Алгоритми та структури даних

Питання до екзамену

1. **Основні визначення згідно Міжнародних термінологічних стандартів (ISO 2382-1:1993, IEEE Std 829-2008): програмне забезпечення, програма, дані, структура.**

**ПЗ** - все или часть программ, процедур, правил и соответствующей документации системы обработки информации (ISO/IEC 2382-1:1993)

**Программа** - синтаксическая единица, которая соответствует правилам определённого языка программирования, состоящая из определений и операторов или инструкций, необходимых для определённой функции, задачи или решения проблемы

**Данные** - представление информации в формализованном виде, пригодном для передачи, связи или обработки

**Структура** – отношение между элементами системы

1. **Сутність поняття «структура даних».**

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

Структуры данных формируются с помощью типов данных, ссылок и операций над ними в выбранном языке программирования.

1. **Класифікація структур даних за рівнем складності.**

Различают простые (базовые, примитивные) структуры (типы) данных и интегрированные (структурированные, композитные, сложные). Простыми называются структуры данных, которые не могут быть разделены на составные части, большие, чем биты. С точки зрения физической структуры важным является то обстоятельство, что в данной машинной архитектуре, в данной системе программирования всегда можно сказать, каков будет размер данного простого типа и какова структура его размещения в памяти. С логической точки зрения простые данные являются неделимыми единицами.

Интегрированными называются структуры данных, составными частями которых являются другие структуры данных – простые или в свою очередь интегрированные. Интегрированные структуры данных конструируются с использованием средств интеграции данных, предоставляемых алгоритмическими языками.

1. **Класифікація структур даних за наявністю зав’язків між елементами структури.**

В зависимости от отсутствия или наличия явно заданных связей между элементами данных различают несвязные структуры (векторы, массивы, строки, стеки, очереди) и связные структуры (связные списки).

1. **Класифікація структур даних за мінливістю. Загальна характеристика.**

По изменчивости: статические, полустатические, динамические.

Изменчивость, то есть изменение числа элементов и (ли) связей между элементами структуры.

**Статические** - к этой группе относят массивы, множества, записи, таблицы.

**Полустатические** - это стеки, очереди, деки, дерева.

**Динамические** - линейные и разветвленные связные списки, графы, деревья.



**/\***

**Запись (декартово произведение)** — это совокупность элементов данных разного типа. В простейшем случае запись содержит постоянное число элементов, которые называют полями. Совокупность записей одинаковой структуры называется файлом.

**Множество** — это такая структура, которая представляет собой набор неповторяющихся данных одного и того же типа (множество есть совокупность различных элементов, мыслимая как единое целое).

**Таблица** — это способ передачи содержания, заключающийся в организации СД, в которой отдельные элементы помещены в ячейки, каждой из которых сопоставлена пара значений — номер строки и номер колонки (столбца). Таким образом, устанавливается смысловая связь между элементами, принадлежащими одному столбцу или одной строке.

**Списком** называют упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения (письменный перечень, число, состав; документ, содержащий перечень каких-либо сведений). Список, отражающий отношения соседства между элементами, называют линейным.

\*/

1. **Класифікація структур даних за мінливістю. Статичні структури. Приклади використання.**

**Статические** - к этой группе относят массивы, множества, записи, таблицы.

1. **Класифікація структур даних за мінливістю. Полустатичні структури. Приклади використання.**

**Полустатические** - это стеки, очереди, деки, дерева.

1. **Класифікація структур даних за мінливістю. Динамічні структури. Приклади використання.**

**Динамические** - линейные и разветвленные связные списки, графы, деревья.

1. **Класифікація структур даних за характером впорядкованості її елементів.**

**Лінійні структури** залежно від характеру взаємного розташування елементів пам'яті поділяють на структури з послідовним розподілом елементів пам'яті (вектори, рядки, масиви, стеки, черги) і структури з довільним зв'язковим розподілом елементів пам'яті (однозв'язні і двусвязные лінійні списки).

**Нелінійні структури** - багатозв'язані списки, дерева, графи.

1. **Класифікація структур даних за розміщенням в пам’яті ЕОМ.**

**Структури даних для оперативної пам'яті** - це дані, розміщені в статичної та динамічної пам'яті комп'ютера. Всі вищенаведені структури даних - це структури для оперативної пам'яті.

**Структури даних для зовнішньої пам'яті** називають файловими структурами або файлами. Прикладами файлових структур є послідовні файли, файли, організовані розділами, - дерева.

1. **Характеристика структури – зв’язані списки. Приклади задач, у яких використовується ця структура.**

**Хэш-табли́ца или хеш-табли́ца** — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

/\*

LinkedList нужен там, где с одной стороны нужны листы. А с другой — быстрые вставки в середину. Примеров не много, но вот два:

1) Нам периодически приходят обновления о том, что объект из списка пропал. И происходит это часто, а объектов много. Например нам сообщают, что машина продана.

2) Удобно составлять цепи событий, куда можно вставлять промежуточные события.

\*/

**Здесь приведено простейшее консольное приложение, предоставляющее интерфейс телефонной книжки. Оно реализовано на основе контейнера map.**

#include *<iostream>*

#include *<string>*

#include *<map>*

**using** **namespace** std;

int main()

{

string cmd, name, phone;

map< string, string > book;

**while**( cin >> cmd )

{

**if** ( cmd == "add" )

{

cin >> name >> phone;

book[ name ] = phone;

cout << "Added" << endl;

}

**else** **if** ( cmd == "find" )

{

cin >> name;

**try** {

string v = book.at( name );

cout << name << "'s phone is " << v << endl;

}

**catch** (**const** out\_of\_range& e) {

cout << name << " not found" << endl;

}

}

**else** **if** ( cmd == "del" )

{

cin >> name;

book.erase( name );

cout << "Deleted" << endl;

}

**else** **if** ( cmd == "view" )

{

**for**( **auto**& kv : book )

cout << kv.first << "**\t** " << kv.second << endl;

}

**else** **if** ( cmd == "quit" )

**return** 0;

**else**

cerr << "Bad command '" << cmd << "'" << endl;

}

**return** 0;

}

1. **Характеристика структури – вектор. Приклади задач, у яких використовується ця структура.**

**Вектор** - реализация динамического массива переменного размера.

**Вектор (одномерный массив)** — это структура данных с фиксированным числом элементов одного и того же типа (вектор в геометрии (связанный вектор) — упорядоченная пара точек, одна из которых называется началом, другая — концом вектора).

**Пример одномерного динамического вектора, сортировка и удаление дубликатов:**

#include *<vector>*

#include *<string>*

#include *<algorithm> // Для использования алгоритмов std::sort, std::unique*

int main()

{

std::vector< std::string > v\_str; *//Пустой вектор v\_str*

v\_str.push\_back("zz"); *// {"zz"}*

v\_str.push\_back("aa"); *// {"zz", "aa"}*

v\_str.push\_back("bb"); *// {"zz", "aa", "bb"}*

v\_str.push\_back("aa"); *// {"zz", "aa", "bb", "aa"}*

v\_str.push\_back("xx"); *// {"zz", "aa", "bb", "aa", "xx"}*

v\_str.push\_back("dd"); *// {"zz", "aa", "bb", "aa", "xx", "dd"}*

v\_str.push\_back("xx"); *// {"zz", "aa", "bb", "aa", "xx", "dd", "xx"}*

*//Сортировка всех элементов вектора*

std::sort(v\_str.begin(), v\_str.end());

*//Результат сортировки вектора: {"aa", "aa", "bb", "dd", "xx", "xx", "zz"}*

*// Удаление дубликатов*

v\_str.erase( std::unique(v\_str.begin(), v\_str.end() ), v\_str.end() );

*//Результат удаления дубликатов: {"aa","bb","dd","xx","zz"}*

**return** 0;

}

1. **Характеристика структури – масив. Приклади задач, у яких використовується ця структура.**

**Массив (функция с конечной областью определения)** — это простая совокупность элементов данных одного типа, средство оперирования группой данных одного типа. Отдельный элемент массива задается индексом. Массив может быть одномерным, двумерным и т.д. Разновидностями одномерных массивов переменной длины являются структуры типа кольцо, стек, очередь и двухсторонняя очередь (дек).

#include "stdafx.h"

#include <iostream>

using namespace std;

int main(int argc, char\* argv[])

{

    cout << "obrabotka massiva" << endl;

    int array1[16] = { 5, -12, -12, 9, 10, 0, -9,

                    -12, -1, 23, 65, 64, 11, 43, 39, -15 }; // объявление и инициализация одномерного массива

    cout << "indeks" << "\t\t" << "element massiva" << endl; // печать заголовков

    for (int counter = 0; counter < 16; counter++)  //начало цикла

    {

    //вывод на экран индекса ячейки массива, а затем содержимого этой ячейки, в нашем случае - это целое число

     cout << "array1[" << counter << "]" << "\t\t" << array1[counter] << endl;

    }

    system("pause");

    return 0;

}

1. **Характеристика структури – запис (в мові С++ це структура). Приклади задач, у яких використовується ця структура.**

Запись (декартово произведение) — это совокупность элементов данных разного типа. В простейшем случае запись содержит постоянное число элементов, которые называют полями. Совокупность записей одинаковой структуры называется файлом. (Файлом называют также набор данных во внешней памяти, например на магнитном диске.) Для того чтобы иметь возможность извлекать из файла отдельные записи, каждой записи присваивают уникальное имя или номер, которое служит ее идентификатором и располагается в отдельном поле. Этот идентификатор называют ключом.

Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры. Существуют два основных типа файлов: текстовые и двоичные.

Условно можно выделить два типа операций с файлом — связанные с его открытием и выполняющиеся без его открытия. Операции первого типа обычно служат для чтения и записи информации или подготовки к чтению или записи. Операции второго типа выполняются с файлом как с «объектом» файловой системы, в котором файл является наименьшим элементом структурирования.

**Операции, связанные с открытием файла**[[править](https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB&veaction=edit&section=12) | [править код](https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB&action=edit&section=12)]

В зависимости от операционной системы те или иные операции могут отсутствовать.

Обычно выделяют дополнительные сущности, связанные с работой с файлом:

* [Дескриптор](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B4%D0%B5%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%80) файла (хэндлер, описатель). При *открытии файла* (в случае, если это возможно), операционная система возвращает число (или указатель на структуру), с помощью которого выполняются все остальные файловые операции. По их завершении файл *закрывается*, а хэндлер теряет смысл.
* Файловый указатель. Число, являющееся смещением относительно нулевого байта в файле. Обычно по этому адресу осуществляется чтение/запись, в случае, если вызов операции чтения или записи не предусматривает указание адреса. При выполнении операций чтения или записи файловый указатель увеличивается на число прочитанных или записанных байт. Последовательный вызов операций чтения таким образом позволяет прочитать весь файл последовательно, не заботясь о позиционировании.
* Файловый буфер. Операционная система или библиотека языка программирования осуществляет [кэширование](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) файловых операций в специальном буфере (участке памяти). При закрытии файла буфер сбрасывается.
* Режим доступа. В зависимости от потребностей программы, файл может быть открыт на чтение или запись. Кроме того, некоторые операционные системы и библиотеки предусматривают режим работы с текстовыми файлами. Режим обычно указывается при открытии файла.
* Режим общего доступа. В случае многозадачной операционной системы возможна ситуация, когда несколько программ одновременно хотят открыть файл на чтение или запись. Для регулирования этого существуют режимы общего доступа, указывающие на возможность осуществления совместного доступа к файлу (например, файл, в который производится запись, может быть открыт для чтения другими программами — это стандартный режим работы с [файлами регистрации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB_%D1%80%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%86%D0%B8%D0%B8)).

**Операции**

* Открытие файла (обычно в качестве параметров передается имя файла, режим доступа и режим совместного доступа, а в качестве результата выступает файловый дескриптор), кроме того обычно имеется возможность в случае открытия на запись указать на то, должен ли размер файла изменяться на нулевой.
* Закрытие файла. В качестве аргумента выступает значение, полученное при открытии файла. При закрытии все файловые буферы сбрасываются.
* Запись — в файл помещаются данные.
* Чтение — данные из файла помещаются в область памяти.
* Перемещение указателя — указатель перемещается на указанное число байт вперёд или назад или перемещается по указанному смещению относительно начала или конца. Не все файлы позволяют выполнение этой операции (например, файл на ленточном накопителе может не «уметь» перематываться назад).
* Сброс буферов — содержимое файловых буферов с не записанной в файл информацией записывается. Используется обычно для указания на завершение записи логического блока (для сохранения данных в файле на случай сбоя).
* Получение текущего значения файлового указателя.

**Операции, не связанные с открытием файла**[[править](https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB&veaction=edit&section=13) | [править код](https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB&action=edit&section=13)]

Операции, не требующие открытия файла, оперируют с его «внешними» признаками — размером, именем, положением в [дереве каталогов](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BA%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2&action=edit&redlink=1). При таких операциях невозможно получить доступ к содержимому файла, файл является минимальной единицей деления информации.

Возможные операции с файлами: создание, удаление, переименование, копирование, перенос на другую файловую систему, создание [символьной ссылки](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0) или [жёсткой ссылки](https://ru.wikipedia.org/wiki/%D0%96%D1%91%D1%81%D1%82%D0%BA%D0%B0%D1%8F_%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0), получение и изменение атрибутов.

В зависимости от файловой системы, носителя информации, операционной системы часть операций может быть недоступна.

1. **Характеристика структури – стек. Приклади задач, у яких використовується ця структура.**

**Стек** (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Стек является общей структурой данных для представления данных, которые должны обрабатываться в определенном порядке. Например, когда функция вызывает другую функцию, которая, в свою очередь, вызывает третью функцию, важно, чтобы третья функция вернулась на вторую функцию, а не первую.

**Операции со стеком**

Возможны три операции со стеком: добавление элемента (иначе проталкивание, push), удаление элемента (pop) и чтение головного элемента (peek)[5].

При проталкивании (push) добавляется новый элемент, указывающий на элемент, бывший до этого головой. Новый элемент теперь становится головным.

При удалении элемента (pop) убирается первый, а головным становится тот, на который был указатель у этого объекта (следующий элемент). При этом значение убранного элемента возвращается.

**Область применения**

Программный вид стека используется для обхода структур данных, например, дерево или граф. При использовании рекурсивных функций также будет применяться стек, но аппаратный его вид. Кроме этих назначений, стек используется для организации стековой машины, реализующей вычисления в обратной инверсной записи.

Для отслеживания точек возврата из подпрограмм используется стек вызовов.

Арифметические сопроцессоры, программируемые микрокалькуляторы и язык Forth используют стековую модель вычислений.

Идея стека используется в стековой машине среди стековых языков программирования.

Применение стека упрощает и ускоряет работу программы, так как идет обращение к нескольким данным по одному адресу.

1. **Характеристика структури – дек. Приклади задач, у яких використовується ця структура.**

**Дек (от англ. deque — double ended queue)** — структура данных, представляющая из себя список элементов, в которой добавление новых элементов и удаление существующих производится с обоих концов. Эта структура поддерживает как FIFO, так и LIFO, поэтому на ней можно реализовать как стек, так и очередь. В первом случае нужно использовать только методы головы или хвоста, во втором — методы push и pop двух разных концов. Дек можно воспринимать как двустороннюю очередь. Он имеет следующие операции:

empty — проверка на наличие элементов,

pushBack (запись в конец) — операция вставки нового элемента в конец,

popBack (снятие с конца) — операция удаления конечного элемента,

pushFront (запись в начало) — операция вставки нового элемента в начало,

popFront (снятие с начала) — операция удаления начального элемента.

1. **Характеристика структури – черга. Приклади задач, у яких використовується ця структура.**

**О́чередь** — структура данных с доступом к элементам: «первый пришёл — первый вышел» (FIFO, англ. first in, first out). Добавление элемента (enqueue) возможно лишь в конец очереди, выборка — только из начала очереди (dequeue), при этом выбранный элемент из очереди удаляется.

* Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.
* Очереди часто используются в программах для реализации буфера, в который можно положить элемент для последующей обработки, сохраняя порядок поступления. Например, если база данных поддерживает только одно соединение, можно использовать очередь потоков, которые будут, как ни странно, ждать своей очереди на доступ к БД.

1. **Характеристика структури – рядок. Приклади задач, у яких використовується ця структура.**
2. **Характеристика структури – дерева. Приклади задач, у яких використовується ця структура.**

**Дерево** —структура данных эмулирующая древовидную структуру в виде набора связанных узлов. Является связным графом, не содержащим циклы. Большинство источников также добавляют условие на то, что рёбра графа не должны быть ориентированными. В дополнение к этим трём ограничениям, в некоторых источниках указывается, что рёбра графа не должны быть взвешенными.

**Корневой узел** — самый верхний узел дерева

**Корень** — одна из вершин, по желанию наблюдателя.

лист, листовой или терминальный узел — узел, не имеющий дочерних элементов

**Внутренний узел** — любой узел дерева, имеющий потомков, и таким образом, не являющийся листовым узлом

Дерево считается ориентированным, если в корень не заходит ни одно ребро.

* управление иерархией данных;
* упрощение поиска информации (см. обход дерева);
* управление сортированными списками данных;
* синтаксический разбор арифметических выражений (англ. parsing), оптимизация программ;
* в качестве технологии компоновки цифровых картинок для получения различных визуальных эффектов;
* форма принятия многоэтапного решения (см. деловые шахматы).

1. **Характеристика структури – бінарне дерево. Приклади задач, у яких використовується ця структура.**

**Двои́чное де́рево** — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево не является упорядоченным ориентированным деревом.

• Используются для сжатия файлов

1. **Характеристика структури –дерево пошуку. Приклади задач, у яких використовується ця структура.**

это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

• У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.

• У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

• Для реализации множеств

• Реализация ассоциативных массивов

1. **Характеристика структури – графи. Приклади задач, у яких використовується ця структура.**

Граф представляет собой набор узлов, соединенных друг с другом в виде сети. Узлы также называются вершинами. Пара (x, y) называется ребром, которое указывает, что вершина x соединена с вершиной y. Ребро может содержать вес/стоимость, показывая, сколько затрат требуется, чтобы пройти от x до y.

• Поиск кратчайшего пути

1. **Методи сортування масивів. Загальна характеристика.**

Время выполнения, объем доп. Памяти, устойчивость.

• **Бабл** (O(n^2)) – простейший, в учебных целях:

За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются {\displaystyle N-1} N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде. Отсюда и название алгоритма).

• **Вставками (инсерт)** - алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов Θ(n^2),

• **Выбором (селект)** - алгоритм сортировки. Может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(n^2), предполагая что сравнения делаются за постоянное время.

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

• **Слиянием(мерж)** - алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

N(log N)

• **Быстрая сортировка(квик сорт)** - Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(n log n) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.