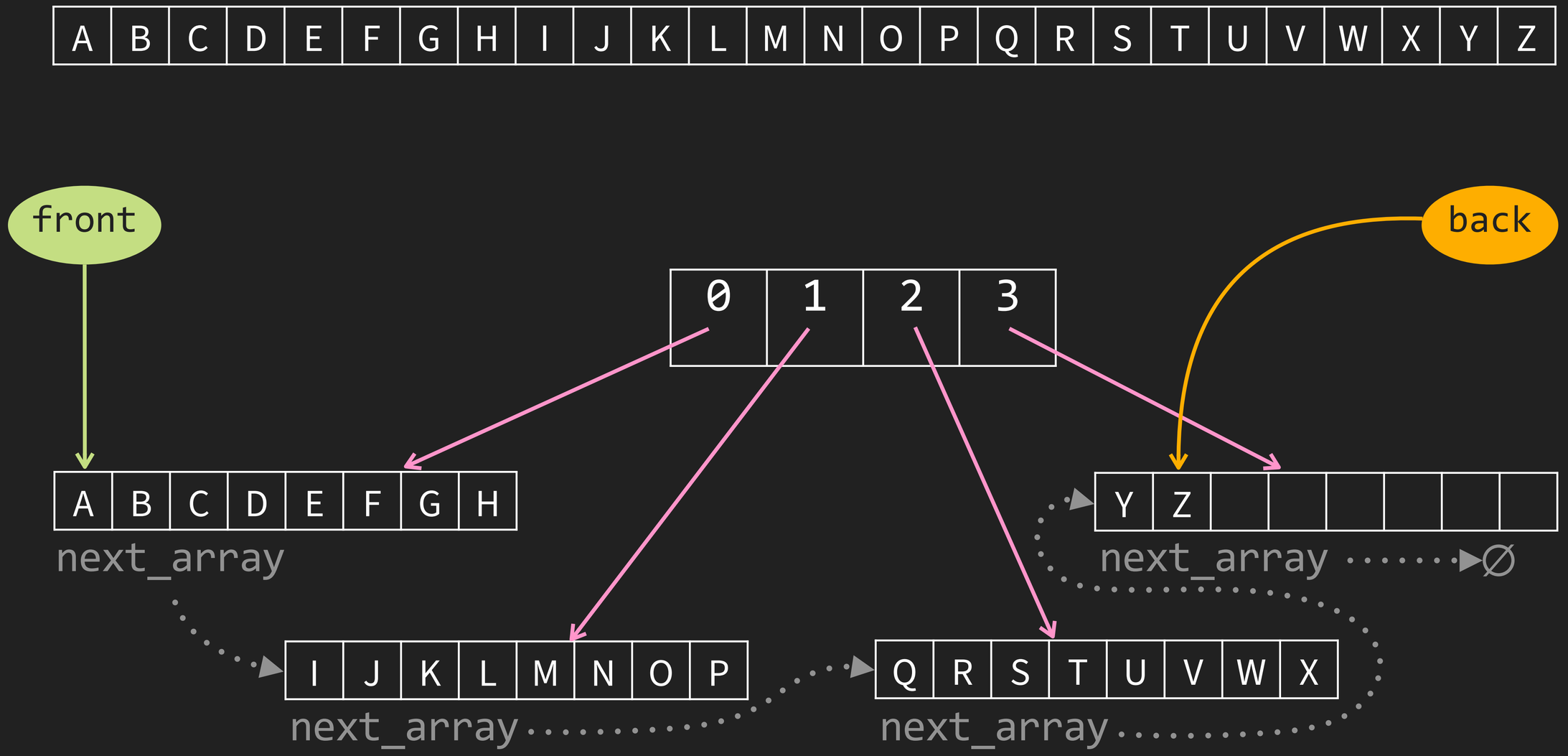








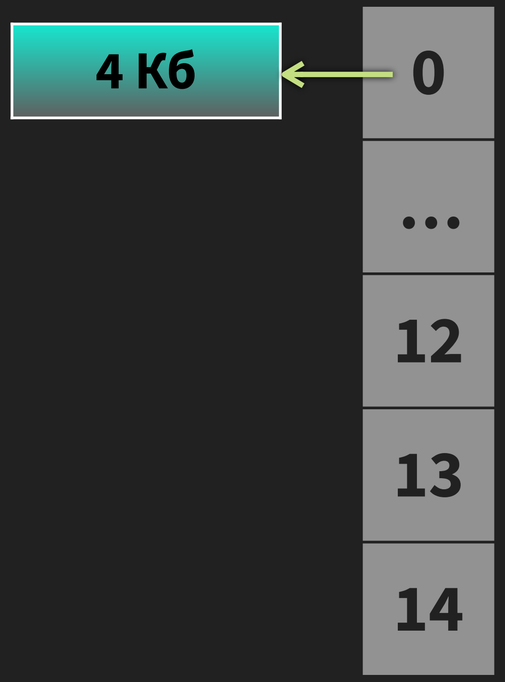

```
Node **adjList;
std::list<int> *adjList;
std::vector<int> adjList[N];
```


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
front
back
0 1 2 3
A B C D E F G H
next_array
Y Z
next_array
∅
I J K L M N O P
next_array
Q R S T U V W X
next_array

гибридное размещение • inode pointer structure

UNIX–структура данных для работы
с большими файлами

0 прямой уровень доступа
12 блоков по 4 Кб = 48 Кб

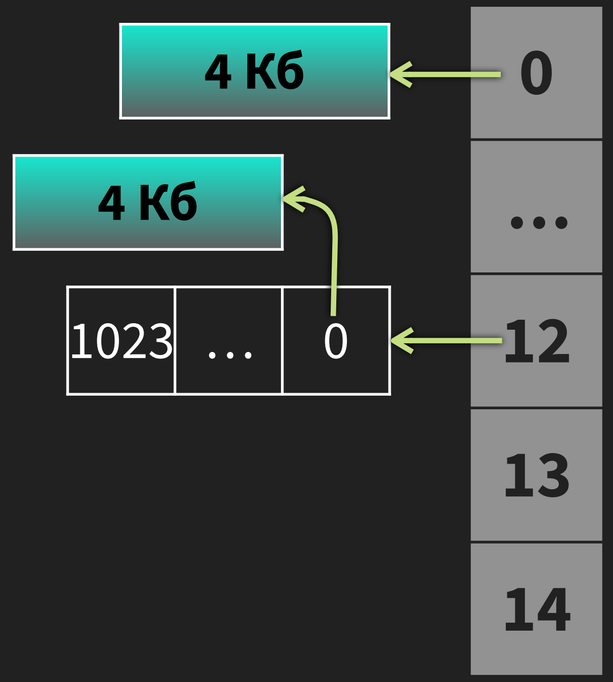


гибридное размещение • inode pointer structure

UNIX–структура данных для работы с большими файлами

0 прямой уровень доступа
12 блоков по 4 Кб = 48 Кб

1 первый уровень косвенности
1024 блоков по 4 Кб = 4 Мб



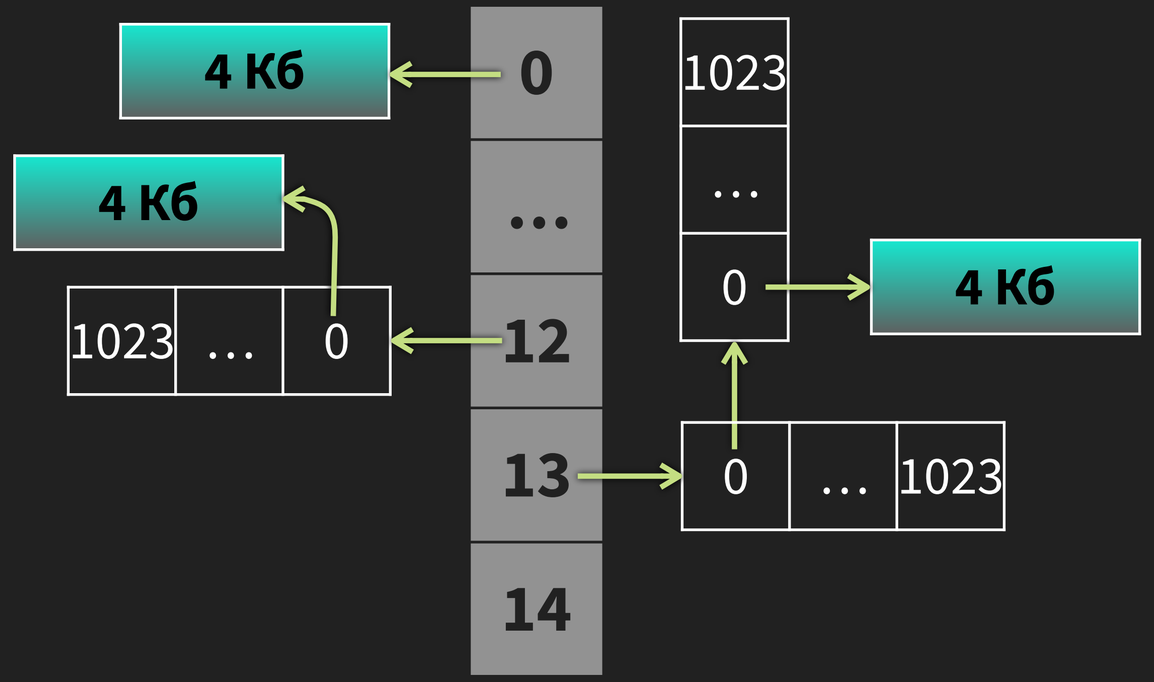
гибридное размещение • inode pointer structure

UNIX–структура данных для работы с большими файлами

0 прямой уровень доступа
12 блоков по 4 Кб = 48 Кб

1 первый уровень косвенности
1024 блоков по 4 Кб = 4 Мб

2 второй уровень косвенности
1024 блоков по 4 Мб = 4 Гб



гибридное размещение • inode pointer structure

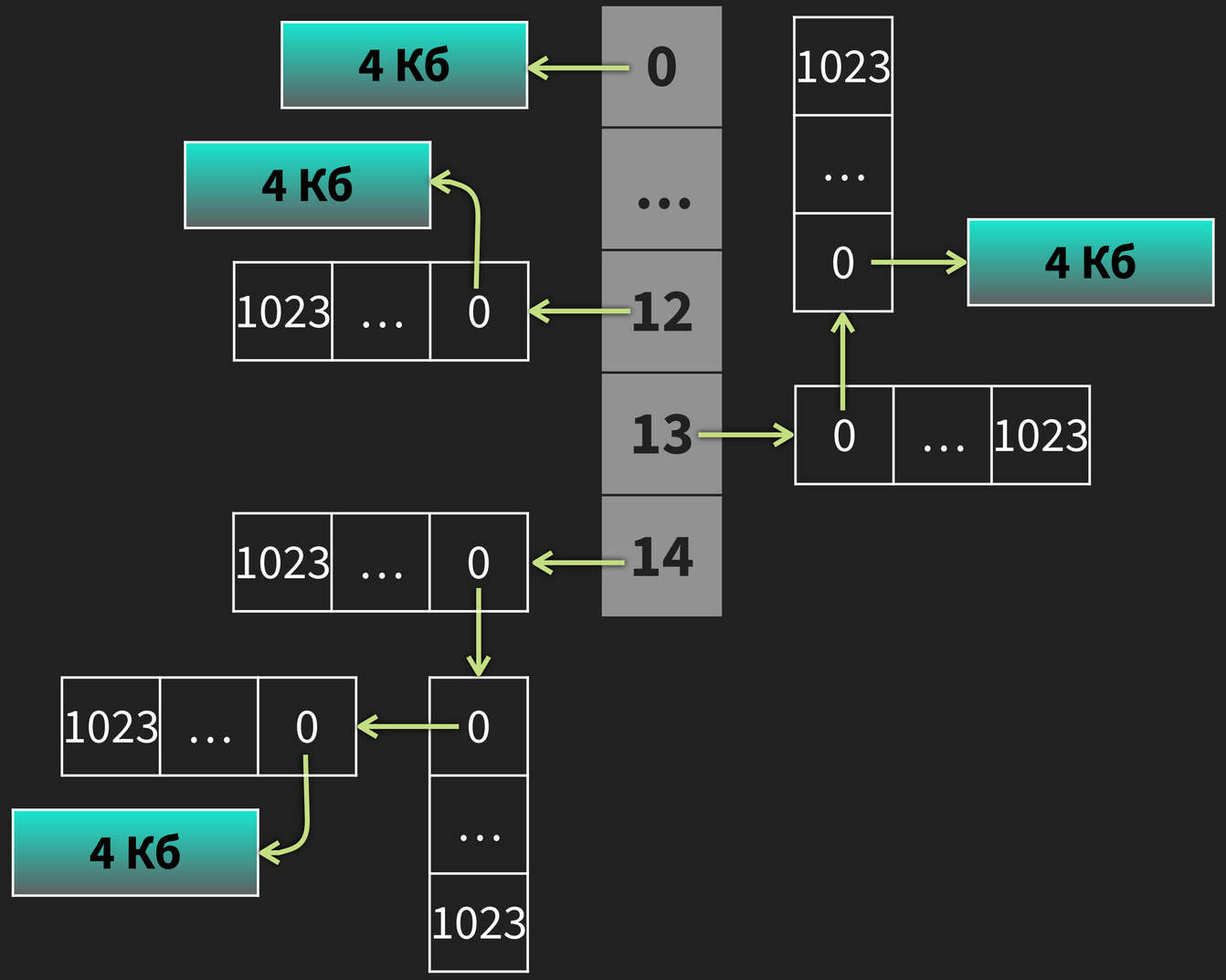
UNIX–структура данных для работы с большими файлами

0 прямой уровень доступа
12 блоков по 4 Кб = 48 Кб

1 первый уровень косвенности
1024 блоков по 4 Кб = 4 Мб

2 второй уровень косвенности
1024 блоков по 4 Мб = 4 Гб

3 третий уровень косвенности
1024 блоков по 4 Гб = 4 Тб



как размещается основная структура данных в памяти?

непрерывное
связное
индексное
гибридное
...

ABSTRACT_DATA_TYPE.cpp

```cpp
template<typename T>
class AbstractDataType {
public:
    AbstractDataType();
    ~AbstractDataType();

    // методы доступа
    // методы модификации

private:
    DataStructure<T> *storage;

    size_t count;
    size_t max_size;
    // ...
}
```

как оценить эффективность методов [алгоритмов]?

эффективность структуры данных

	начало	**произвольная позиция**	**конец**
поиск объекта	?	?	?
вставка объекта	?	?	?
удаление объекта	?	?	?

	начало	произвольная позиция	конец
поиск объекта	?	?	?
вставка объекта	?	?	?
удаление объекта	?	?	?

ОТСОРТИРОВАННЫЙ МАССИВ

	начало	произвольная позиция	конец
поиск объекта	GOOD	FAIR	GOOD
вставка* объекта	BAD	BAD	GOOD* / BAD
удаление объекта	BAD	BAD	GOOD

ОТСОРТИРОВАННЫЙ МАССИВ

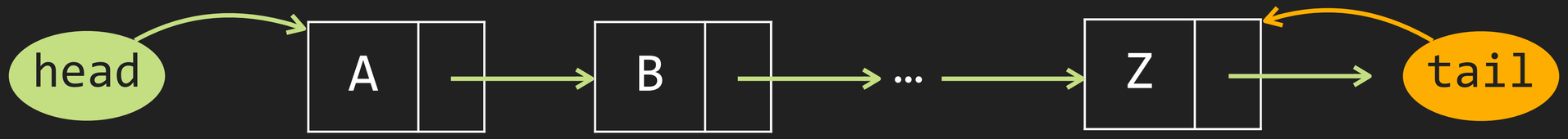
	начало	произвольная позиция	конец
поиск объекта	$\Theta(1)$	$O(\log N)$	$\Theta(1)$
вставка* объекта	$\Theta(N)$	$\Theta(N)$	$\Theta(1)$ *
удаление объекта	$\Theta(N)$	$\Theta(N)$	$\Theta(1)$

ОТСОРТИРОВАННЫЙ МАССИВ

R	E	G	S	I	O	P	B	G	T	Y	Y	C	V	S	A	S	K	L	C	Z	V	B	A	P	T

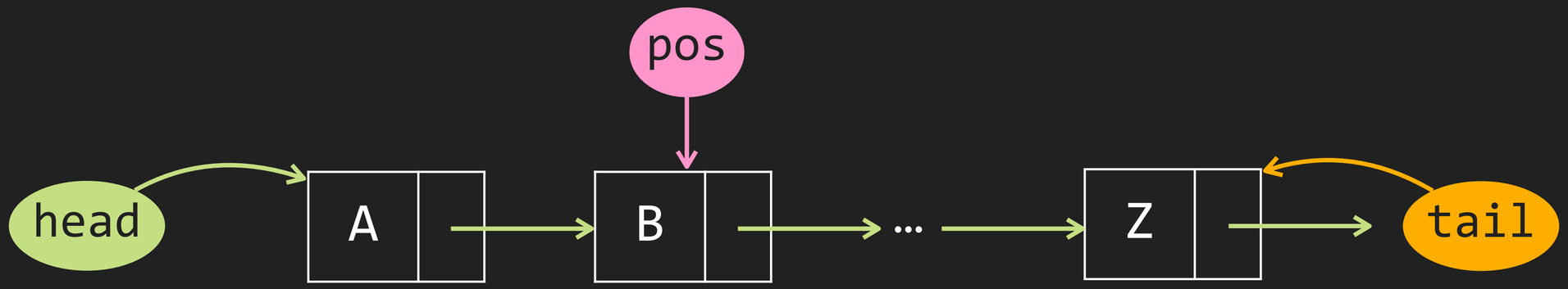
	начало	**произвольная позиция**	**конец**
поиск объекта	GOOD	BAD	GOOD
вставка объекта	BAD	BAD	GOOD* / BAD
удаление объекта	BAD	BAD	GOOD

НЕОТСОРТИРОВАННЫЙ МАССИВ



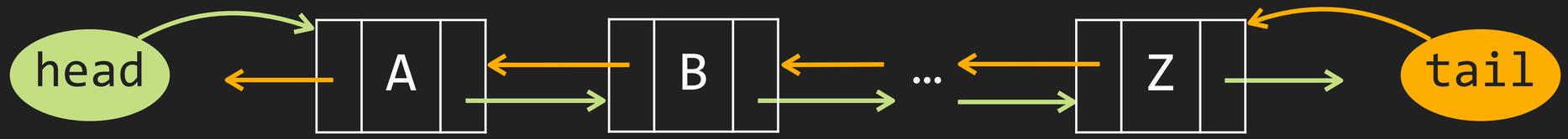
	начало	произвольная позиция	конец
поиск объекта	GOOD	BAD	GOOD
вставка объекта	GOOD	BAD	GOOD
удаление объекта	GOOD	BAD	BAD

ОДНОСВЯЗНЫЙ СПИСОК



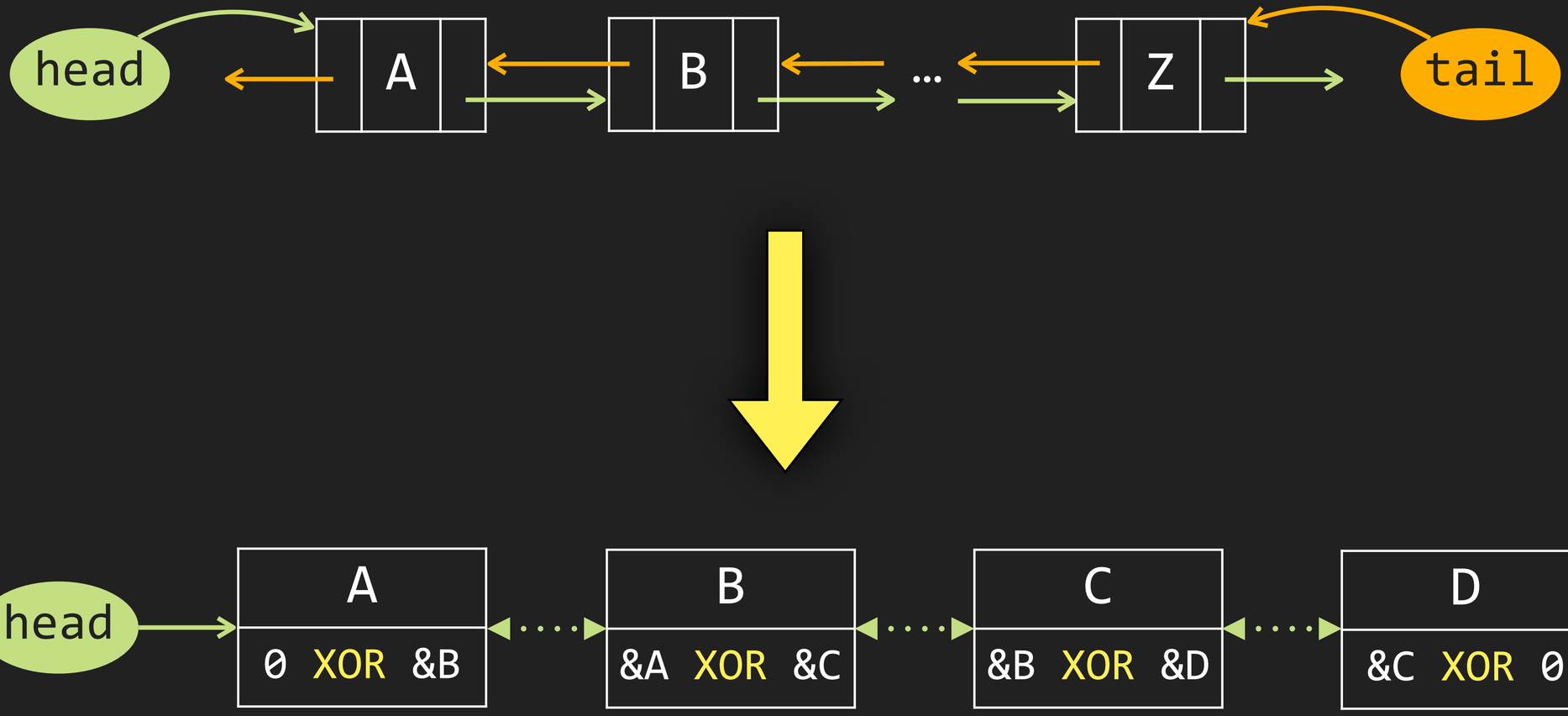
	начало	произвольная позиция	конец
поиск объекта	GOOD	BAD	GOOD
вставка объекта	GOOD	GOOD	GOOD
удаление объекта	GOOD	GOOD	BAD

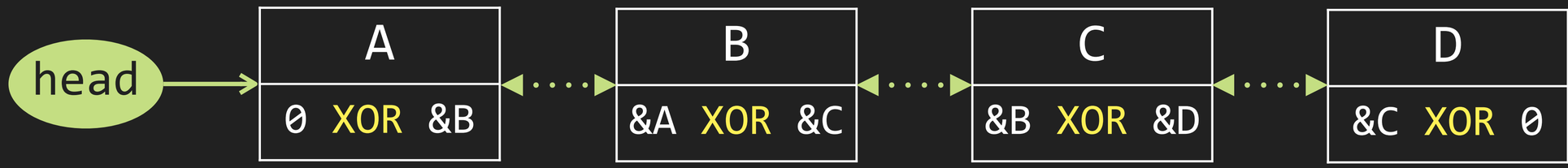
ОДНОСВЯЗНЫЙ СПИСОК



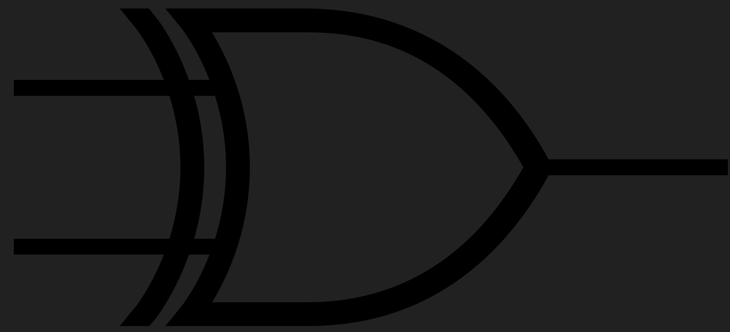
	начало	произвольная позиция	конец
поиск объекта	GOOD	BAD	GOOD
вставка объекта	GOOD	GOOD	GOOD
удаление объекта	GOOD	GOOD	GOOD

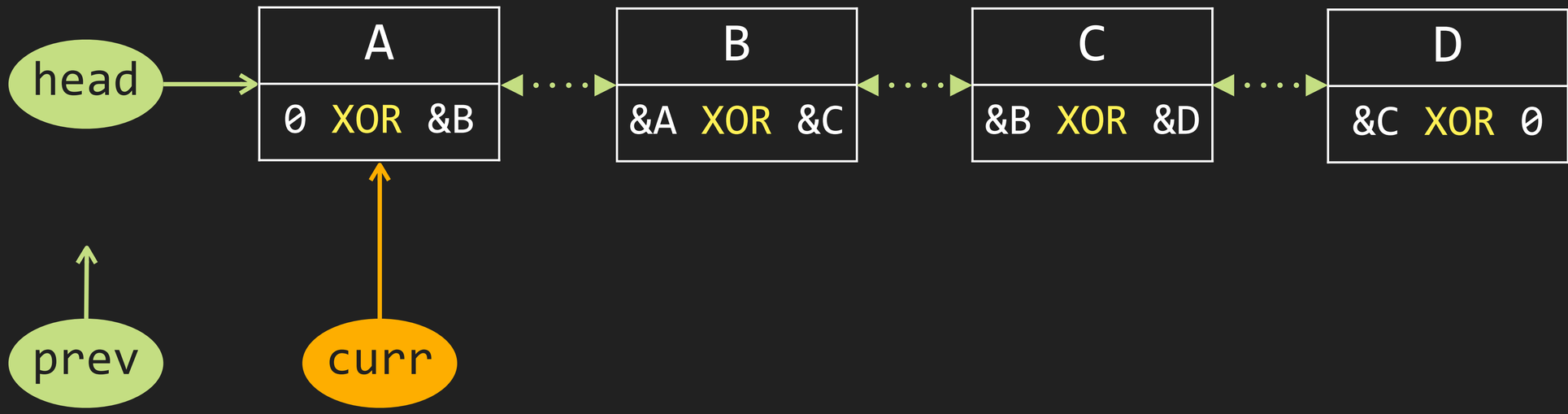
ДВУСВЯЗНЫЙ СПИСОК


head
A
B
…
Z
tail
head
A
0 XOR &B
B
&A XOR &C
C
&B XOR &D
D
&C XOR 0

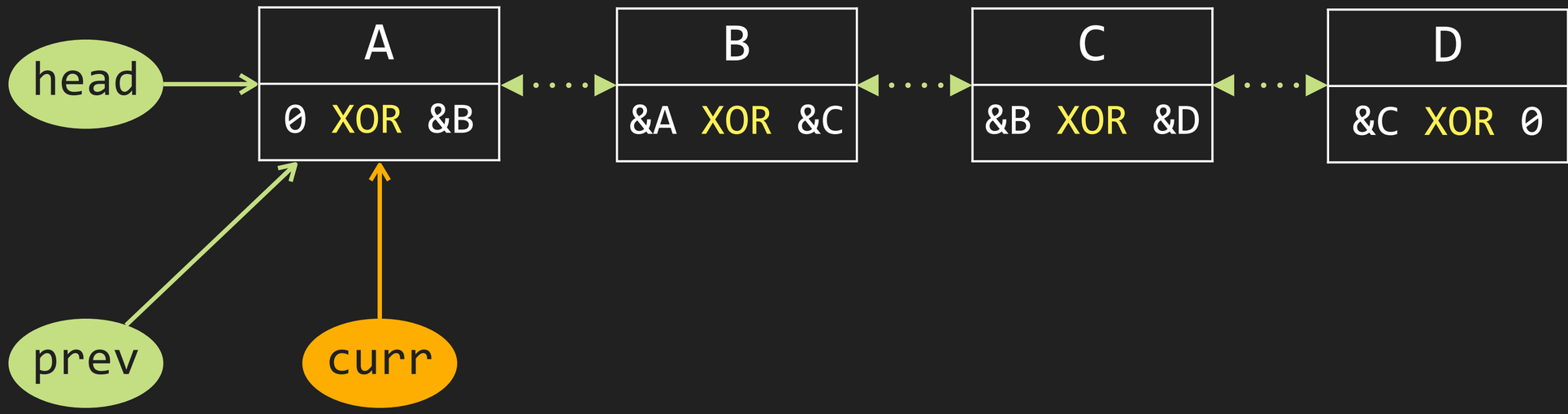


A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

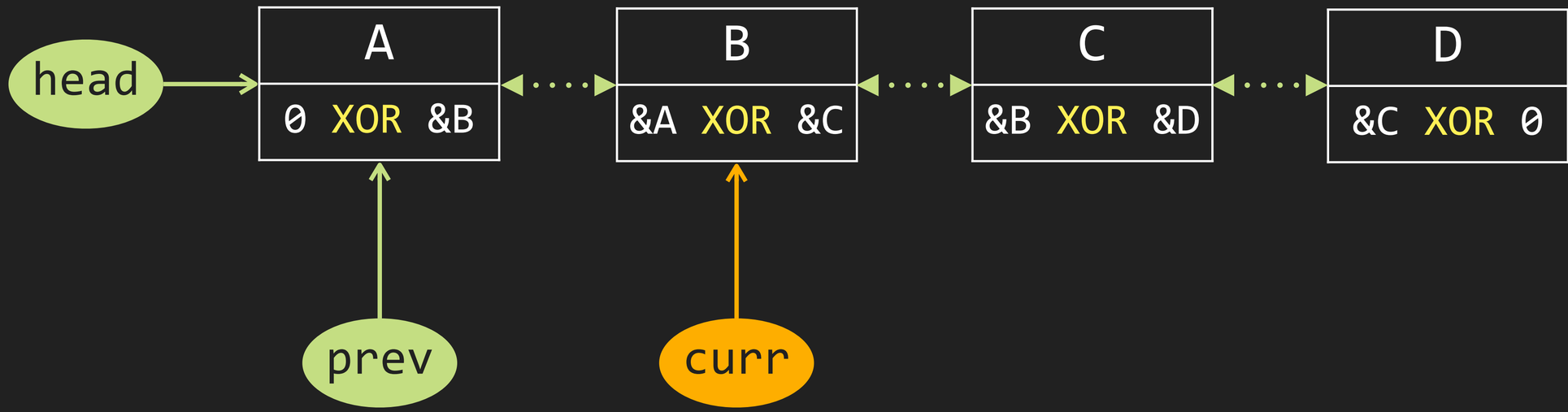




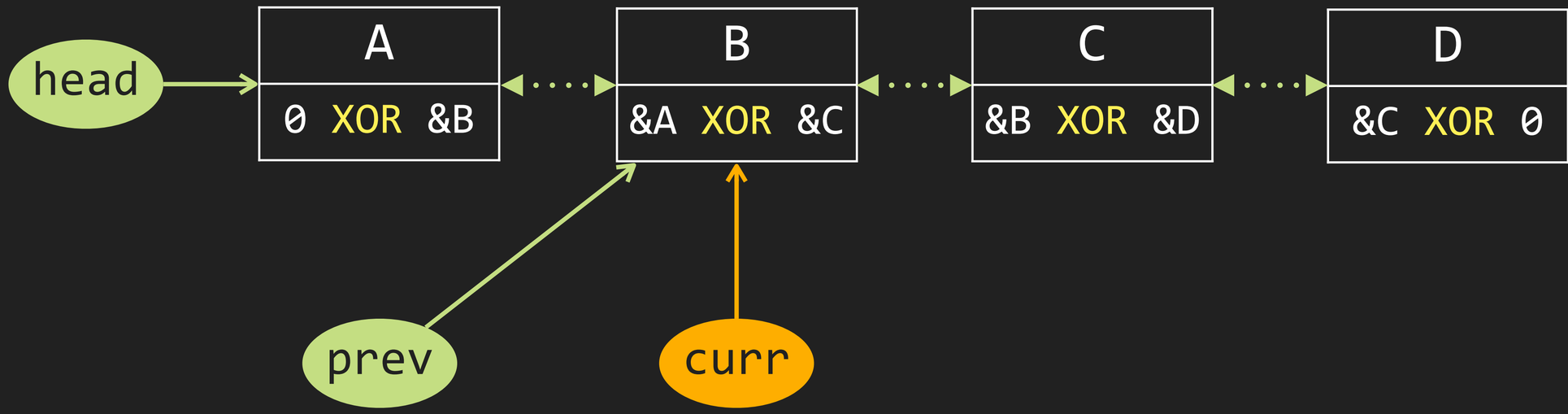

```
curr = head

prev = nullptr
```



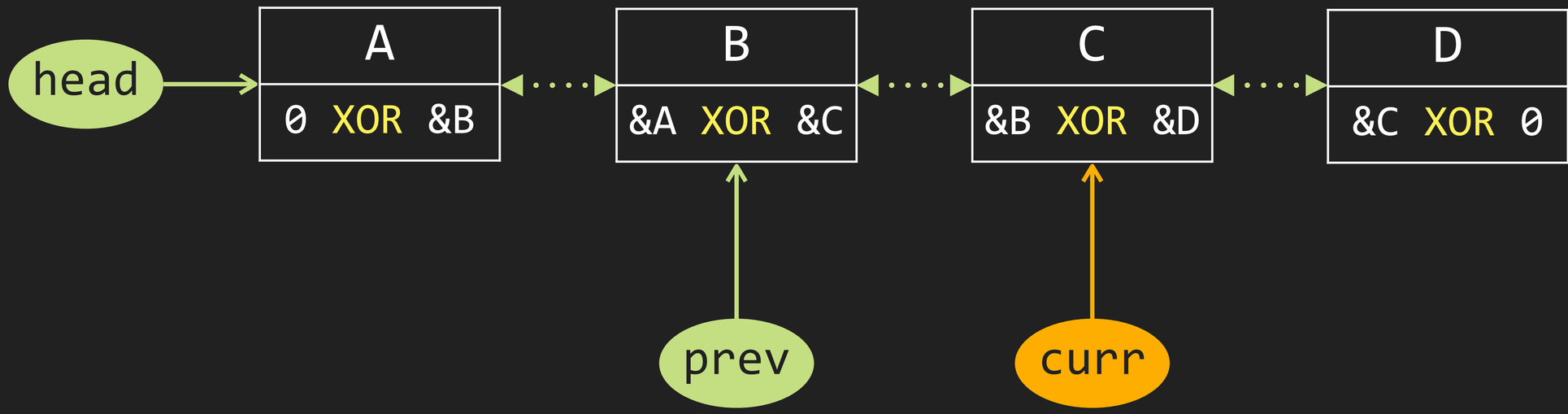

```
prev = curr

curr = prev XOR (curr->link)
     = 0 XOR (0 XOR &B)
     = &B
```




```
prev = curr

curr = prev XOR (curr->link)
     = 0 XOR (0 XOR &B)
     = &B
```




```
prev = curr

curr = prev XOR (curr->link)
     = &A XOR (&A XOR &C)
     = &C
```




```
prev = curr

curr = prev XOR (curr->link)
     = &A XOR (&A XOR &C)
     = &C
```

NODE.cpp

```cpp
struct Node {
    int data;
    Node *link;

    Node(int val) {
        data = val;
        link = nullptr;
    }
};
```

XORLinkedList.cpp

```cpp
class XORLinkedList {
public:
    XORLinkedList() : head(nullptr) {}

    void insert(int val);
    void traverse();
    // ...
private:
    Node *head;

    Node *XOR(Node *a, Node *b) {
        return reinterpret_cast<Node *>(
            reinterpret_cast<uintptr_t>(a) ^
            reinterpret_cast<uintptr_t>(b)
        )
    }
}
```

XORLinkedList.cpp

```cpp
class XORLinkedList {
public:
    XORLinkedList() : head(nullptr) {}

    void insert(int val);
    void traverse();
    // ...
private:
    Node *head;

    Node *XOR(Node *a, Node *b) {
        return reinterpret_cast<Node *>(
            reinterpret_cast<uintptr_t>(a) ^
            reinterpret_cast<uintptr_t>(b)
        )
    }
}
```

приведение несовместимых типов `reinterpret_cast<…>`

архитектурно-зависимый тип для приведения — `uintptr_t`

1. сокращение затрат по памяти — 1 указатель на узел
2. поддержка проходов в обоих направлениях
2. небезопасная реализация — SEGV, SEGFAULT и прочее…

резюме

1. абстрактный тип данных — модель хранения, доступа и модификации объектов
2. классификация структур данных по расположению в памяти и оценка их эффективности

тизер следующей лекции

1. асимптотический анализ временной сложности алгоритма: символы Ландау O, Ω, Θ
2. оценка временной сложности рекурсивного алгоритма: SELECTION SORT, MERGE SORT