计算机学习笔记

Jackyy Hua

hshuai97@qq.com

2022 年 9 月 27 日

目录

摘要

好记性要靠烂笔头。笔记旨在记录学习中的一些常见易忘知识，温故而知新，日积月累。文中代码均上机运行通过。

关键词：C++；Python; PHP; 数据结构; PyTorch; DGL; Latex; 面试笔记

1 C++

C++ 主要用于基于操作系统之上的软件开发，而 C 语言主要用于操作系统中的管理系统开发。

1.1 类型转换运算符

64bit 操作系统下，C++ 中常用的基本数据类型：bool (1 byte), int (4 bytes), float (4 bytes), double (8 bytes), char (1 byte), string。一个类型转换运算符（type casting operator）是一个一元运算符，其将一种数据类型强制转为另一种数据类型。C++ 主要有以下几种类型转换[1]：

- **static_cast**
- **dynamic_cast**
- **constant_cast**
- **reinterpret_cast**

静态类型转换 static_cast 发生在编译阶段，没有运行时（run-time）的检查；dynamic_cast 发生在运行时，更安全一些。

1.2 C++ 中的 Templates

template[2]主要用于函数和类的数据传入。

函数 template 用于为一个功能函数传递不同的数据类型。其在编译阶段进行该功能函数的复制扩展以适应不同的数据类型，编译后的代码将会变多：

```
#include<iostream>
using namespace std;
//typename关键字可以用class关键字代替
template <typename T> T fun(T x, T y) { //函数模板
   return (x > y) ? x : y;
}

int main() {
   cout << fun<int>(2, 3) << endl; //编译阶段会用int代替T
   cout << fun<char>('a', 'b') << endl; //编译阶段会用char代替T

   return 0;
}
```

类 template,

[1] https://www.geeksforgeeks.org/static_cast-in-c-type-casting-operators/

[2] https://www.geeksforgeeks.org/templates-cpp/

1.3 范围解析运算符

C++ 中的范围解析运算符（scope resolution operator）:: 主要有 6 个用途，参考[3]:

- **当局部变量和全局变量同名时，读取全局变量**
- **用于在类外定义函数**
- **用于读取类的静态变量**
- **多继承时，区分不同基类中相同的变量**
- **用于命名空间**
- **引用类中的类的成员变量**

引用全局变量：

```
#include <iostream>
#include <string>
using namespace std;

int x=2;
int main() {
   int x=1;
   cout << ::x << endl; //输出全局变量值2
   cout << x << endl; //输出局部变量值1

   return 0;
}
```

在类的外部定义函数：

```
#include <iostream>
#include <string>
using namespace std;

class A {
public:
   void fun(int x); //只声明函数
};

//在类外定义函数
void A::fun(int x) {
   cout << x << endl;
}

int main() {
   int x1=2;
   A a;
   a.fun(x1); //调用类的函数输出2

   return 0;
}
```

读取类的静态变量：

[3] https://www.geeksforgeeks.org/scope-resolution-operator-in-c/

```
#include <iostream>
#include <string>
using namespace std;

class A {
private:
   static int x;
public:
   static int y;
   void fun(int x) {
      cout << "A::x is " << A::x << endl; //输出1
      cout << "Local x is " << x << endl; //输出3
   }
};

int A::x = 1; //类中静态成员变量必须显示初始化
int A::y = 2;

int main() {
   int x=3;
   A a;
   a.fun(x);
   cout << A::y << endl; //输出2，注：类外只可调用类中的public成员

   return 0;
}
```

当一个衍生类继承自多个基类时，用以区别不同基类中同名的变量：

```
#include <iostream>
#include <string>
using namespace std;

class A {
protected:
   int x;
public:
   A() {
      x = 1;
   }
};

class B {
protected:
   int x;
public:
   B() {
      x = 2;
   }
};

class C: public A, public B {
public:
   void fun() {
      cout << "A's x is " << A::x << endl; //输出1
      cout << "B's x is " << B::x << endl; //输出2
   }
```

```
};

int main() {
   C c;
   c.fun();

   return 0;
}
```

用于命名空间：

```
#include<iostream>

int main(){
   std::cout << "Hello" << std::endl;

   return 0;
}
```

用于引用类中的类的成员变量（类中可以再定义类）：

```
#include<iostream>
using namespace std;

class A{
public:
   int x;
   class B{ //必须是A类的public成员，B的成员变量才能被外部函数访问
   public:
      int x;
      static int y;
      int foo() {
         cout << "Hello" << endl;
      }
   };
};

int A::B::y = 5; //类的静态成员必须如此显示的初始化

int main() {
   cout << A::B::y << endl; \\可以调用public的成员(B类)的成员
   return 0;
}
```

1.4 类的知识

需要了解的类的知识：

- **类对象指针与类对象**
- **基类、衍生类所占内存**
- **类的构造函数与析构函数**
- **重载（overload）与覆盖（override）**

1.4.1 类对象指针与类对象

类对象指针和类对象名都可以调用类的成员:

```
#include <iostream>
using namespace std;
class A {
   int x; // 默认是private
public:
   void f(){
      cout<<"hello"<<endl;
   }
};

int main(){
   A *a; //声明A类的对象指针变量
   A a1; //声明A类的对象
   a = &a1; //将对象a的地址赋值给指针变量a，使得a指向类对象
   a->f(); //等价于 (*a).f(); 等价于a1.f()

   return 0;
}
```

1.4.2 基类、衍生类所占内存

空类占有 1 个字节（因为系统会为实例化的对象分配一个地址，空类也可以被实例化），指针在 32bit 系统中占 4 个字节，在 64bit 系统中占 8 个字节，与指针所指向的具体数据类型无关。类函数是实例化的对象的共用函数，不计入类所占用的内存（虚函数情况见下面介绍）。

基类所占内存为类成员中依次出现的成员按最大对齐，结构体与类的计算方法类似（拼接对齐）：

```
//sizeof(A)返回12
class A{
   char a0; //对齐，4 bytes，只使用1 byte,浪费3 bytes
   int a1; //最大的为4 bytes
   float a2; //4 bytes
};

//sizeof(B)返回8
class B{
   int b1; //4 bytes
   char b2; // 对齐，4 bytes,只使用1 byte
   char b3; // 对齐，使用前面的剩余的3 bytes中的1 byte
};

//sizeof(C)返回16
class C{
   int c1; //对齐，8 bytes，只使用4 bytes
   char c2; //对齐，使用前面剩余的4个bytes，浪费3个 bytes
   double c3; //对齐，使用8 bytes
};
```

类中如果有一个或多个虚函数，则会有且仅有一个指向虚函数的指针, 同样适用于上面的对齐机制:

```
//sizeof(A)返回16
class A {
   char a; //对齐，8 bytes, 只使用1 byte
   int a1; //对齐，使用前面剩的7 bytes中的4bit, 浪费3 bytes
public:
    virtual void f() {} //虚函数有一个指针，64位系统指针为8 bytes
    virtual void f1() {}
};
```

衍生类所占内存大小是本类的大小加上父类的大小，两者全局对齐（必须全局对齐，即衍生类成员和基类成员对齐），内部单独计算再相加，衍生类的衍生类也必须按照该方法进行计算：

```
//sizeof(A)仍然是8
class A {
   int a0;
   char a1;
};

//衍生类，先将基类的成员和衍生类的成员中最大的对齐，不能拼接，然后分别计算再相加
//sizeof(B)返回 24
class B: public A {
   char b0; //对齐，  8 bytes, 占用1 byte
   double b1; //对齐，  8 bytes
};

//sizeof(C)返回32
class C : public B {
   char c0;
};
```

1.4.3 类的构造函数与析构函数

类的构造函数（constructor）与析构函数（deconstructor）：构造函数与析构函数都没有返回值，一个类可以有多个构造函数，只有一个析构函数，析构函数不能重载，构造函数可以重载，重载可以使用默认参数。

1.4.4 重载与覆盖

重载（overload）: 重载是在一个类中，有多个同名函数，但通过参数类型的不同可以区分开的情况称为重载。不可重载的运算符:

- :: 范围解析运算符
- . 成员读取运算符
- .* 通过指向成员的指针读取成员的运算符

- ?: 三元条件运算符 (ternary conditional)

覆盖（override）：覆盖是在衍生类中，使用与基类同名的函数，但函数功能不同，称之为覆盖。

1.5 C++ 之 Qt 工具

Qt 是 C++ 的主流界面开发工具

1.5.1 用 QT 写一个 PC 小游戏

需求：娱乐

功能：

1.5.2 用 C++ 写 ios 投屏软件

需求：开发一款投屏软件用于个人自助理发，市面上的此类软件均需收费且存在诸多问题；具有相似功能的视频会议单个 ID 操作不便，监控摄像头需要额外购买。

功能：将 iphone 相机画面实时传输到 ipad 上，使用 VLAN 通信。

1.6 C++ 的其他知识

1.6.1 using namespace std

namespace 的功能同范围解析运算符::，都是用来指示引用特定的成员变量、成员函数等：

```
namespace std{ //提前定义了标准库中的一些常用的输入输出函数
   iostream cout;
   iostream cin;
   iostream endl;
   iostream cerr; // 标准错误流
   iostream clog; //标准日志流
}

using namespace std;//全局任意地方可直接使用iostream中的cout等函数
```

1.6.2 vector 向量

1.6.3 this 指针

1.6.4 new 运算符

new operator 用来为类对象分配内存，delete operator 则是释放内存。

1.6.5 virtual 函数

虚函数能够保证类对象在不论用何种类型引用类型或者指针类型调用类成员函数时能准确调用（实现运行时的多态），其中一些关于虚函数的约束[4]：

- 虚函数不能设置为 static;
- 虚函数可以成为另一个类的友元函数；
- 虚函数可以用指针、引用来操作以实现运行时的多态;
- 虚函数的原型应当在基类和衍生类中保持相同；
- 虚函数总是在基类中定义，在衍生类中进行覆盖；（衍生类中可以不覆盖，将会使用基类的虚函数）;
- 一个类可以有虚析构函数，但不能有虚构造函数

运行时的多态体现（多态出现在同名函数的情况中）：

```
#include<iostream>
using namespace std;

class A {
   int a;
public:
   virtual void fun1() {
      cout << "base fun1" << endl;
   }
   void fun2() {
      cout << "base fun2" << endl;
   }
};

class B : public A {
public:
   void fun1() {
      cout << "derived fun1" << endl;
   }
   void fun2() {
      cout << "derived fun2" << endl;
   }
};

int main() {
   A* a;
   B b;
   a = &b; //用指向基类的指针指向衍生类，不安全的用法

   a->fun1(); //基类指针指向的基类成员函数为虚函数，则输出衍生类的fun1
   a->fun2(); //基类指针指向的基类成员函数为正常函数，则输出基类的fun2

   return 0;
}
```

[4] https://www.geeksforgeeks.org/virtual-function-cpp/

1.6.6 typeid 运算符

typeid 是一个 operator, 包含在 typeinfo 库中，用于返回对象的动态数据类型或运行时数据类型。一个示例：

```
#include<iostream>
#include<typeinfo>
using namespace std;

int main() {
   char c[] = "abcd";
   const type_info& ti1 = typeid(c);
   cout << ti1.name(); //char [5], 字符串长度为4加'\0'作为结束标志

   return 0;
}
```

2 Python

2.1 基本数据类型

2.2 用 python 绘实验结果图

3 PHP

3.1 PHP 的基本知识

3.2 PHP 项目

PHP 的一个具体项目：

4 Tomcat

5 Linux

5.1 Linux 的常用命令

5.2 Linux 系统管理维护

6 PyTorch

This is.

7 DGL

7.1 图的构建及可视化

构件图：

```
import dgl
u, v = [0, 0, 0, 1, 1], [0, 1, 2, 0, 1]
g = dgl.graph((u, v), num_nodes=4)
g.ndata['x'] = torch.rand(4, 8)
print(g.ndata
```

将上面的图可视化：

```
import networkx as nx
import matplotlib.pyplot as plt
nx_g = g.to_networkx() # .to_undirected()
# Kamada-Kawaii layout usually looks pretty for arbitrary graphs
pos = nx.kamada_kawai_layout(nx_g)
nx.draw(nx_g, pos, with_labels=True, node_color=[[0.7, 0.3, 0.3]])
plt.show()
```

8 文本分类

8.1 卷积神经网络

8.2 图神经网络

9 语音处理

10 数据库

数据库的基本知识：

10.1 SQLite 数据库的使用

SQLite 是用 C 语言编写的一款开源的轻量 (small)、快速 (fast)、自容 (high-reliability)、高可靠 (high-reliability)、全功能 (full-featured) 的数据库[5].

10.2 Hive 数据仓库工具

Hive 是基于 Hadoop 的一个仓库管理工具，其可以自动将 SQL 语句转成 MapReduce 任务，主要用于大数据的存储与分析。

11 计算机网络

11.1 虚拟局域网 VLAN

11.2 HTTP 协议

11.3 TCP、IP 协议

11.4 WebSocket

12 数据结构

本节的知识主要参考率辉主编的 2020 版《数据结合高分笔记》，主要记录常见易忘内容。

[5] https://www.sqlite.com/index.html

12.1 线性表与链表

12.2 栈和队列

12.2.1 顺序栈和链栈的定义

栈的数学性质：当 n 个元素以某种顺序进栈，任意时刻可以出栈，则一共有 N 种排列：

$$N = \frac{1}{n+1} C_{2n}^{n} \tag{1}$$

顺序栈的定义：

```
int stack[maxSize]; int top = -1; //maxSize事先已定义
```

链栈的定义：

```
typedef struct LNode{
   int data;
   struct LNode *next;
}LNode;
```

顺序栈具有 FILO 的特性，具有记忆功能。例子：C 语言中算数表达式只有小括号，编写函数，判断一个表达式中的括号是否正确配对，表达式存在字符数组 exp[] 中，表达式字符个数为 n。

```
#include<iostream>
using namespace std;

int match(int exp[], int n){
   char a[n]; int top=-1;
   for (int i=0; i<=n-1; ++i){
      if (a[i] == '(')
         a[++top] = '(';
      if (a[i] == ')')
         if (top == -1) //如果栈空时进')', 则括号不匹配
            return 0;
         else
            --top;
   }
   if (top == -1)
      return 1; //括号正确匹配
   else
      return 0;
}

int main(){
   int n = 5;
   char a[n] = {'(', 'a', '+', 'b', '('};
   int flag = match(a, n);
   if （flag==0）
      cout << "不匹配" << endl;
   else
      cout << “匹配” << endl;
   return 0;
}
```

链栈的基本操作例子：用不带头节点的单链表存储链栈，设计初始化栈、判断栈是否为空、进栈和出栈函数。

```
void initStack(LNode *&lst){ //初始化栈
   lst = NULL;
}

int isEmpty(LNode *lst){ //判断链栈是否为空
   if (lst == NULL)
      return 1;
   else
      return 0;
}

void push(LNode *&lst, int x){ //元素x进栈
   LNode *p;
   p = (LNode *)malloc(sizeof(LNode)) //申请一个节点所需空间
   p->next = NULL;
   p->data = x;

   //插入操作
   p->next = lst;
   lst = p;
}

int pop(LNode *&lst, int &x){ // 元素x出栈
   LNode *p;
   if (lst == NULL)
      return 0;
   x = lst->data;

   //删除操作
   p = lst;
   lst = p->next;
   free(p);
}
```

12.2.2 顺序队列和链队列的定义

顺序队列一般使用改进的循环顺序队列（原始顺序队列会有”假溢出“问题，即 rear 和 front ” 指针” 只能朝都递增或都递减方向移动），循环顺序队列的定义为：

```
typedef struct SqQueue{ //定义一个顺序队列
   int data;
   int front;
   int rear;
}SqQueue;

void init(SqQueue &qu){ //初始化顺序队列，要对原始数组修改，用&应用型
   qu.front = qu.rear = 0;
}

//判断顺序队是否为空
int isEmpty(SqQueue qu){
```

```
    if (qu.front == qu.rear) //队空
        return 1;
    else
        return 0;
}

//元素x进队
int push(SqQueue &qu, int x){
    if ((qu.rear+1) % maxSize==qu.front) //队满不能进队
        return 0;
    qu.rear = (qu.rear + 1) % maxSize;
    qu.data[qu.rear] = x;
    return 1;
}

//元素x出队
int pop(SqQueue &qu, int &x){
    if (qu.rear == qu.front) //队空不能出队
        return 0;
    qu.front = (qu.front + 1) % maxSize;
    x = qu.data[qu.front];
    return 1;
}
```

链队列的定义：

```
//先定义链队列的节点
typedef struct QNode{
    int data;
    struct QNode *next;
}QNode;

//再定义链队列
typedef struct{
    QNode *front;
    QNode *rear;
}LiQueue;

//初始化链队
void init(LiQueue *lqu){ lqu指针指向的节点发生改变,指向了新申请的节点
    lqu = (LiQueue*)malloc(sizeof(LiQueue));
    lqu->front=lqu->rear=NULL;
}

//判断链队列是否为空
int isEmpty(LiQueue *lqu){
    if (lqu->rear==NULL || lqu->front==NULL) //任意指针为空则链队列为空
        return 1;
    return 0;
}
```

链队列进队没有限制，出队时队空不能出队：

```
 //元素x进链队列
int push(LiQueue *lqu, int x){
    QNode *p;
    p = (QNode *)malloc(sizeof(QNode));
```

```
    p->next = NULL;
    p->data = x;

    //插入操作
    lqu->rear->next=p;
    lqu->rear = p;

    return 1;
}

//元素x出栈
int pop(LiQueue *lqu, int &x){
    LiQueue *p;
    if (lqu->front==NULL || lqu->rear==NULL)
        return 0;
    p = lqu->front;

    if (lqu->front == lqu->rear){ //仅有一个节点时需将rear和front同时置NULL
        lqu->front = lqu->rear = NULL;
    }
    lqu->front = lqu->front->next;

    x = p->data;
    free(p);
    return 1;
}
```

顺序队列 (循环顺序队列) 要比链队列使用起来更简约，但对内存操作不如链队列方便，对于需要用队列数据结构来解决问题的，优先使用循环顺序队列。

12.2.3 抽象数据类型 ADT

设计一个图书馆 ADT：数据对象集，数据关系集、操作集

```
ADT 书{
    数据对象集:
        书名;
        书号;
        作者;
};

ADT 书架{
    数据对象集:
        书架号;
        书 = {书0, 书1, ..., 书n};
    数据关系集:
        书在书架的排列方式 = {<书0, 书1>, <书1, 书2>, ... , <书n, 书n-1>};
};

ADT 图书馆{
    数据对象集:
        图书馆名;
        书架 = {书架0, 书架1, ..., 书架n};
    数据关系集:
        书架在图书馆的排列方式 = {<书架0, 书架1>, <书架1, 书架2>, ..., <书架n-1,
```

```
            书架n>};
    数据操作集:
        根据书架号查找书架;
        根据书号查找书;
        根据书名查找作者;
        借还书;
};
```

12.3 串

串即字符串，C 语言中的字符串可以如下定义：

```
char str[] = "hello world"; //求str长度需要遍历字符串每个元素,复杂度O(n)

char str[maxSize], int length = 11; //时间复杂度为O(1)
```

C++ 中增加了一个 string 库，包含了字符串的基本操作，每个字符串是一个字符数组：

```
#include<iostream>
# include <string>
using namespace std;

int main() {
    string s1 = "Hello";
    string s2 = "Hellp";
    string s3;
    string s4 = "ael";

    //C++的string库已包含常用的字符串操作
    s3 = s1; //赋值运算
    s3 = s1 + s2; //拼接操作
    int len = s3.length(); // 求字符串的长度（不包括'\0'）
    //int a = s3.size(); size()和length()函数功能相同

    if (s1 < s2) //字符串直接比较大小
        cout << "s1 is less than s2" << endl;

    size_t found = s1.find(s4); //字符串的查找匹配
    if (found != string::npos)
        cout << "index: " << found << endl;
    if (found == string::npos)
        cout << "No found" << endl;

    return 0;
}
```

12.3.1 KMP 算法

KMP 算法优点：不需要回溯，对于规模较大的外存中的字符串的匹配操作可以分段进行，分段读入内存进行模式匹配后写回外存，在发生不匹配时确保外存中的字符串没有符合的情况，不需要再次读入内存。

12.4 矩阵与广义表

下三角矩阵用一维数组表示：

12.5 树与二叉树

12.6 图

12.7 排序

排序的概念:，排序的稳定性分析，排序的时间复杂度分析及比较，基本的排序方法有：

- 插入类排序：直接插入排序、折半插入排序、希尔排序
- 交换类排序：冒泡排序、快速排序
- 选择类排序：简单选择排序、堆排序
- 二路归并排序
- 基数排序
- 外部排序：置换-选择排序、最佳归并树、败者树

12.7.1 插入类排序

插入类排序主要有 3 种：直接插入排序、折半插入排序和希尔排序。

直接插入排序：

折半插入排序：

希尔排序：

12.7.2 交换类排序

交换类排序主要有冒泡排序 (bubble sort) 和快速排序。

冒泡排序如下，其时间复杂度 $T(n) = O(n^2)$, (最好情况下，整个序列有序，只需计算 n 次，复杂度为 $O(n)$), 空间复杂度：额外辅助空间只有 temp，因此空间复杂度为 $O(1)$。

```
#include <iostream>
using namespace std;

template <class T> void BubbleSort(T a[], int n){
   int flag; int temp;
   for (int i=n-1; i>=1; --i){
      flag = 0;
      for (int j=1; j<=i; ++j)
         if (a[j-1]>a[j]){
            temp = a[j-1];
            a[j-1] = a[j];
            a[j] = temp;
            flag = 1;
         }
      if (flag==0)
         return
   }
}

int main(){
```

```
    int a[5] = {20, 30, 40, 10, 5};
    char b[4] = {'b', 'a', 'd', 'c'};
    n1 = sizeof(a) / sizeof(a[0]);
    n2 = sizeof(b) / sizeof(b[0]);

    BubbleSort<int>(a, n1);
    BubbleSort<char>(b, n2);

    for (int i = 0; i<= n1-1; ++i)
       cout << a[i] << endl;
    for (int i = 0, i <= n2-1; ++i)
       cout << b[i] << endl;

    return 0;
}
```

快速排序，时间复杂度为：

12.7.3 选择类排序

选择类排序主要有 2 种：简单选择排序和堆排序：

12.7.4 二路归并排序

12.7.5 基数排序

12.7.6 外部排序

外部排序主要有 3 种：置换-选择排序，最佳归并树和败者树。

置换-选择排序：先置换再选择排序

最佳归并树：

败者树：

12.8 查找

12.8.1 查找的概念

12.8.2 顺序查找、折半查找

12.8.3 二叉排序树与平衡二叉树

12.8.4 B-树

B-树的基本概念：

B-树的基本操作：

12.8.5 散列表

12.9 数据结构其他知识

12.9.1 中缀转后缀、前缀

中缀表达式 $a*(b+c)-d$ 转后缀, 根据运算优先级，依次将每次最小运算单元中的运算符提到最后，变量相对位置不变: $abc+*d-$。另一种方法是按运算优先级加括号，将右括号用运算符替换。

中缀转前缀同理，按运算优先级将最小的运算单元中的运算符提出放最前，变量相对位置不变。则示例变为：$-*a+bcd$.

13 Latex

13.1 图、表的使用

图的使用：

```
\begin{figure}[H]
\center
\includegraphics*[width=12cm]{fig/1.png}
\centering
\caption{图片示例}\label{g1-1}
\end{figure}
```

表的使用：

```
\begin{table}[htbp]
\centering
\caption{this is a caption}
\begin{tabular}{cccccc}
   \hline
   Model & 20NG & R8 & R52 & Ohsumed & MR \\
   \hline
   TextGCN & 0.8634 & 0.9707 & 0.9356 & 0.6836 & 0.7674 \\
   \hdashline
   HieGAT & 0.8584 & \textbf{0.9783} & 0.9454 & 0.6984 & \textbf
      {0.7804}\\
   \hline
\end{tabular}
\end{table}
```

13.2 在 latex 中写代码

使用 listings 包，参考 Overleaf 的介绍[6]：

```
\begin{lstlisting}[language=Python, caption=Python example]
   import numpy as np

   def function(m,n):
      return m+n
```

[6]https://www.overleaf.com/learn/latex/Code_listing

```
\ end{lstlisting}
```

13.3 数学符号、公式的排版

附录 A 中列举了详细的数学符号，这里记录一些常用数学符号：

```
约等于：\approx，点乘：\cdot，叉乘：\times，省略号：\cdots，\ldots，
垂直省略号：\vdots，斜省略号：\ddots，除号：\div，向量箭头：\vec{a}，
大于等于：\geq，小于等于：\leq，不等于：\neq，正比于：\propto
```

数学运算符下方加条件限制：

```
\max\limits_{a<x<b}
```

示例效果：$\max\limits_{a<x<b}$

上下居中对齐公式排版：

```
\begin{align}
   & y = f(x) \\
   & z = g(x) + f(x)
\end{align}
```

效果如下：

$$y = f(x) \tag{2}$$

$$z = g(x) + f(x) \tag{3}$$

一对多公式排版：

```
\begin{equation*}
   LeakyReLu(x) = \begin{cases}
      x, & x \geq 0; \\
      negative\_slope * x, & <0
   \end{cases}
\end{equation*}
```

效果如下：

$$LeakyReLu(x) = \begin{cases} x, & x \geq 0; \\ negative\_slope * x, & < 0 \end{cases}$$

矩阵形公式：

```
\begin{equation*}
   \left[
   \begin{array}{c}
      y_1 \\
      y_2
   \end{array}
   \right]
   =Softmax (
   \left[
   \begin{array}{cc}
      w_{11} & w_{12} \\
      w_{21} & w_{2}
   \end{array}
   \right] +
```

```
    \left[
    \begin{array}{c}
       b_1 \\
       b_2
    \end{array}
    \right]
    )
\end{equation*}
```

矩阵公式效果如下：

$$\left[\begin{array}{c} y_1 \\ y_2 \end{array}\right] = Softmax(\left[\begin{array}{cc} w_{11} & w_{12} \\ w_{21} & w_2 \end{array}\right] + \left[\begin{array}{c} b_1 \\ b_2 \end{array}\right])$$

13.4 用 latex 写 PPT

调整字体大小：

```
\tiny、\small、\large、\Large、\LARGE、\huge、\Huge
```

幻灯片脚注标注参考文献: footfullcite{}

删除 PPT 底部图标:

```
\setbeamertemplate{navigation symbols}{}
\setbeamertemplate{footline}[page number]{} //只保留页码
```

13.5 罗列事项

按序号罗列：

```
\begin{enumerate}
   \item ...
   \item ...
\end{enumerate}
```

按点罗列：

```
\begin{itemize}
   \item ...
   \item ...
\end{itemize}
```

14 其他常识

14.1 Fibonacci 数

Fibonacci number (sequence): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …. 其数学表达式为：$F_n = F_{n-1} + F_{n-2}(n \geq 2, F_0 = 0, F_1 = 1)$

14.2 排列组合 Cn 和 An

排列：从 n 个不同的元素中取出 m 个元素，**按一定的顺序排成一列**，所能得到的所有排列个数。符号为 A_n^m，计算公式为：

$$A_n^m = n(n-1)\cdots(n-m+1) = \frac{n!}{(n-m)!}, (A_n^n = n!, 0! = 1) \tag{4}$$

组合：从 n 个不同元素中取出 m 个元素，**把它并成一组**，所能得到的所有组合数。符号为 C_n^m，计算公式为：

$$C_n^m = \frac{n(n-1)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}, (C_n^0 = 1) \tag{5}$$

其中 $A_n^m = C_n^m \cdot A_m^m, A_n^m = nA_{n-1}^{m-1}, C_n^m = C_n^{n-m}, C_{n+1}^m = C_n^m + C_n^{m-1}$

14.3 t 检验

15 面试与笔试

15.1 面试

- 回答问题的 **star** 原则: situation, task, action, result

15.1.1 自我介绍

专业相关的自我介绍：下午好面试官，我叫华帅，本科读的专业是计算机科学与技术，绩点 3.19，没有挂科及相关不良记录，硕士阶段研究方向为自然语言处理与智能计算，绩点 3.42，年级排名前 20。我对 C++ 软件开发以及数据挖掘数据分析比较感兴趣，同时了解深度学习的一些基本知识及深度学习框架 PyTorch，熟悉 python 编程、PHP 网页开发，熟练使用文档排版工具 Latex。

15.1.2 软件开发的设计模式

设计模式是众多程序员前辈的经验总结，可以有效提升开发效率，减少走弯路的可能。（本节参考该网站[7]）

设计模式主要包括创建型模式，结构型模式和行为模式 3 大类。

创建型模式提供了创建对象的机制，能够提升已有代码的灵活性和可复用性。主要分为：

- 工厂方法（Factory Method）
- 抽象工厂 (Abstract Factory)
- 生成器 (Builder)
- 原型 (Prototype)
- 单例 (Singleton)

结构型模式介绍如何将对象和类组装成更大的结构，并同时保持结构的灵活和高效。主要分为：

- 适配器 (Adapter)
- 桥接 (Bridge)

[7] https://refactoringguru.cn/design-patterns/cpp

- 组合 (Composite)
- 装饰 (Decorator)
- 外观 (Facade)
- 享元 (Flyweight)
- 代理 (Proxy)

行为模式负责对象间的高效沟通和职责委派。主要分为：

- 责任链 (Chain of Responsibility)
- 命令 (Command)
- 迭代器 (Iterator)
- 中介者 (Mediator)
- 备忘录 (Memento)
- 观察者 (Observer)
- 状态 (State)
- 策略 (Strategy)
- 模板方法 (Template Method)
- 访问者 (Visitor)

15.2 笔试选择题

- 使用 0-9 这 10 个数进行排列，要求任意两个数字都相邻出现过，求这个序列的最小长度。

- 排成一排的 10 棵树，每棵间隔 10 米，第一棵树处有水井，小张为每一棵树浇水，则至少要走多少米。

- 小明早上发现闹钟没电停了，换了电池调到 7：10，去图书馆，图书馆时钟正常工作，为 8：50，10：20 离开图书馆，以和来时相同速度返回家，这时家里闹钟为 11：50，请问应当把闹钟纠正到几点。

- 已知 a 数比 b 数大 75%, 则 b 数比 a 数小多少。

15.3 笔试编程题

- 字符串的消消乐：一个长度为 n 的字符串，A 和 B 两人轮流操作，问先手是否可以获胜。

```
输入示例：  3
           abcdcba
           aab
           baab

输出示例：  No
           Yes
           No
```

- 裁剪邮票：
- 水池抽水排水问题

```
//用周期函数求解
```

A latex 数学符号大全

数学公式一般使用主流的标准符号定义, 可以在这个网页[8]找到这本书"Deep Learning"[1]，常用符号的定义如下，需要使用 `math_commands.tex`:

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathbf{A}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

[8] https://github.com/goodfeli/dlbook_notation

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{A}$
$\mathbf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

下面是对应的 latex 代码：

Numbers and Arrays:

```
\centerline{\bf Numbers and Arrays}
\begin{tabular}{p{1in}p{5in}}
   $\displaystyle a$ & A scalar (integer or real)\\
   $\displaystyle \va$ & A vector\\
   $\displaystyle \mA$ & A matrix\\
   $\displaystyle \tA$ & A tensor\\
   $\displaystyle \mI_n$ & Identity matrix with $n$ rows and $n$ columns
      \\
   $\displaystyle \mI$ & Identity matrix with dimensionality implied by
      context\\
   $\displaystyle \ve^{(i)}$ & Standard basis vector $[0,\dots,0,1,0,\
      dots,0]$ with a 1 at position $i$\\
   $\displaystyle \text{diag}(\va)$ & A square, diagonal matrix with
      diagonal entries given by $\va$\\
   $\displaystyle \ra$ & A scalar random variable\\
   $\displaystyle \rva$ & A vector-valued random variable\\
   $\displaystyle \rmA$ & A matrix-valued random variable\\
\end{tabular}
```

Sets and Graphs:

```
\vspace{0.5cm}
\centerline{\bf Sets and Graphs}
\begin{tabular}{p{1in}p{5in}}
   $\displaystyle \sA$ & A set\\
   $\displaystyle \R$ & The set of real numbers \\
   $\displaystyle \{0, 1\}$ & The set containing 0 and 1 \\
   $\displaystyle \{0, 1, \dots, n \}$ & The set of all integers between
      $0$ and $n$\\
   $\displaystyle [a, b]$ & The real interval including $a$ and $b$\\
   $\displaystyle (a, b]$ & The real interval excluding $a$ but including
       $b$\\
   $\displaystyle \sA \backslash \sB$ & Set subtraction, i.e., the set
      containing the elements of $\sA$ that are not in $\sB$\\
```

```
   $\displaystyle \gG$ & A graph\\
   $\displaystyle \parents_\gG(\ervx_i)$ & The parents of $\ervx_i$ in $\
      gG$
\end{tabular}
```

Indexing:

```
\vspace{0.5cm}
\centerline{\bf Indexing}
\begin{tabular}{p{1in}p{5in}}
   $\displaystyle \eva_i$ & Element $i$ of vector $\va$, with indexing
      starting at 1 \\
   $\displaystyle \eva_{-i}$ & All elements of vector $\va$ except for
      element $i$ \\
   $\displaystyle \emA_{i,j}$ & Element $i, j$ of matrix $\mA$ \\
   $\displaystyle \mA_{i, :}$ & Row $i$ of matrix $\mA$ \\
   $\displaystyle \mA_{:, i}$ & Column $i$ of matrix $\mA$ \\
   $\displaystyle \etA_{i, j, k}$ & Element $(i, j, k)$ of a 3-D tensor $
      \tA$\\
   $\displaystyle \tA_{:, :, i}$ & 2-D slice of a 3-D tensor\\
   $\displaystyle \erva_i$ & Element $i$ of the random vector $\rva$ \\
\end{tabular}
```

Calculus:

```
\vspace{0.5cm}
\centerline{\bf Calculus}
\begin{tabular}{p{1.45in}p{4in}}
   $\displaystyle\frac{d y} {d x}$ & Derivative of $y$ with respect to
      $x$\\ [2ex]
   $\displaystyle \frac{\partial y} {\partial x} $ & Partial derivative
      of $y$ with respect to $x$ \\
   $\displaystyle \nabla_\vx y $ & Gradient of $y$ with respect to $\vx$
      \\
   $\displaystyle \nabla_\mX y $ & Matrix derivatives of $y$ with respect
       to $\mX$ \\
   $\displaystyle \nabla_\tX y $ & Tensor containing derivatives of $y$
      with respect to $\tX$ \\
   $\displaystyle \frac{\partial f}{\partial \vx} $ & Jacobian matrix $\
      mJ \in \R^{m\times n}$ of $f: \R^n \rightarrow \R^m$\\
   $\displaystyle \nabla_\vx^2 f(\vx)\text{ or }\mH( f)(\vx)$ & The
      Hessian matrix of $f$ at input point $\vx$\\
   $\displaystyle \int f(\vx) d\vx $ & Definite integral over the entire
      domain of $\vx$ \\
   $\displaystyle \int_\sS f(\vx) d\vx$ & Definite integral with respect
      to $\vx$ over the set $\sS$ \\
\end{tabular}
```

Probability and Information Theory:

```
\vspace{0.5cm}
\centerline{\bf Probability and Information Theory}
\begin{tabular}{p{1.45in}p{4in}}
   $\displaystyle P(\ra)$ & A probability distribution over a discrete
      variable\\
   $\displaystyle p(\ra)$ & A probability distribution over a continuous
      variable, or over
   a variable whose type has not been specified\\
```

```
   $\displaystyle \ra \sim P$ & Random variable $\ra$ has distribution
      $P$\\% so thing on left of \sim should always be a random variable,
       with name beginning with \r
   $\displaystyle \E_{\rx\sim P} [ f(x) ]\text{ or } \E f(x)$ &
      Expectation of $f(x)$ with respect to $P(\rx)$ \\
   $\displaystyle \Var(f(x)) $ & Variance of $f(x)$ under $P(\rx)$ \\
   $\displaystyle \Cov(f(x),g(x)) $ & Covariance of $f(x)$ and $g(x)$
      under $P(\rx)$\\
   $\displaystyle H(\rx) $ & Shannon entropy of the random variable $\rx$
      \\
   $\displaystyle \KL ( P \Vert Q ) $ & Kullback-Leibler divergence of P
      and Q \\
   $\displaystyle \mathcal{N} ( \vx ; \vmu , \mSigma)$ & Gaussian
      distribution %
   over $\vx$ with mean $\vmu$ and covariance $\mSigma$ \\
\end{tabular}
```

Functions:

```
\vspace{0.5cm}
\centerline{\bf Functions}
\begin{tabular}{p{1.45in}p{4in}}
   $\displaystyle f: \sA \rightarrow \sB$ & The function $f$ with domain
      $\sA$ and range $\sB$\\
   $\displaystyle f \circ g $ & Composition of the functions $f$ and $g$
      \\
     $\displaystyle f(\vx ; \vtheta) $ & A function of $\vx$ parametrized
         by $\vtheta$.
     (Sometimes we write $f(\vx)$ and omit the argument $\vtheta$ to
        lighten notation) \\
   $\displaystyle \log x$ & Natural logarithm of $x$ \\
   $\displaystyle \sigma(x)$ & Logistic sigmoid, $\displaystyle \frac{1}
      {1 + \exp(-x)}$ \\
   $\displaystyle \zeta(x)$ & Softplus, $\log(1 + \exp(x))$ \\
   $\displaystyle || \vx ||_p $ & $\normlp$ norm of $\vx$ \\
   $\displaystyle || \vx || $ & $\normltwo$ norm of $\vx$ \\
   $\displaystyle x^+$ & Positive part of $x$, i.e., $\max(0,x)$\\
   $\displaystyle \1_\mathrm{condition}$ & is 1 if the condition is true,
       0 otherwise\\
\end{tabular}
```

参考文献

[1] GOODFELLOW I, BENGIO Y, COURVILLE A, et al. Deep learning: vol. 1[M]. MIT Press, 2016.