目录

导言

这份文档主要用来存放一些实际工作中碰到的实用的代码片段，可能包含 MATLAB、Python、C/C++ 和一些 LaTeX 的小知识。个人笔记，个人娱乐。

如果有人想编译这份手册或想学习一下实现，请务必读以下说明。

字体设置，为了避免侵权，尽可能使用开源字体[①]。

- Source Han Sans: https://github.com/adobe-fonts/source-han-sans/tree/release
- Source Han Serif: https://github.com/adobe-fonts/source-han-serif/tree/release
- Source Code Pro: https://github.com/adobe-fonts/source-code-pro
- PT Sans Narrow: https://fonts.google.com/specimen/PT+Sans+Narrow
- TeX Gyre: 有问题前往https://www.ctan.org获取，一般来说 TeX 发行版自带
- 等宽字体：大多数等宽字体都是程序员使用的，开源居多，颇易获取。我常用 DejaVu Sans Mono，Fira Code 和 Source Code Pro 三种。

```
%% 字体设置
\usepackage{fontspec}
    \setmainfont{Adobe Garamond Pro}  % TeX Gyre Pagella
    \setsansfont{TeX Gyre Heros}
    \setmonofont{Source Code Pro}  % Consolas, DejaVu Sans Mono
    \setCJKmainfont[BoldFont={Source Han Sans SC}, ItalicFont={KaiTi}]{Source
    ↪ Han Serif SC}
    \setCJKmonofont{FangSong}
    \setCJKsansfont{Source Han Sans SC}

%% 数学字体
\usepackage{unicode-math}
    \setmathfont[math-style = ISO, bold-style = ISO]{TeX Gyre Pagella Math}

%% url 样式
\newfontfamily\urlfont{PT Sans Narrow}
```

编译环境设置，代码高亮环境由 minted 宏包提供（需要 Python 环境）。

代码测试环境，各种代码的运行环境为 MATLAB 2017b、Anaconda、Visual Studio 2017 community、MiKTeX（各宏包均为最新）。

如果你觉得本文档里的代码有用，请不要直接复制文档里面的代码（直接复制会复制到换行产生的符号及空格，可能导致代码出现难以预计的错误），请到github 项目的 code 文件夹找对应的文件。

[①] 西文主字体 Adobe Garamond Pro、楷体、仿宋暂时没有找到理想的替代方案

第一章 MATLAB

1 将内置函数背下来

在 MATLAB 中工作时，很多操作其实都是有内置函数的，对 MATLAB 不熟悉就用不成，然后就“曲线救国”了，效率不高且不说，关键自己实现很费脑！这里收集一些数值计算工作中极其常用的函数。

`meshgrid`

2 如何遍历当前文件夹及其子文件夹中的全部文件？

假设现在我们有这样一个文件夹 A，它含有一些文件和子文件夹 B、C、D……，这些子文件夹又包含若干层子文件夹。我们需要将这个父文件夹（A）及其子文件夹（B、C、D……）和孙文件夹中的所有文件名和其路径取出来。

如果你用的是 MATLAB 2016b 及更新的版本，那真的太棒了！`dir()` 函数已经支持遍历搜索了。尝试敲入：

```
dir_data = dir('**/*');
dir_data([dir_data.isdir]) = [];  % 去除所有 . 和 .. 文件夹
```

这将会返回一个包含文件信息的 struct，现在你可以任意操作这些 struct 了，随意拼接路径。解放大脑，哦也！方便归方便，但是，一来肯定有大多数人使用的是 MATLAB 2016b 之前的版本，二来，解放大脑意味着我失去了一次独立思考的机会。

思考

对于实现方法[①]，多层次的遍历，我第一时间想到的是递归。然后就是数据的存储了，`dir()` 函数返回的是一个 struct，这个数据结构储存有文件的信息，我们要充分利用这个数据结构。所以现在思路是，写一个递归函数，这个函数返回包含所有文件信息的 struct。

这个函数应对先处理父文件夹，获取文件和子文件夹，然后储存文件信息，同时去除子文件夹中的‘.’和‘..’这两个特殊文件夹。我们对获取的子文件夹再次调用该函数，并储存文件信息。如此，利用递归获取子子孙孙无穷尽文件夹的信息[②]，最后函数返回存储有所有文件信息的 struct。现在，你可以对这个结构体做你想做的事情。

解

MATLAB 2016b 以上的版本我们可以用函数返回 struct，这个数据结构包含 [folder, name, date, bytes, isdir, datenum] 六个字段的信息，我们可以按自己意愿使用 folder 和 name 拼接出文

[①] 思路来源：How to get all files under a specific directory in MATLAB?

[②] 其实这并不可能，因为递归是有栈高度限制的，调用函数压入栈，返回函数弹出栈，如果文件夹层次太深，一直压栈就会到达栈溢出警告的极限，例如 Python 的栈往往是 100 层，我想 MATLAB 的栈也大致如此，不会太高

件的完整路径。

```
% get all file name in current dir and sub dir, Compatible with MATLAB R2016b
 ↪ and newer
function file_list = get_all_file_name_R2016b_newer(path)

dir_data = dir(path);
file_list = dir_data(~[dir_data.isdir]);  % file name of current dir

% get sub dir information
sub_dir = dir_data([dir_data.isdir]);  % struct
dot_dir = ismember({sub_dir.name}, {'.', '..'});  % logical
sub_dir = sub_dir(~dot_dir);  % struct, remove specific folder

% recursion
for i = 1:length(sub_dir)
    next_dir = fullfile(sub_dir(i).folder, sub_dir(i).name);  % str
    file_list = [file_list; get_all_file_name_R2016b_newer(next_dir)];  %
     ↪ struct
end

end
```

MATLAB 2016a 及之前的版本 dir struct 信息并不包含 folder，如果返回 struct，将只有文件的 [name, date, bytes, isdir, datenum] 五个字段的信息，所以我们并不能根据函数返回的 struct 拼接出文件完整路径，**我们需要自己将路径拼接成一个 cell，然后使用函数返回 cell**。

```
% get all file name in current dir and sub dir, Compatible with MATLAB R2016a
 ↪ and older
function file_list = get_all_file_name_R2016a_older(path)

% file name of current dir
dir_data = dir(path);
file_list_struct = dir_data(~[dir_data.isdir]);
file_list = fullfile(path, {file_list_struct.name})';

% get sub dir information
sub_dir = dir_data([dir_data.isdir]); % struct
dot_dir = ismember({sub_dir.name}, {'.', '..'}); % logical
sub_dir = sub_dir(~dot_dir); % struct, remove specific folder

% recursion
for i = 1:length(sub_dir)
    next_dir = fullfile(path, sub_dir(i).name); % str
```

```
    file_list = [file_list; get_all_file_name_R2016a_older(next_dir)]; %
    ↪ struct
end

end
```

总结

`dir()` 函数遍历整个 F 盘共 2 万余文件文件大约需要 1.555823s。我们实现的递归函数遍历 F 盘文件大约需要 3.703009s。慢是慢了点，但我们成功运用了递归解决问题，不是吗?

3 如何按自然顺序排序字符串?

通常,我们会遇到处理一系列文件名有规律的文件的情况,比如:a1.txt、a2.txt ..... a100.txt。但是，当读取文件名到一个 cell 里后，我们发现文件名往往是乱序排列的，甚至当你使用 `sort` 函数后，排序也不会改变。搜索了一下，在 Mathworks File Exchange 网站找到了一个自然排序的函数[③]，感谢作者 Stephen Cobeldick。效果如下:

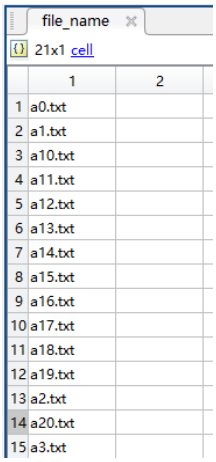

图 1.1: 乱序的文件名

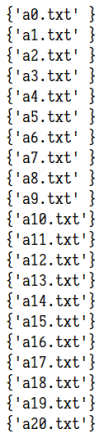

图 1.2: 排序后自然顺序的文件名

4 如何隔行取数据?

闭上眼睛，想象现在有这样一个数组 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`，我们要隔一列取一个数据，或者隔两列取一个数据。得益于 MATLAB 的向量化编程，我们可以很方便的做到,

```
mat_a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
mat_b = mat_a(:, 1:2:length(mat_a));
```

如果你用循环，那么你的代码就不优雅，另，向量化操作比循环快，大型数组优势明显。以上。

[③] https://cn.mathworks.com/matlabcentral/fileexchange/34464-customizable-natural-order-sort

5 如何在遍历数组的同时删除被遍历过的元素?

闭上眼睛，想象现在有这样一个数组 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`，我们需要边遍历元素边删除元素。实现方法和 Python 章节方法一致。

```
mat_a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

while ~isempty(mat_a)
    fprintf("The element being traversed is %d\n", mat_a(1));
    mat_a(1) = [];
    disp(mat_a);
end
```

6 再谈向量化操作

今天又碰到一个数组操作的问题，同样，如果用一般的方法来解决，代码是很冗长的，向量化操作再次助我一臂之力。

有一个 2 列的数组 `all_data`，第一列有正有负，我们称第一列为 x，第二列为 y。现在需要索引 $x > 0$ 时对应的 $[x, y]$ 为一个新的数组 `a`。并且需要从 `a` 中返回 $y = \min(y)$ 时所对应的数组 $[x_0, y_0]$。

```
... ...
-1.44319267634370e-06 9.80637785912817e-06
-1.68967863180042e-07 9.73806551956721e-06
-6.45218837777561e-07 9.75074561060079e-06
6.28923217787410e-07 9.75037059307950e-06
1.54045071473931e-07 9.73772289244816e-06
1.42591401762642e-06 9.80510552313044e-06
... ...
```

先谈向量化获取数组 `a`，利用逻辑索引，保证 x 全大于 0，并取出 1、2 两列；然后利用 `find` 获取 $y = \min(y)$ 的行索引；最后利用索引轻松找到需要的数据。可能看起来比较难理解，但是此时再在外面套文件操作的循环等循环操作是不是清晰多了。

```
a = all_data(all_data(:, 1) > 0, 1:2);

[r, ~] = find(a(:, 2) == min(a(:, 2)));
what_is_i_need = a(r, c);
```

```
clear
clc

file_name_struct = dir('./0518*.txt');
file_name = {file_name_struct(:).name};
```

```
file_name = natsort(file_name);
what_is_i_need = [];

for file_num = 1:length(file_name)
    all_data = load(file_name{file_num});
    a = all_data(all_data(:, 1) > 0, 1:2);

    [r, ~] = find(a(:, 2) == min(a(:, 2)));  %
    what_is_i_need = [what_is_i_need; a(r, 1), a(r, 2)];
end

plot(what_is_i_need(:,2))
```

再来记录一个问题：循环操作里面有一个的 2 列数组 `all_data`，每次循环取第一列中与 0.5 最接近的数据和对应的列，所以该数组大小会不断变，设其维度为 $n \times 2$。如果 $n < 2$，我们将数据置为 0 并保存到一个新数组里面去；如果 $n >= 2$，保存其最小值和最大值到新数组里面去。同样，利用向量化操作最大程度减少代码量。

```
pos = [];
for i = 1:length(time)
    % [x, phi]
    all_data = load(strcat(mph_file, '\', num2str(i), '.txt'));
    all_data = all_data(abs(all_data(:, 2)-0.5) < 0.01, :);
    [r, c] = size(all_data);
    if r == 0 || r == 1
        x_min = 0;
        x_max = 0;
        phi = 0;
        pos = [pos; time(i), x_min, phi; time(i), x_max, phi];
    else
        x_min = all_data(all_data(:, 1) == min(all_data(:, 1)));
        x_max = all_data(all_data(:, 1) == max(all_data(:, 1)));
        [r, ~] = find(all_data == x_min);
        phi_min = all_data(r, 2);
        [r, ~] = find(all_data == x_max);
        phi_max = all_data(r, 2);
        pos = [pos; time(i), x_min, phi_min; time(i), x_max, phi_max];
    end
end
```

7 文件读取

`csvread` 适合读取纯 Comma-Separated Values 文件，`load` 适合读取带注释的 Comma-Separated Values 文件（示例如下），其在读取过程中会自动忽略 csv 文件的注释。

```
% x                      y                      IsoLevel
-1.348651530446577E-5    1.798983884698175E-5    0.5
-1.4987361701783775E-5   2.4219367655756994E-5   0.5
-1.494145158530118E-5    2.3443649068772022E-5   0.5
... ...
```

8 如何将两个维度不一致的矩阵串联起来?

现有矩阵 `a = [1]` 和矩阵 `b = b = [5; 9; 4; 4]`，将其横向拼接成一个矩阵 `c`，

```
c = [1, 5;
    1, 9;
    1, 4;
    1, 4]
```

思路很简单，因为 `a` 的维度不够，所以将其扩维，然后拼接。

```
a = [1];
b = [5; 9; 4; 4];
[r, ~] = size(b);
c = [repmat(a, r, 1), b];
```

9 颜色的处理

Hex 是一种常用的十六进制颜色码，MATLAB 并不能识别，从 MathWorks File Exchange 找了一个 `rgb2hex and hex2rgb` 的函数[④]，另，MathWorks File Exchange 真特么是个宝库，缺什么找什么，一找一个准。

有时候需要将 `double` 类型的矩阵转换成 rgb 色值的三维矩阵，没有 MATLAB 内置函数能做到，在 MathWorks File Exchange 上找了个 `double2rgb` 函数[⑤]可以很方便的做到这一点。

10 谈谈 MATLAB 中的图形对象

可视化是一项很费时费力的工作，MATLAB 可视化成本更高，由于参数混杂，很难进行快速调整。我脑子不好使，先记录一下 MATLAB 基本图像元素的构成，更详细介绍请查阅 MATLAB 帮助文档[⑥]。

重点关注两类对象，顶层对象 `Root, Figure, Axes` 和插图对象 `Colorbar, Legend`。图形对象相关的函数如Figure 1.3所示，用于查找、复制和删除图形对象。

```
r = groot;
fig = figure;
ax = gca;
```

④ https://ww2.mathworks.cn/matlabcentral/fileexchange/46289-rgb2hex-and-hex2rgb
⑤ https://www.mathworks.com/matlabcentral/fileexchange/30264-double2rgb
⑥ https://ww2.mathworks.cn/help/matlab/graphics-objects.html

gca	当前坐标区或图
gcf	当前图窗的句柄
gcbf	包含正在执行其回调的对象的图窗句柄
gcbo	正在执行其回调的对象的句柄
gco	当前对象的句柄
groot	图形根对象
ancestor	图形对象的父级
allchild	查找指定对象的所有子级
findall	查找所有图形对象
findobj	查找具有特定属性的图形对象
findfigs	查找可见的屏幕外图窗
gobjects	初始化图形对象的数组
isgraphics	对有效的图形对象句柄为 True
ishandle	测试是否有效的图形或 Java 对象句柄
copyobj	复制图形对象及其子级
delete	删除文件或对象

图 1.3: MATLAB 中的图形对象的标识

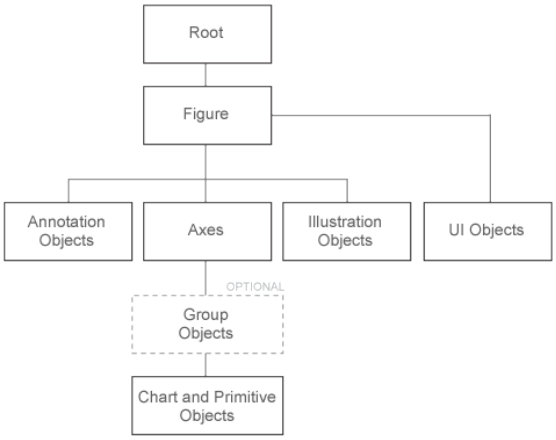

图 1.4: MATLAB 中的图形对象

```
c = colorbar;
lgd = legend('a','b','c');
```

新版本 MATLAB 推荐使用调用对象的方式修改图形对象属性，老版本用 `get, set` 函数分别查阅和修改对象属性。

```
p = plot(1:10);
p.Color = 'r';
set(p, 'Color', 'red');
```

1.10.1 子图

`subplot` 函数可以画子图，有两种调用方式，

```
subplot(m,n,p)
subplot('Position',pos)
```

第一种方式是创建一个 $m \times n$ 的网格，并在第 p 个网格上绘图；第二种方式是在指定 `pos` 上绘图，座标属性为 [left bottom width height]。可以通过查阅图像对象来获取图形属

性。

11 函数！函数！

这里收集几个关于函数的最佳实践，首先是函数的返回值问题。很多时候，MATLAB 函数的返回值不止一个，通常让变量一一返回，在外部使用变量一一接收。最佳做法是将需要返回的变量装入`struct`中，这样在外部仅使用一个变量就能接收所有返回值。

12 如何在矩阵中插入一行/一列

看了一圈，只能通过拼接矩阵来实现，并且 MATLAB 自己没有这样的函数。自己写了一个。

```
% 按行插入
function mat = row_insert(mat_row, pos, mat_add)
[~, c_row] = size(mat_row);
[~, c_add] = size(mat_add);
if c_row == c_add
    mat = [mat_row(1:pos-1, :); mat_add; mat_row(pos:end, :)];
else
    error(' 要插入的矩阵列要与原始矩阵一致')
end
end

% 按列插入
function mat = col_insert(mat_row, pos, mat_add)
[r_row, ~] = size(mat_row);
[r_add, ~] = size(mat_add);
if r_row == r_add
    mat = [mat_row(:, 1:pos-1), mat_add, mat_row(:, pos:end)];
else
    error(' 要插入的矩阵行要与原始矩阵一致')
end
end
```

13 图形处理

1.13.1 基本法

1.13.2 图形插值

图形插值就是让模糊的图形（像素较少）变成比较清楚的图形（像素变多），稍微科学一点的说法是，图像插值就是利用已知邻近像素点的灰度值（或 rgb 值）来产生未知像素点的灰度值（或 rgb 值），以便由原始图像再生出具有更高分辨率的图像。图像处理的学问太多，涉及到的数学也比较深，这里只简单记录应用。

常见的插值算法有最近邻插值算法，双线性插值算法，三次卷积等。具体内容可以去学习一些 DIP(Digital image processing) 的公开课。如：

http://inside.mines.edu/~whoff/courses/EENG510/lectures/

http://cs.nju.edu.cn/liyf/dip15/dip.htm

常用的是 MATLAB 内置的 `interp2` 函数，

Github 上找了个现成的工具箱[7]，快速实现 bicubic 和 bilinear 两种插值算法，效果很不错。

[7] https://github.com/FlorentBrunet/image-interpolation-matlab

第二章 Python

1 如何展开一个嵌套的序列?

我们现在有这样一个序列 `items = [1, 2, [3, 4, [5, 6, [9, 8], 7], 8]]`，我们想逐级展开这个序列，然后将所有元素装入一个序列。

如果这个序列层级较少，我们可以用多层 `for` 循环来遍历这个序列。一旦这个序列超过 3 层，过多的循环会让你很头疼。同样，这种多层级的问题我们可以用**递归**来解决。构建一个函数，这个函数能处理第一层的元素，由于第二层是 `list`，它是一个可迭代对象，我们只需要判断第二层是不是可迭代对象，同时忽略 `str, bytes` 对象[①]。只要内层是可迭代的，我们就开始递归，对其应用该函数。

```
from collections import Iterable

#def unfold(items, unfolded=None, ignore_types=(str, bytes)):
#    unfolded = list() if unfolded is None else unfolded
#
#    for item in items:
#        if isinstance(item, Iterable) and not isinstance(item, ignore_types):
#            unfold(item, unfolded=unfolded)
#        else:
#            unfolded.append(item)
#
#    return unfolded


# 更优雅的版本
def unfold(items, ignore_types=(str, bytes)):
    unfolded = []
    for item in items:
        if isinstance(item, Iterable) and not isinstance(item, ignore_types):
            unfolded.extend(unfold(item))
        else:
            unfolded.append(item)

    return unfolded
```

① `str, bytes` 也是可迭代对象，我们要避免其展开成单个字符。

由于存在递归，所以函数会被调用很多次，每次调用所得的数据都需要保留，如何在多次的调用之间共享保留数据呢？我采用一个默认参数来实现[②]，首次调用时不给默认参数新值，这会产生一个空的`list`，当对内层对象调用时，将上一次产生的数据赋值给这个参数。输出结果：

```
>>> items1 = ['Paula', ['Thomas', 'Lewis', ['siyu', 'ziyan', ['jianyuan']]]]
>>> items2 = [1, 2, [3, 4, [5, 6, [9, 8], 7], 8]]
>>> items3 = [[1, 2], 3, (4, [5, 6])]
>>> print(unfold(items1))
>>> print(unfold(items2))
>>> print(unfold(items3))
['Paula', 'Thomas', 'Lewis', 'siyu', 'ziyan', 'jianyuan']
[1, 2, 3, 4, 5, 6, 9, 8, 7, 8]
[1, 2, 3, 4, 5, 6]
```

但这样做有两个显而易见的坏处，一是当我们的嵌套序列有无限多层，递归会栈溢出；二是序列整个被读取到内存中了，当序列元素非常多，比如 1 亿，内存会被撑死。坏处一我们不去管他，大多数情况下是适用的，坏处二可以很容易的利用 generator 来解决[③]。

```
from collections import Iterable


def unfold(items, ignore_types=(str, bytes)):

    for item in items:
        if isinstance(item, Iterable) and not isinstance(item, ignore_types):
            yield from unfold(item)
        else:
            yield item
```

使用 generator 一来能防止内存爆炸，二来不需要在函数的多次调用见传递数据，代码更清晰明朗。需要注意，generator 是惰性序列，边调用边计算，我们需要使用`for`迭代出每一个元素或者直接用`list()`获取全部元素。

```
items1 = ['Paula', ['Thomas', 'Lewis', ['siyu', 'ziyan', ['jianyuan']]]]
items2 = [1, 2, [3, 4, [5, 6, [9, 8], 7], 8]]
items3 = [[1, 2], 3, (4, [5, 6])]
print(list(unfold(items1)))
print(list(unfold(items2)))
print(list(unfold(items3)))
```

[②] 前几天没有回想起 list 有个 extend 方法，显然用 extend 方法来实现更加优雅

[③] 思路来源 http://python3-cookbook.readthedocs.io/zh_CN/latest/c04/p14_flattening_nested_sequence.html

2 如何遍历当前文件夹及其子文件夹中的全部文件？

前面用 MATLAB 实现了一个，现在用 Python 来实现。第一种方法是利用递归来实现，思路同样是先找文件，然后找子文件夹，最后对子文件夹递归；第二种方法是利用 os.walk 模块，并将其做成 generator，这样在应对大量的文件时会有优势。推荐第二种方法，一来 os 模块考虑了很多我们忽略了的细节[④]，二来 generator 是一个优雅的设计，用 Python 就应该好好学用 generator。

```
from os import listdir, walk
from os.path import isfile, isdir, join


def get_all_file_name(path):

    file_list = [join(path, f) for f in listdir(path) if isfile(join(path,
     ↪ f))]
    sub_dir = [join(path, d) for d in listdir(path) if isdir(join(path, d))]

    for i in range(len(sub_dir)):
        next_dir = sub_dir[i]
        file_list.extend(get_all_file_name(next_dir))

    return file_list


def get_all_file_name_generator(path):

    for folder, subdirs, files in walk(path):
        for f in files:
            yield join(folder, f)
```

3 如何在遍历 list 时删除元素？

存在一个 list_a = [1, 2, 3, 4, 5, 6, 7, 8, 9]，现在需要逐一操作内部元素，并在操作结束之后删除它。使用 `while` 判断 `list` 是否为空，不为空则 `pop` 第一个元素，在循环下依次操作每一个元素。

```
list_a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

while list_a:
```

④ 比如，如果递归版本的函数遍历的根目录是一个磁盘，这个磁盘上的特殊的文件夹“System Volume Information”又是禁止被访问的，这时就会抛出一个 PermissionError。笨一点的解决办法是从子目录的 list 中删除这个目录，好一点的办法就是用 os 模块了。

```
    temp = list_a.pop(0)
    print(temp)
```

4 数值计算

第三章 C 和 C++

1 C 语言的动态数组

大多数时候为了方便（其实是我菜），会使用库较多（方便）的 C++，但是 C 语言在实际生产中使用率仍然很高，比如长期使用的 ANSYS Fluent 的 UDF 就不得不用 C 语言。下面是一个简易的动态数组的实现，来源[①]。

```c
#include <iostream>

typedef struct
{
    int *array;
    size_t used;
    size_t size;
} Array;

void initArray(Array *a, size_t initialSize)
{
    a->array = (int *)malloc(initialSize * sizeof(int));
    a->used = 0;
    a->size = initialSize;
}

void insertArray(Array *a, int element)
{
    // a->used is the number of used entries, because a->array[a->used++]
    ↪ updates a->used only *after* the array has been accessed.
    // Therefore a->used can go up to a->size
    if (a->used == a->size)
    {
        a->size *= 2;
        a->array = (int *)realloc(a->array, a->size * sizeof(int));
    }
    a->array[a->used++] = element;
}
```

① https://stackoverflow.com/questions/3536153/c-dynamically-growing-array

```c
void freeArray(Array *a)
{
    free(a->array);
    a->array = NULL;
    a->used = a->size = 0;
}

int main()
{
    Array a;
    int i;

    initArray(&a, 5); // initially 5 elements
    for (i = 0; i < 100; i++)
        insertArray(&a, i);     // automatically resizes as necessary
    printf("%d\n", a.array[9]); // print 10th element
    printf("%d\n", a.used);     // print number of elements
    freeArray(&a);
    return 0;
}
```

第四章 算法

算法可视化的网站，https://www.cs.usfca.edu/%7Egalles/visualization/Algorithms.html，晕乎乎的时候看几遍动画就明白了。

1 排序

先来了解几个基本概念：

排序：将一组“无序”的记录序列调整为“有序”的记录序列。

稳定性：假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，则称这种排序算法是稳定的，否则称为不稳定的。

排序算法的分类：插入类、交换类、选择类、归并类和基数类。

基于比较的排序算法的最佳性能为 $O(n \log n)$。

4.1.1 冒泡排序

复杂度：$O(n^2)$

1、对数组 array[n] 进行从 0 n-1 项的扫描，每碰到相邻两项数值大的在前小的在后时，对二者进行交换。当扫描进行完成后，0 n-1 中最大的元素必然已经在 array[n-1] 就位，而所有数值较小，序号却靠后的元素，序号也减小了 1。

2、既然最大的元素已在 array[n-1] 的位置就位，接下来的扫描只需从 0 n-2。具体过程同 1。同样的，扫描结束后 0 n-2 中最大的元素（全数组第二大的元素）必然已经在 array[n-2] 就位，而所有数值较小，序号却靠后的元素，序号也减小了 1。

3、如此不断重复，直到最小的元素在 array[0] 的位置就位。

从上述描述中我们可以看到“冒泡排序”这个名字的由来：每一次扫描，都可以使得数值较小，序号却靠后的元素的序号减少 1，宏观来看这些元素就像是从数组底部向上慢慢上浮的泡泡。

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

template <typename T>
void bubble_sort(T arr[], int size)
{
    int i, j;
    for (i = 0; i < size - 1; i++)
        for (j = 0; j < size - 1 - i; j++)
            if (arr[j] > arr[j + 1])
```

```
                swap(arr[j], arr[j + 1]);
}

int main()
{
    int arr[] = {61, 17, 29, 22, 34, 60, 72, 21, 50, 1, 62};
    int size = (int)sizeof(arr) / sizeof(*arr);
    bubble_sort(arr, size);
    for (int i = 0; i < size; i++)
        cout << arr[i] << ' ';
    cout << endl;

    float arrf[] = {17.5, 19.1, 0.6, 1.9, 10.5, 12.4, 3.8, 19.7, 1.5, 25.4,
    ↪  28.6, 4.4, 23.8, 5.4};
    size = (int)sizeof(arrf) / sizeof(*arrf);
    bubble_sort(arrf, size);
    for (int i = 0; i < size; i++)
        cout << arrf[i] << ' ';
    return 0;
}
```

4.1.2　插入排序

复杂度：$O(n^2)$，具体算法描述如下：

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中*从后向前扫描*
3. 如果该元素（已排序）大于新元素，将该元素移到下一位置
4. 重复步骤 3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到该位置后
6. 重复步骤 2 5

插入排序和人们打牌时所用的排序方式类似：抽第一张牌，此时手上的牌只有一张，所以是有序的。再抽一张牌，和手上的那张牌的大小进行比较，比它大就放在后面，否则放在前面。再抽一张牌，和手上的牌进行比较，插入在合适的位置，保持手上的牌有序。不断重复，直到牌抽完。从宏观来看，插入排序把数组分割成两部分，*前段有序后段无序*，随着插入排序的进行，后段无序的牌也越来越少，直到后段全部融入前段，排序也就结束了。

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

template <typename T>
void insert_sort(T arr[], int size)
{
```

```
    // 从第二个元素开始循环遍历未排序数组
    for (int i = 1; i < size; i++)
    {
        int temp = arr[i];
        int j = i - 1; //已排序数组最大索引
        // 从后向前扫描已排序数组
        while (j >= 0)
        {
            if (arr[j] > temp)
                arr[j + 1] = arr[j];
            else
                break;
            j--;
        }
        arr[j + 1] = temp;
    }
}

int main()
{
    int arr[] = {61, 17, 29, 22, 34, 60, 72, 21, 50, 1, 62};
    int size = (int)sizeof(arr) / sizeof(*arr);
    insert_sort(arr, size);
    for (int i = 0; i < size; i++)
        cout << arr[i] << ' ';
    cout << endl;

    float arrf[] = {17.5, 19.1, 0.6, 1.9, 10.5, 12.4, 3.8, 19.7, 1.5, 25.4,
    ↪  28.6, 4.4, 23.8, 5.4};
    size = (int)sizeof(arrf) / sizeof(*arrf);
    insert_sort(arrf, size);
    for (int i = 0; i < size; i++)
        cout << arrf[i] << ' ';
    return 0;
}
```

4.1.3　归并排序

复杂度：$O(n \log n)$

归并排序的操作有两步，分割和归并

1、分割：将数组二等分，并将得到的子数组继续二等分，直到每个子数组只剩下一个元素为止。

2、归并：不断将原本属于同一个数组的两个子数组归并成一个有序的数组，方法为不

断比较子数组的首元素，并弹出较小的放入合并后组成的数组中。直到所有子数组合并为一个数组。

4.1.4　选择排序

复杂度：$O(n^2)$

在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

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

template <typename T>
void select_sort(T arr[], int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        // 假设最小元素为未排序部分的第一个
        int min = i;
        // 遍历未排序部分
        for (int j = i + 1; j < size; j++)
        {
            // 找到当前循环内的最小值
            if (arr[j] < arr[min])
            {
                min = j;
            }
        }
        swap(arr[i], arr[min]);
    }
}

int main()
{
    int arr[] = {61, 17, 29, 22, 34, 60, 72, 21, 50, 1, 62};
    int size = (int)sizeof(arr) / sizeof(*arr);
    select_sort(arr, size);
    for (int i = 0; i < size; i++)
        cout << arr[i] << ' ';
    cout << endl;

    float arrf[] = {17.5, 19.1, 0.6, 1.9, 10.5, 12.4, 3.8, 19.7, 1.5, 25.4,
    ↪  28.6, 4.4, 23.8, 5.4};
```

```
    size = (int)sizeof(arrf) / sizeof(*arrf);
    select_sort(arrf, size);
    for (int i = 0; i < size; i++)
        cout << arrf[i] << ' ';
    return 0;
}
```

4.1.5　快速排序

复杂度：$O(n^2)$（最坏情况）；$O(n \log n)$（平均情况），光听名字就很快。它其采用了一种分治的策略，通常称其为分治法 (Divide-and-ConquerMethod)。分治法的基本思想是：将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

1. 从数列中取出一个数作为基准数（枢轴，pivot）；

2. 将数组进行划分 (partition)，将比基准数大的元素都移至枢轴右边，将小于等于基准数的元素都移至枢轴左边；

3. 递归地（recursively）把子数列排序。

C++ 实现的递归版本：

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

template <typename T>
int partition(T arr[], int low, int high)
{
    int pivot = arr[high]; // 挑最后一个元素做 pivot
    int storeIndex = low - 1;

    // 遍历数组
    for (int j = low; j < high; j++)
    {
        if (arr[j] <= pivot)
        {
            storeIndex++; // 移动储存点
            swap(arr[storeIndex], arr[j]);
        }
    }
    // 将 pivot 放到两个子数组中间
    swap(arr[storeIndex + 1], arr[high]);

    return (storeIndex + 1); // 返回索引
}
```

```cpp
template <typename T>
void quick_sort(T arr[], int low, int high)
{
    // 数组尺寸大于 1
    if (low < high)
    {
        int p = partition(arr, low, high);
        quick_sort(arr, low, p - 1);
        quick_sort(arr, p + 1, high);
    }
}

int main()
{
    int arr[] = {61, 17, 29, 22, 34, 60, 72, 21, 50, 1, 62, 99, 90};
    int size = (int)sizeof(arr) / sizeof(*arr);
    quick_sort(arr, 0, size - 1);
    for (int i = 0; i < size; i++)
        cout << arr[i] << ' ';
    cout << endl;

    float arrf[] = {17.5, 19.1, 0.6, 1.9, 10.5, 12.4, 3.8, 19.7, 1.5, 25.4,
    ↪  28.6, 4.4, 23.8, 5.4};
    size = (int)sizeof(arrf) / sizeof(*arrf);
    quick_sort(arrf, 0, size - 1);
    for (int i = 0; i < size; i++)
        cout << arrf[i] << ' ';
    return 0;
}
```

Python 实现的递归版本：

```python
def quicksort(items, p, r):
    if p < r:
        q = partition(items, p, r)
        quicksort(items, p, q - 1)
        quicksort(items, q + 1, r)


def partition(items, p, r):
    x = items[r]
    i = p - 1
```

```
    for j in range(p, r):
        if items[j] <= x:
            i = i + 1
            items[i], items[j] = items[j], items[i]

    items[i + 1], items[r] = items[r], items[i + 1]
    return i + 1


items = [2, 5, 9, 3, 7, 0, -1]
quicksort(items, 0, len(items) - 1)
print(items)
```

问题：为什么临时储存点的索引是 low-1？

第五章 Git

1 如何给 Git 仓库添加一个空文件夹?

默认情况下，空文件夹不被记录，也不能被推送。特殊需求参见How can I add an empty directory to a Git repository? - Stack Overflow