第一次实验报告
——版本控制 (Git) 和 LaTeX 文档编辑

杨昕昱 23020007142

2024 09

目录

1 调试及性能分析

1.1 journalctl

使用 Linux 上的 journalctl 命令来获取最近一天中超级用户的登录信息及其所执行的指令。

```
yxy@DESKTOP-6MF70N6:~$ journalctl | grep sudo
Aug 30 10:58:48 DESKTOP-6MF70N6 usermod[619]: add 'yxy' to group 'sudo'
Aug 30 10:58:48 DESKTOP-6MF70N6 usermod[619]: add 'yxy' to shadow group 'sudo'
Sep 03 21:43:04 DESKTOP-6MF70N6 sudo[408]: pam_unix(sudo:auth): conversation failed
Sep 03 21:43:04 DESKTOP-6MF70N6 sudo[408]: pam_unix(sudo:auth): auth could not identify
 password for [yxy]
yxy@DESKTOP-6MF70N6:~$
```

1.2 shellcheck

下载 shellcheck , 输入命令 “sudo apt-get install shellcheck”。

```
yxy@DESKTOP-6MF70N6:~$ sudo apt-get update
sudo apt-get install shellcheck
[sudo] password for yxy:
Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1805 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2022 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [352 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.8 kB
]
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2437 kB
]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [295 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.3
 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2377
 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [419 kB
]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1124 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [261 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [26
.2 kB]
Get:17 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [409
kB]
Get:18 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [902 kB
]
Get:19 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [176 kB
]
Get:20 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [
19.2 kB]
```

```
Fetched 12.9 MB in 4s (3030 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  shellcheck
0 upgraded, 1 newly installed, 0 to remove and 53 not upgraded.
Need to get 2359 kB of archives.
After this operation, 16.3 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 shellcheck amd64 0.8.0-2 [2
359 kB]
Fetched 2359 kB in 6s (418 kB/s)
Selecting previously unselected package shellcheck.
(Reading database ... 24206 files and directories currently installed.)
Preparing to unpack .../shellcheck_0.8.0-2_amd64.deb ...
Unpacking shellcheck (0.8.0-2) ...
Setting up shellcheck (0.8.0-2) ...
Processing triggers for man-db (2.10.2-1) ...
```

1.3 cProfile

代码如下：

Ubuntu > home > yxy > sorts.py > quicksort_inplace

```python
import random

def test_sorted(fn, iters=1000):
    for i in range(iters):
        l = [random.randint(0, 100) for i in range(0, random.randint(0, 50))]
        assert fn(l) == sorted(l)
        # print(fn.__name__, fn(l))

@profile
def insertionsort(array):

    for i in range(len(array)):
        j = i-1
        v = array[i]
        while j >= 0 and v < array[j]:
            array[j+1] = array[j]
            j -= 1
        array[j+1] = v
    return array

@profile
def quicksort(array):
    if len(array) <= 1:
        return array
    pivot = array[0]
    left = [i for i in array[1:] if i < pivot]
    right = [i for i in array[1:] if i >= pivot]
    return quicksort(left) + [pivot] + quicksort(right)

```

```python
@profile
def quicksort_inplace(array, low=0, high=None):
    if len(array) <= 1:
        return array
    if high is None:
        high = len(array)-1
    if low >= high:
        return array

    pivot = array[high]
    j = low-1
    for i in range(low, high):
        if array[i] <= pivot:
            j += 1
            array[i], array[j] = array[j], array[i]
    array[high], array[j+1] = array[j+1], array[high]
    quicksort_inplace(array, low, j)
    quicksort_inplace(array, j+2, high)
    return array


if __name__ == '__main__':
    for fn in [quicksort, quicksort_inplace, insertionsort]:
        test_sorted(fn)

```

使用 cProfile 比较插入排序和快速排序的性能。

```
yxy@DESKTOP-6MF70N6:~$ python3 -m cProfile -s time sorts.py
         821320 function calls (754963 primitive calls) in 0.169 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    77705    0.038    0.000    0.079    0.000 random.py:292(randrange)
    77705    0.023    0.000    0.031    0.000 random.py:239(_randbelow_with_getrandbits
)
33522/1000    0.019    0.000    0.030    0.000 sorts.py:23(quicksort)
34808/1000    0.018    0.000    0.020    0.000 sorts.py:32(quicksort_inplace)
     1000    0.015    0.000    0.015    0.000 sorts.py:11(insertionsort)
    77705    0.011    0.000    0.090    0.000 random.py:366(randint)
     3000    0.009    0.000    0.095    0.000 sorts.py:6(<listcomp>)
   233115    0.009    0.000    0.009    0.000 {built-in method _operator.index}
    98486    0.005    0.000    0.005    0.000 {method 'getrandbits' of '_random.Random'
 objects}
    16261    0.005    0.000    0.005    0.000 sorts.py:27(<listcomp>)
    16261    0.005    0.000    0.005    0.000 sorts.py:28(<listcomp>)
    77705    0.003    0.000    0.003    0.000 {method 'bit_length' of 'int' objects}
    70311    0.003    0.000    0.003    0.000 {built-in method builtins.len}
        3    0.003    0.001    0.168    0.056 sorts.py:4(test_sorted)
     3000    0.002    0.000    0.002    0.000 {built-in method builtins.sorted}
        2    0.000    0.000    0.000    0.000 {method 'read' of '_io.BufferedReader' ob
jects}
        2    0.000    0.000    0.000    0.000 {built-in method marshal.loads}
        1    0.000    0.000    0.001    0.001 random.py:1(<module>)
```

可知快速排序的时间更快。

1.4 line_profiler

输入“pip install line_profiler”安装 line_profiler。

```
yxy@DESKTOP-6MF70N6:~$ pip install line_profiler
Defaulting to user installation because normal site-packages is not writeable
Collecting line_profiler
  Downloading line_profiler-4.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (717 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 717.6/717.6 kB 2.7 MB/s eta 0:00:00
Installing collected packages: line_profiler
  WARNING: The script kernprof is installed in '/home/yxy/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed line_profiler-4.1.3
```

使用 line_profiler 比较插入排序和快速排序的性能。然后为需要分析的函数添加装饰器 @profile，并执行：”kernprof -l -v sorts.py”。

```
yxy@DESKTOP-6MF70N6:~$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.078071 s
File: sorts.py
Function: insertionsort at line 9

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     9                                           @profile
    10                                           def insertionsort(array):
    11
    12     25956       2161.9      0.1      2.8      for i in range(len(array)):
    13     24956       2494.1      0.1      3.2          j = i-1
    14     24956       2332.6      0.1      3.0          v = array[i]
    15    222949      28529.2      0.1     36.5          while j >= 0 and v < array[j]:
    16    197993      21702.2      0.1     27.8              array[j+1] = array[j]
    17    197993      17628.9      0.1     22.6              j -= 1
    18     24956       3134.4      0.1      4.0          array[j+1] = v
    19      1000         87.8      0.1      0.1      return array

Total time: 0.0403935 s
File: sorts.py
Function: quicksort at line 21

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    21                                           @profile
    22                                           def quicksort(array):
    23     33518       4249.4      0.1     10.5      if len(array) <= 1:
    24     17259       1289.2      0.1      3.2          return array
    25     16259       1644.5      0.1      4.1      pivot = array[0]
    26     16259      12682.6      0.8     31.4      left = [i for i in array[1:] if i < pivot]
    27     16259      12189.4      0.7     30.2      right = [i for i in array[1:] if i >= pivot]
    28     16259       8338.4      0.5     20.6      return quicksort(left) + [pivot] + quicksort(right)
```

可知插入排序的耗时更高一些。快速排序的瓶颈在于 left 和 right 的赋值，而插入排序的瓶颈在 while 循环。

1.5 memory_profiler

输入“pip install memory_profiler”安装 memory_profiler。

```
yxy@DESKTOP-6MF70N6:~$  pip install memory_profiler
Defaulting to user installation because normal site-packages is not writeable
Collecting memory_profiler
  Downloading memory_profiler-0.61.0-py3-none-any.whl (31 kB)
Collecting psutil
  Downloading psutil-6.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010_x86_64.manylin
ux_2_17_x86_64.manylinux2014_x86_64.whl (290 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 290.5/290.5 KB 112.4 kB/s eta 0:00:00
Installing collected packages: psutil, memory_profiler
Successfully installed memory_profiler-0.61.0 psutil-6.0.0
```

同样需要添加 @profile 装饰器。首先分析快速排序的内存使用情况：

```
yxy@DESKTOP-6MF70N6:~$ python3 -m memory_profiler sorts.py
Filename: sorts.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    21   19.801 MiB   19.801 MiB       33812   @profile
    22                                         def quicksort(array):
    23   19.801 MiB    0.000 MiB       33812       if len(array) <= 1:
    24   19.801 MiB    0.000 MiB       17406           return array
    25   19.801 MiB    0.000 MiB       16406       pivot = array[0]
    26   19.801 MiB    0.000 MiB      158368       left = [i for i in array[1:] if i <
pivot]
    27   19.801 MiB    0.000 MiB      158368       right = [i for i in array[1:] if i >
= pivot]
    28   19.801 MiB    0.000 MiB       16406       return quicksort(left) + [pivot] + q
uicksort(right)
```

然后分析插入排序的内存使用情况：

```
yxy@DESKTOP-6MF70N6:~$ python3 -m memory_profiler sorts.py
Filename: sorts.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     9   19.988 MiB   19.941 MiB        1000   @profile
    10                                         def insertionsort(array):
    11
    12   19.988 MiB    0.047 MiB       25700       for i in range(len(array)):
    13   19.988 MiB    0.000 MiB       24700           j = i-1
    14   19.988 MiB    0.000 MiB       24700           v = array[i]
    15   19.988 MiB    0.000 MiB      225524           while j >= 0 and v < array[j]:
    16   19.988 MiB    0.000 MiB      200824               array[j+1] = array[j]
    17   19.988 MiB    0.000 MiB      200824               j -= 1
    18   19.988 MiB    0.000 MiB       24700           array[j+1] = v
    19   19.988 MiB    0.000 MiB        1000       return array
```

同时对比原地操作的快速排序算法内存情况：

```
yxy@DESKTOP-6MF70N6:~$  python3 -m memory_profiler sorts.py
Filename: sorts.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    30   19.840 MiB   19.840 MiB       34004   @profile
    31                                         def quicksort_inplace(array, low=0, high
=None):
    32   19.840 MiB    0.000 MiB       34004       if len(array) <= 1:
    33   19.840 MiB    0.000 MiB          45           return array
    34   19.840 MiB    0.000 MiB       33959       if high is None:
    35   19.840 MiB    0.000 MiB         955           high = len(array)-1
    36   19.840 MiB    0.000 MiB       33959       if low >= high:
    37   19.840 MiB    0.000 MiB       17457           return array
    38
    39   19.840 MiB    0.000 MiB       16502       pivot = array[high]
    40   19.840 MiB    0.000 MiB       16502       j = low-1
    41   19.840 MiB    0.000 MiB      125417       for i in range(low, high):
    42   19.840 MiB    0.000 MiB      108915           if array[i] <= pivot:
    43   19.840 MiB    0.000 MiB       56429               j += 1
    44   19.840 MiB    0.000 MiB       56429               array[i], array[j] = array[j
], array[i]
    45   19.840 MiB    0.000 MiB       16502       array[high], array[j+1] = array[j+1]
, array[high]
    46   19.840 MiB    0.000 MiB       16502       quicksort_inplace(array, low, j)
    47   19.840 MiB    0.000 MiB       16502       quicksort_inplace(array, j+2, high)
    48   19.840 MiB    0.000 MiB       16502       return array
```

1.6 停止监听进程

首先执行 python -m http.server 4444 启动一个最简单的 web 服务器来监听 4444 端口。在另外一个终端中，执行 lsof | grep LISTEN 打印出所有监听端口的进程及相应的端口。找到对应的 PID 然后使用 kill <PID> 停止该进程。

```
yxy@DESKTOP-6MF70N6:~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
Terminated
yxy@DESKTOP-6MF70N6:~$ lsof | grep LISTEN
python3   10464                    yxy    3u     IPv4           364045      0t0    TCP *:4444 (LISTEN)
yxy@DESKTOP-6MF70N6:~$ kill 10464
```

2 元编程

2.1 安装 LaTeX

对于 Ubuntu/Debian 系统：

sudo apt-get update

sudo apt-get install texlive-full

```
Setting up context-modules (20210301-1) ...
Setting up texlive-full (2021.20220204-1) ...
Processing triggers for libglib2.0-0:amd64 (2.72.4-0ubuntu2.3) ...
Setting up libgtk-3-0:amd64 (3.24.33-1ubuntu2.2) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
Setting up libgtk-3-bin (3.24.33-1ubuntu2.2) ...
Setting up libvte-2.91-0:amd64 (0.68.0-1ubuntu0.1) ...
Processing triggers for man-db (2.10.2-1) ...
Setting up qt5-gtk-platformtheme:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Processing triggers for udev (249.11-0ubuntu3.11) ...
Setting up libvted-3-0:amd64 (3.10.0-1ubuntu1) ...
Setting up at-spi2-core (2.44.0-3) ...
Setting up libgtkd-3-0:amd64 (3.10.0-1ubuntu1) ...
Processing triggers for install-info (6.8-4build1) ...
Setting up tilix (1.9.4-2build1) ...
Processing triggers for tex-common (6.17) ...
Running updmap-sys. This may take some time... done.
Running mktexlsr /var/lib/texmf ... done.
Building format(s) --all.
        This may take some time... done.
Processing triggers for sgml-base (1.30) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.8+dfsg-1ubuntu0.3) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
```

2.2 编写 Makefile

用 vim Makefile 打开文件并写入内容：

```
# 定义 LaTeX 编译器（尽管在这个规则中我们没有直接使用它，但可以作为参考）
PDFLATEX := pdflatex

# 默认目标
all: paper.pdf

# 编译 LaTeX 文档
paper.pdf: paper.tex plot-data.png
        pdflatex paper.tex

# 生成图像文件
# 使用模式规则来匹配任何以 .dat 结尾的文件并生成相应的 .png 文件
plot-%.png: %.dat plot.py
        ./plot.py -i $< -o $@

# 清理生成的文件（可选）
clean:
        rm -f *.aux *.log *.pdf *.png

# 深度清理（可选），通常与 clean 相同，但可以根据需要添加更多文件
deep-clean: clean

.PHONY: all clean deep-clean
~
```

2.3 创建依赖文件

要构建 plot-data.png ，要先创建 paper.tex 、plot.py 、data.dat ，写入以下内容：

```
yxy@DESKTOP-6MF70N6:~$ cat paper.tex
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\includegraphics[scale=0.65]{plot-data.png}
\end{document}
```

```
yxy@DESKTOP-6MF70N6:~$ cat plot.py
#!/usr/bin/env python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('-i', type=argparse.FileType('r'))
parser.add_argument('-o')
args = parser.parse_args()

data = np.loadtxt(args.i)
plt.plot(data[:, 0], data[:, 1])
plt.savefig(args.o)
```

```
yxy@DESKTOP-6MF70N6:~$ cat data.dat
1 1
2 2
3 3
4 4
5 8
```

2.4 make 命令

成功执行 make 命令，输出结果如下：

```
yxy@DESKTOP-6MF70N6:~$ make
python3 ./plot.py -i data.dat -o plot-data.png
pdflatex paper.tex
This is pdfTeX, Version 3.141592653-2.6-1.40.22 (TeX Live 2022/dev/Debian) (preloaded format=pdflatex)
 restricted \write18 enabled.
entering extended mode
(./paper.tex
LaTeX2e <2021-11-15> patch level 1
L3 programming layer <2022-01-21>
(/usr/share/texlive/texmf-dist/tex/latex/base/article.cls
Document Class: article 2021/10/04 v1.4n Standard LaTeX document class
(/usr/share/texlive/texmf-dist/tex/latex/base/size10.clo))
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphicx.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/keyval.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphics.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/trig.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-cfg/graphics.cfg)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-def/pdftex.def)))
(/usr/share/texlive/texmf-dist/tex/latex/l3backend/l3backend-pdftex.def)
No file paper.aux.
(/usr/share/texlive/texmf-dist/tex/context/base/mkii/supp-pdf.mkii
[Loading MPS to PDF converter (version 2006.09.02).]
) (/usr/share/texlive/texmf-dist/tex/latex/epstopdf-pkg/epstopdf-base.sty
(/usr/share/texlive/texmf-dist/tex/latex/latexconfig/epstopdf-sys.cfg))
[1{/var/lib/texmf/fonts/map/pdftex/updmap/pdftex.map} <./plot-data.png>]
(./paper.aux) )</usr/share/texlive/texmf-dist/fonts/type1/public/amsfonts/cm/cm
r10.pfb>
Output written on paper.pdf (1 page, 21223 bytes).
Transcript written on paper.log.
```

3 大杂烩

3.1 VPN

连接一个 VPN，如登录中国海洋大学的 VPN 后，可以使用学校相关网站：

3.2 Markdown

可以在 vs code 中安装 Markdown 插件，编辑并预览 Markdown 文档。以下是我用 Markdown 编辑过的实验报告及预览：


Test6 23020007142 杨昕昱.md
扩展: Markdown All in One
作业 > 计导作业 > Test6 > Test6 23020007142 杨昕昱.md > # 实验报告 > ## 三、实验中遇到的问题及解决方法
1 # 实验报告
2 ## 一、实验内容
3 >随机生成100000个随机数，进行冒泡排序和快速排序，并比较执行时间
4 ## 二、实验过程
5 #### 1.C语言
6 ![Alt text](%E5%B1%8F%E5%B9%95%E6%88%AA%E5%9B%BE(35).png)
7 ![Alt text](%E5%B1%8F%E5%B9%95%E6%88%AA%E5%9B%BE(34).png)
8 >比较可知快速排序比冒泡排序快
9 #### 2.python语言(在虚拟机上进行)
10 ![Alt text](%E5%B1%8F%E5%B9%95%E6%88%AA%E5%9B%BE(38).png)
11 >比较可知快速排序比冒泡排序快
12 ## 三、实验中遇到的问题及解决方法
13 * 第一项
14 > 问题：代码出现错误
15 > 解决方法：根据调试修改
16 * 第二项
17 > 问题：在Markdown中无法插入图片
18 > 解决方法：从网上查找方法，再复制图片路径

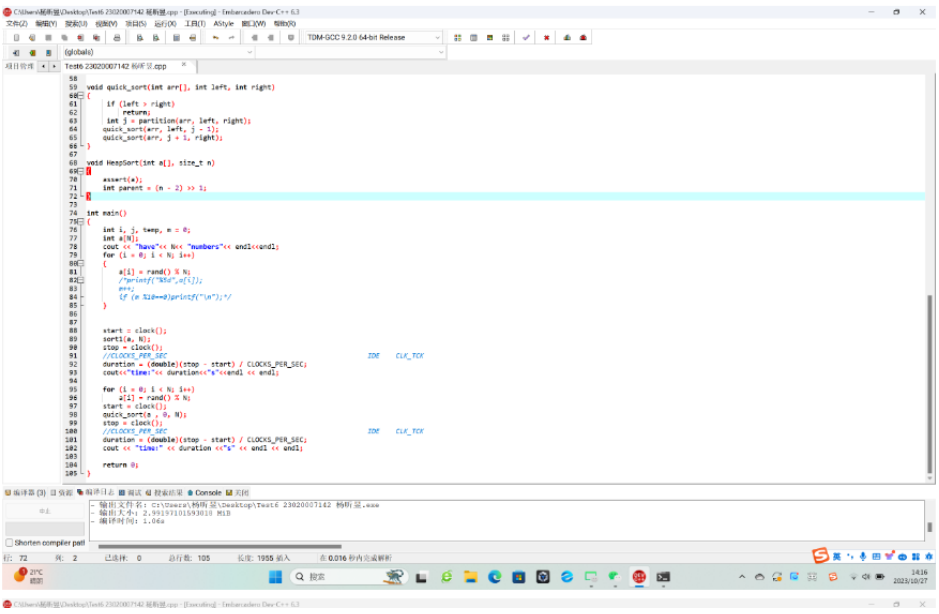
实验报告

一、实验内容

> 随机生成100000个随机数，进行冒泡排序和快速排序，并比较执行时间

二、实验过程

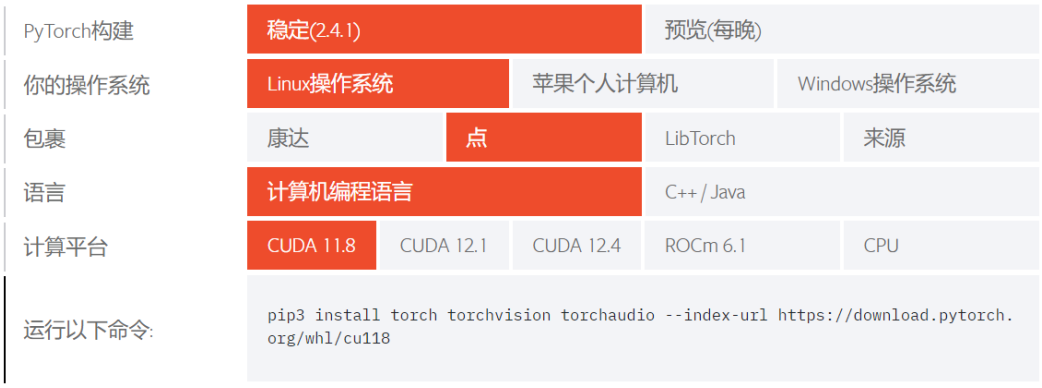
1.C语言



4 PyTorch

4.1 安装

pytorch 的安装可以直接查看官网教程，如下所示，官网地址：https://pytorch.org/get-started：



输入命令“pip3 install torch torchvision torchaudio –index-url https://download.pytorch.org/
若成功输出，则安装成功：

```
yxy@DESKTOP-6MF70N6:~$ python3
Python 3.10.12 (main, Jul 29 2024, 16:56:48) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from __future__ import print_function
>>> import torch

>>> x = torch.rand(5, 3)
>>> print(x)
tensor([[0.6529, 0.5893, 0.8902],
        [0.8479, 0.6867, 0.0945],
        [0.7320, 0.4229, 0.6311],
        [0.6193, 0.1069, 0.3787],
        [0.3691, 0.9890, 0.6012]])
>>>
```

然后是验证能否正确运行在 GPU 上，输入下列代码，这份代码中 cuda.is_available() 主要是用于检测是否可以使用当前的 GPU 显卡，如果返回 True，当然就可以运行，否则就不能。

```
>>> import torch
able()>>> torch.cuda.is_available()
True
>>>
```

4.2 张量 (Tensors) 声明与定义

首先导入必须的库，主要是 torch：

```
from ___futur___ import print_function
import torch
```

· torch.empty(): 声明一个未初始化的矩阵
· torch.rand()：随机初始化一个矩阵
· torch.zeros()：创建数值皆为 0 的矩阵
· torch.tensor()：直接传递 tensor 数值来创建
· tensor.new_ones()：new_*() 方法需要输入尺寸大小
· torch.randn_like(old_tensor)：保留相同的尺寸大小
· 对 tensors 的尺寸大小获取可以采用 tensor.size() 方法

```
>>> from __future__ import print_function
ch>>> import torch
>>> # 创建一个 5*3 的矩阵
, 3)
print(x)>>> x = torch.empty(5, 3)
>>> print(x)
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
>>> # 创建一个随机初始化的 5*3 矩阵
>>> rand_x = torch.rand(5, 3)
int(rand_x)>>> print(rand_x)
tensor([[0.3968, 0.6619, 0.0438],
        [0.8008, 0.9395, 0.9160],
        [0.9517, 0.7149, 0.9841],
        [0.1242, 0.7731, 0.7991],
        [0.7363, 0.7235, 0.5987]])
>>> # 创建一个数值皆是 0, 类型为 long 的矩阵
, 3, dtype=torch>>> zero_x = torch.zeros(5, 3, dtype=torch.long)
o_x)>>> print(zero_x)
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
>>> tensor1 = torch.tensor([5.5, 3])
>>> print(tensor1)
tensor([5.5000, 3.0000])
>>> # 显示定义新的尺寸是 5*3, 数值类型是 torch.double
>>> tensor2 = tensor1.new_ones(5, 3, dtype=torch.double)  # new_* 方法需要输入 tensor
大小
>>> print(tensor2)
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], dtype=torch.float64)
>>> # 修改数值类型
ike(tensor2, dty>>> tensor3 = torch.randn_like(tensor2, dtype=torch.float)
print('tensor3: ', tensor3)>>> print('tensor3: ', tensor3)
tensor3:  tensor([[-0.2037, -1.8464,  0.9095],
        [-0.3918,  1.0885,  0.5484],
        [-0.5382,  0.1352,  1.1569],
        [-0.3856, -0.4709,  0.5892],
        [ 0.7437,  0.1749, -1.8771]])
>>> print(tensor3.size())
torch.Size([5, 3])
>>>
```

4.3 张量 (Tensors) 操作

对于加法的操作，有几种实现方式：

· + 运算符

· torch.add(tensor1, tensor2, [out=tensor3])

· tensor1.add_(tensor2)：直接修改 tensor 变量

```
>>> tensor4 = torch.rand(5, 3)
or4= ', tensor3 >>> print('tensor3 + tensor4= ', tensor3 + tensor4)
tensor3 + tensor4=  tensor([[ 0.6479, -0.5345,  2.1223],
        [ 0.3184,  2.7632,  1.3258],
        [ 0.3233,  0.9963,  2.7655],
        [ 0.6517,  0.5750,  2.0614],
        [ 1.5066,  1.4451, -0.6034]])
>>> result = torch.empty(5, 3)
t)
print('add re>>> torch.add(tensor3, tensor4, out=result)
tensor([[ 0.6479, -0.5345,  2.1223],
        [ 0.3184,  2.7632,  1.3258],
        [ 0.3233,  0.9963,  2.7655],
        [ 0.6517,  0.5750,  2.0614],
        [ 1.5066,  1.4451, -0.6034]])
>>> print('add result= ', result)
add result=  tensor([[ 0.6479, -0.5345,  2.1223],
        [ 0.3184,  2.7632,  1.3258],
        [ 0.3233,  0.9963,  2.7655],
        [ 0.6517,  0.5750,  2.0614],
        [ 1.5066,  1.4451, -0.6034]])
>>> tensor3.add_(tensor4)
tensor([[ 0.6479, -0.5345,  2.1223],
        [ 0.3184,  2.7632,  1.3258],
        [ 0.3233,  0.9963,  2.7655],
        [ 0.6517,  0.5750,  2.0614],
        [ 1.5066,  1.4451, -0.6034]])
>>> print('tensor3= ', tensor3)
tensor3=  tensor([[ 0.6479, -0.5345,  2.1223],
        [ 0.3184,  2.7632,  1.3258],
        [ 0.3233,  0.9963,  2.7655],
        [ 0.6517,  0.5750,  2.0614],
        [ 1.5066,  1.4451, -0.6034]])
>>>
```

对于 Tensor 的访问，和 Numpy 对数组类似，可以使用索引来访问某一维的数据，对 Tensor 的尺寸修改，可以采用 torch.view()，如下所示：

```
>>> print(tensor3[:, 0])
tensor([0.6479, 0.3184, 0.3233, 0.6517, 1.5066])
>>> x = torch.randn(4, 4)
(16)
# -1 表示除给定维>>> y = x.view(16)
度外的其余维度的乘积
z = x.view(-1, 8)
print(x.size(), y.size(), z.size())>>> # -1 表示除给定维度外的其余维度的乘积
>>> z = x.view(-1, 8)
>>> print(x.size(), y.size(), z.size())
torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
>>>
```

如果 tensor 仅有一个元素，可以采用.item() 来获取类似 Python 中整

数类型的数值：

```
>>> x = torch.randn(1)
>>> print(x)
t(x.item())tensor([0.7132])
>>> print(x.item())
0.7131855487823486
```

4.4 Tensor 转换为 Numpy 数组

调用 tensor.numpy() 可以实现这个转换操作。

```
>>> a = torch.ones(5)
)>>> print(a)
tensor([1., 1., 1., 1., 1.])
>>> b = a.numpy()
>>> print(b)
[1. 1. 1. 1. 1.]
```

两者是共享同个内存空间的，b 随着 a 的改变而改变。

```
>>> a.add_(1)
tensor([2., 2., 2., 2., 2.])
>>> print(a)
tensor([2., 2., 2., 2., 2.])
>>> print(b)
[2. 2. 2. 2. 2.]
```

4.5 Numpy 数组转换为 Tensor

转换的操作是调用 torch.from_numpy(numpy_array) 方法。

```
>>> import numpy as np
>>> a = np.ones(5)
>>> b = torch.from_numpy(a)
(a, 1, out=a)
print(a)
print(b)>>> np.add(a, 1, out=a)
array([2., 2., 2., 2., 2.])
>>> print(a)
[2. 2. 2. 2. 2.]
>>> print(b)
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
>>>
```

4.6 CUDA 张量

Tensors 可以通过.to 方法转换到不同的设备上，即 CPU 或者 GPU 上。

```
>>> # 当 CUDA 可用的时候，可用运行下方这段代码，采用 torch.device() 方法来改变 tensors
是否在 GPU 上进行计算操作
ch.cuda.is_available():
    device = torch.devic>>> if torch.cuda.is_available():
e("cuda")          # 定义一个 CUDA 设备对象
    y = torch.ones_like(x, device=device)  #...     device = torch.device("cuda")
    # 定义一个 CUDA 设备对象
tensor
    x = x...     y = torch.ones_like(x, device=device)  # 显示创建在 GPU 上的一个 tenso
r
.to(device)                       # 也可以采用 .to("c...     x = x.to(device)
            # 也可以采用 .to("cuda")
...     z = x + y
...     print(z)
...     print(z.to("cpu", torch.double))       # .to() 方法也可以改变数值类型
...
tensor([1.7132], device='cuda:0')
tensor([1.7132], dtype=torch.float64)
>>>
```

输出结果，第一个结果就是在 GPU 上的结果，打印变量的时候会带有 device=’cuda:0’，而第二个是在 CPU 上的变量。

4.7 autograd 张量

首先导入必须的库，开始创建一个 tensor，并让 requires_grad=True 来追踪该变量相关的计算操作。

```
>>> import torch
>>> x = torch.ones(2, 2, requires_grad=True)
>>> print(x)
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> y = x + 2
>>> print(y)
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
>>> print(y.grad_fn)
<AddBackward0 object at 0x7fcec76110c0>
>>> z = y * y * 3
>>> out = z.mean()
>>> print('z=', z)
z= tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>)
>>> print('out=', out)
out= tensor(27., grad_fn=<MeanBackward0>)
>>>
```

实际上，一个 Tensor 变量的默认 requires_grad 是 False ，可以像上述定义一个变量时候指定该属性是 True，当然也可以定义变量后，调用.requires_grad_(True) 设置为 True。

4.8 autograd 梯度

接下来就是开始计算梯度，进行反向传播的操作。out 变量是上一小节中定义的，它是一个标量，因此 out.backward() 相当于 out.backward(torch.tensor(1.)):

```
>>> out.backward()
>>> print(x.grad)
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
>>> x = torch.randn(3, requires_grad=True)
>>> y = x * 2
>>> while y.data.norm() < 1000:
...     y = y * 2
```

```
>>> print(y)
tensor([-0.4611, -3.4808,  3.4254], grad_fn=<MulBackward0>)
>>> v = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
>>> y.backward(v)
>>> print(x.grad)
tensor([2.0000e-01, 2.0000e+00, 2.0000e-04])
>>> print(x.requires_grad)
True
>>> print((x ** 2).requires_grad)
True
>>> with torch.no_grad():
...     print((x ** 2).requires_grad)
...
False
>>>
```