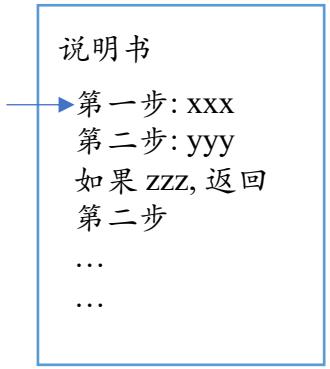
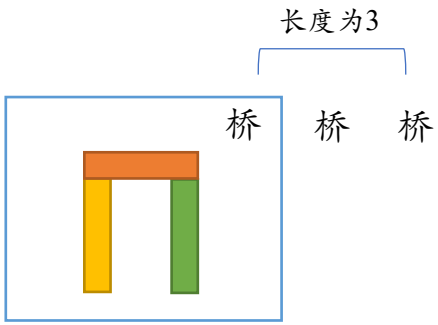
OI 入门组冲刺

1. 计算机程序的构造与解释(简介)

你将会遇到...



计算机程序的执行



如何“搭积木”

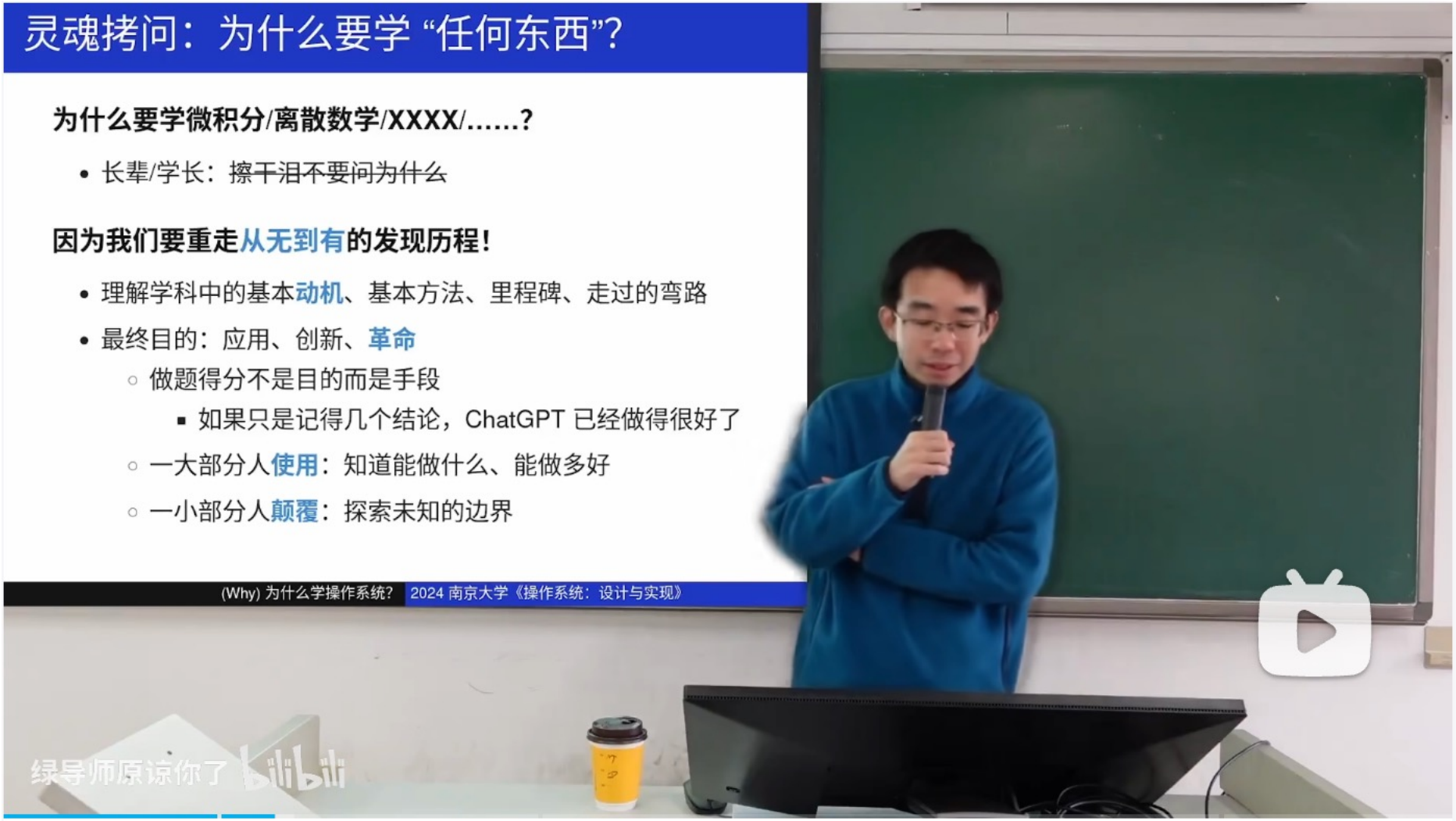
在区间1..10解决A问题

在1..5解决A问题　　在6..10解决A问题

递归精灵

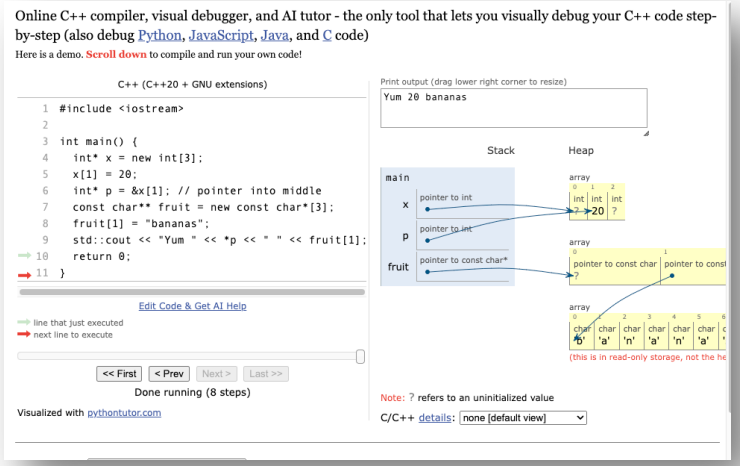
Recursion Fairy

学习 “任何东西”


灵魂拷问：为什么要学 “任何东西”?
为什么要学微积分/离散数学/XXXX/……?
长辈/学长：擦干泪不要问为什么
因为我们要重走从无到有的发现历程!
理解学科中的基本动机、基本方法、里程碑、走过的弯路
最终目的：应用、创新、革命
做题得分不是目的而是手段
如果只是记得几个结论，ChatGPT 已经做得很好了
一大部分人使用：知道能做什么、能做多好
一小部分人颠覆：探索未知的边界
(Why) 为什么学操作系统?
2024 南京大学《操作系统：设计与实现》
绿导师原谅你了
bilibili

竞赛的语言: C with classes

- 让我们的进度处在同一页…
- 变量：带有类型的小盒子
  - 数组：给一组带有类型的小盒子
  - 额外的警告：小心整数溢出！
- 控制流语句：`if`，`while`
  - `for`只是`while`的一个“方言”
- 函数调用
  - 参数传递是复制
  - 调用中的调用：使用栈传递参数
- 创建新类型：`struct`
- 最后的方法：使用 `pythontutor.com`



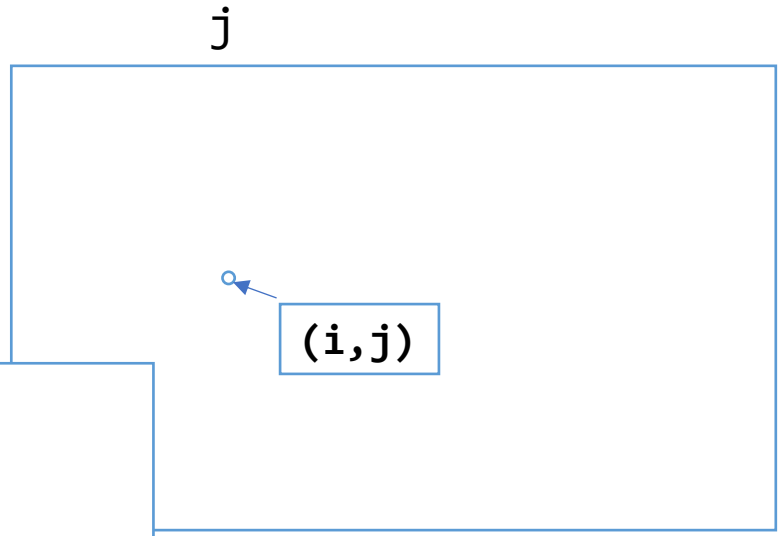
练习1:图形绘制

练习2:for转while

练习3:复印机/共享文档?

练习: 屏幕上画图

- 屏幕可以看做二维数组
- 现在有一个 25×50 的屏幕
- 某个函数返回1 = 发光; 返回0=不发光




```
bool f(int i, int j){
  double x = ((j * 2) / 15.0 - 3) / 2;
  double y = (i * 2) / 15.0 - 2;
  return pow(pow(x, 2) + pow(y, 2) - 1, 3) + \
             pow(x, 2) * pow(y, 3) < 0;
}
```

- 所有发光的点会构成什么图形?

练习: 把for转为while

```
for( init ; judge ; alter ){
    body
}
```

如果用 while 怎么写?

练习: 复印机还是共享文档?

```
void a(int x, int y){
    x = x+1;
}

int main(){
    int x = 1, y = 2;
    int z = 1, w = 2;
    a(x, y); a(z, w);
    // 这时候x,y,z,w是多少?
    return 0;
}
```

```
int x = 1, y = 2;
void a(int x, int y){
    x = x+1;
}

int main(){
    int z = 1, w = 2;
    a(x, y); a(z, w);
    // 这时候x,y,z,w是多少?
    return 0;
}
```

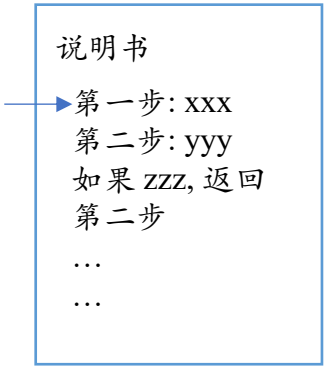
```
int x = 1, y = 2;
void a(int q, int w){
    x = x+1;
}

int main(){
    int z = 1, w = 2;
    a(x, y); a(z, w);
    // 这时候x,y,z,w是
       多少?
    return 0;
}
```

问题: 如何模拟执行计算机程序?

- 类比: 读说明书

- 任何东西不懂的时候 “最后” 的办法
  - 选一个足够小, 足够全面的例子
  - 做可视化/输出中间变量等

- 数学证明也是一个很好的辅助(但是有时候看不懂)
  - 支线任务: 让大家具备能看懂数学证明的前置知识



更高的视角看程序

过程

- 操作数据的规则

我们写的C++代码

(实际程序当然也会加载到内存里面, 但是方便起见现在可以把它们分开来看)

数据

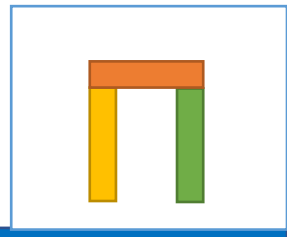
- 希望操作的东西

内存里面的某些东西

- 构造抽象　　函数(调用)　　结构体

```
int 吃饭(){
A();B();C();
}
```

时间
- 小时(int)
- 分钟(int)
- 秒(int)



马上: 构造过程抽象

- 在练习中体会与学习
- 若干个例子
  - 致敬传奇经典《计算机程序的构造与解释》
  - 一些真实的习题
  - 还有一些Bonus题目

BV19U4y187U5

🌶️[SICP1986] “唯一编程神课”

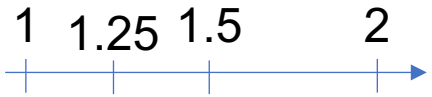
(但大概率需要多年的编程经验才能看懂)

求平方根: 思想

$$\sqrt{x} := y, \text{其中} y \text{满足} y^2 = x.$$

- 数学定义: 是什么?
- 计算机程序: 怎么做?

当然是先猜啦! (以$\sqrt{2}$为例)



猜测y_i	猜测的平方y_i^2	大了还是小了?
1	1	小了
1.5	2.25	大了
1.25	1.5625	小了
1.375	.....	

希望调大一点

- 直到我们猜测了一个 “足够好” 的值($\left|y_i^2 - x\right| < \epsilon$)

求平方根: 过程抽象

$\sqrt{x} := y$, 其中y满足$y^2 = x$.

- 数学定义: 是什么?
- 计算机程

输入：区间$[L, R]$，猜测$\frac{(L+R)}{2}$. → 返回一个新的区间

猜测y_i	猜测的平方y_i^2	大了还是小了?
1	1	小了
1.5	2.25	大了
1.25	1.5625	小了
1.375	.....	

希望调大一点

- 直到我们猜测了一个 “足够好” 的值($\left|y_i^2 - x\right| < \epsilon$)

求平方根: 写一写代码

DISCLAIMER: 仅供演示 wishful thinking, 平时写代码不用这么上纲上线!

一些函数

- 阶乘函数: $\mathrm{fac}(n) \coloneqq \begin{cases} 1 & n = 1 \\ \mathrm{fac}(n-1) & n > 1 \end{cases}$
- Fibonacci函数: $\mathrm{fibo}(n) \coloneqq \begin{cases} 1 & ,n = 1,2 \\ \mathrm{fibo}(n-1) + \mathrm{fibo}(n-2) & ,n > 2 \end{cases}$

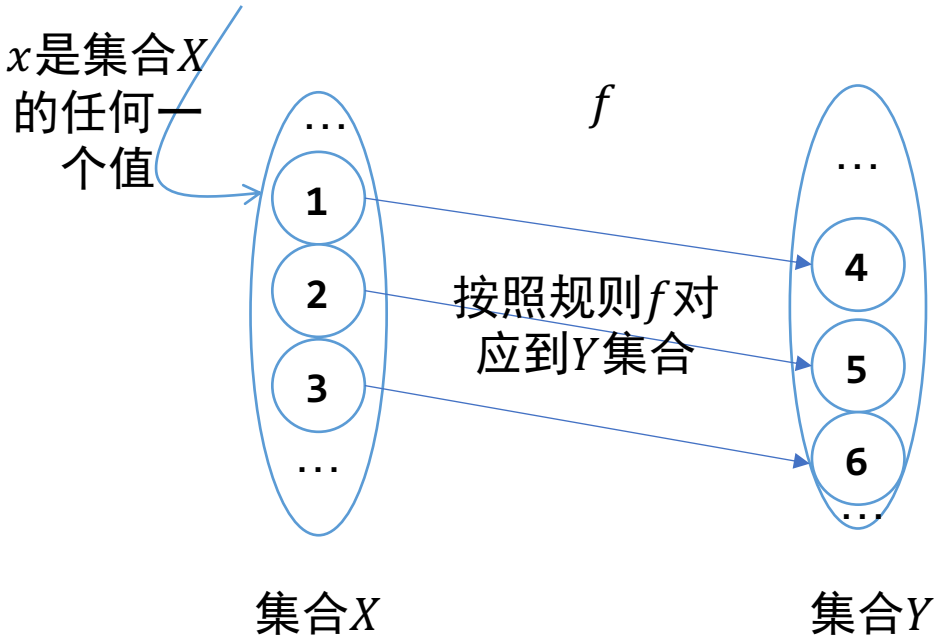
- $\mathrm{fac}(3), \mathrm{fibo}(5)$
  - 用纸和笔计算
  - 写成计算机函数, 执行过程

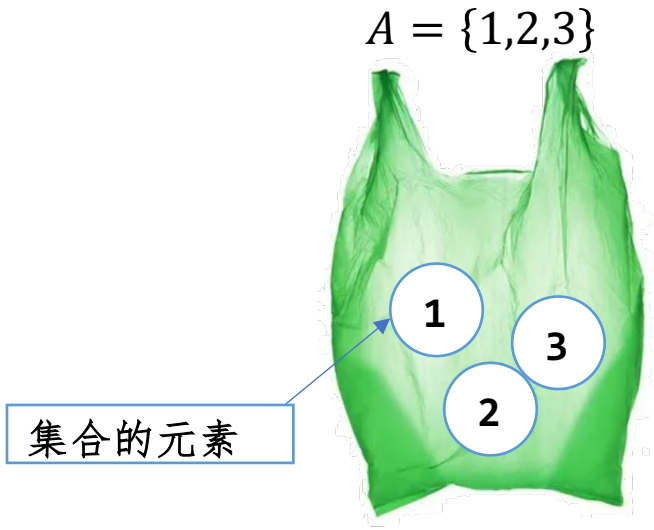
- 思考: 计算机的函数与数学中的函数有什么区别?

Aside: 函数(映射)

- 映射: 把一堆东西对应到另一堆东西

$f(x) = x + 3$的意思是…





我们把集合理解为由
- 若干确定的
- 有充分区别的(不重复)
- 具体或抽象的对象

合并而成的一个整体

---- George Cantor

- 为什么叫函数(包含变量的数, 但是更有趣的野史是)
  - 送信: 左边送信, 右边收信
  - 同一封信不能同时送给两个人

图片来源: https://cnycentral.com/news/local/governor-cuomo-introduces-legislation-to-ban-single-use-plastic-bags-in-new-york

换零钱的方法数

- 问题描述:
  - 无限个0.5元, 0.25元, 0.10元, 0.05元, 0.01元的硬币
  - 凑出1元的方法数?

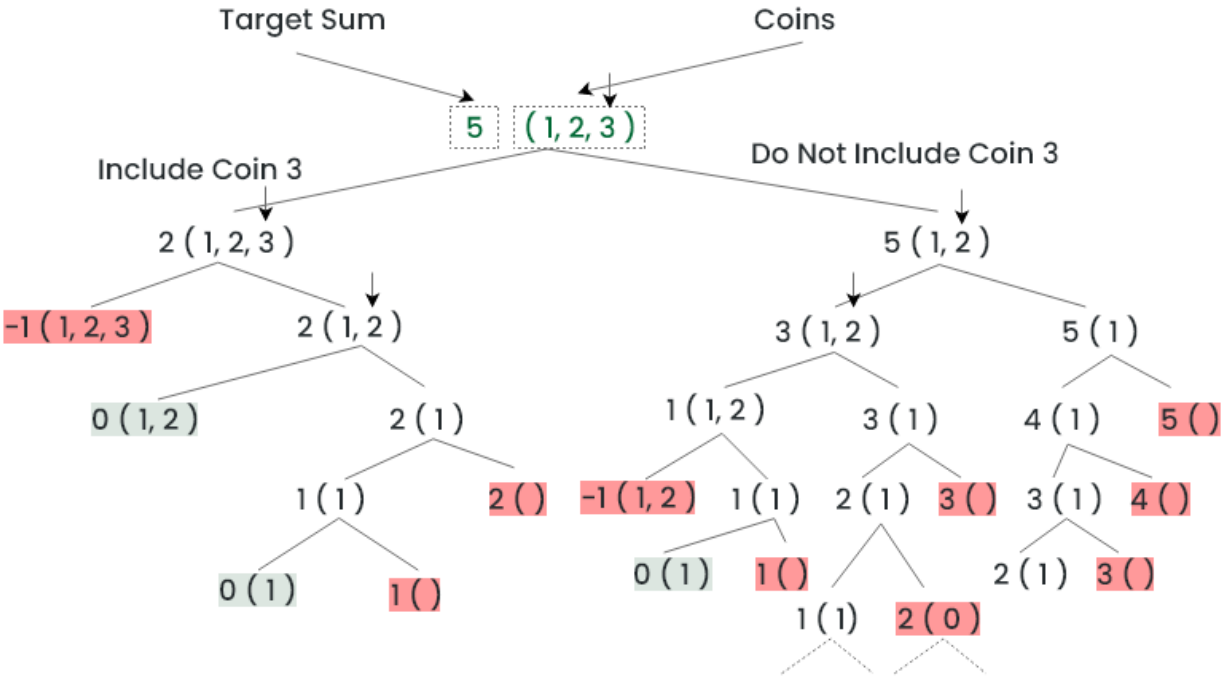
- 分步相加原理: 两个不相交的集合, 他们的元素个数等于第一个的个数+第二个的个数
- #xxx 的意思是 xxx 的方法数

将总数为a元的现金换为n种硬币的不同方式
- # 将a元换为除了第一种硬币之外的其他硬币换零钱的不同方式
- # 将$a-d$元换为包含第一种硬币换零钱的的不同方式

d是第一种硬币的面值

- 问题: 为什么只要减去d, 而不用考虑减去$2d, 3d, \ldots$的情况?

换零钱的方法数(例子)



- 答案: 因为我选了第一种硬币之后还可以再选第一种, 等效于选了2个第一种

图片来源: https://www.geeksforgeeks.org/coin-change-dp-7/

换零钱的方法数: 总结

- 问题描述:
  - 0.5元, 0.25元, 0.10元, 0.05元, 0.01元的硬币
  - 凑出1元的方法数?
- 边界情况

将总数为a元的现金换为n种硬币的不同方式
- # 将a元换为除了第一种硬币之外的其他硬币换零钱的不同方式
- # 将$a-d$元换为包含第一种硬币换零钱的的不同方式

d是第一种硬币的面值

- $a=0$, 算作有1种换零钱的方式
- $a<0$, 算作有0种换零钱的方式(超了)
- $n=0$, 有0种换零钱的方案(没有可选了)

关键是如何看待问题

- 暴力枚举: num(当前的和)

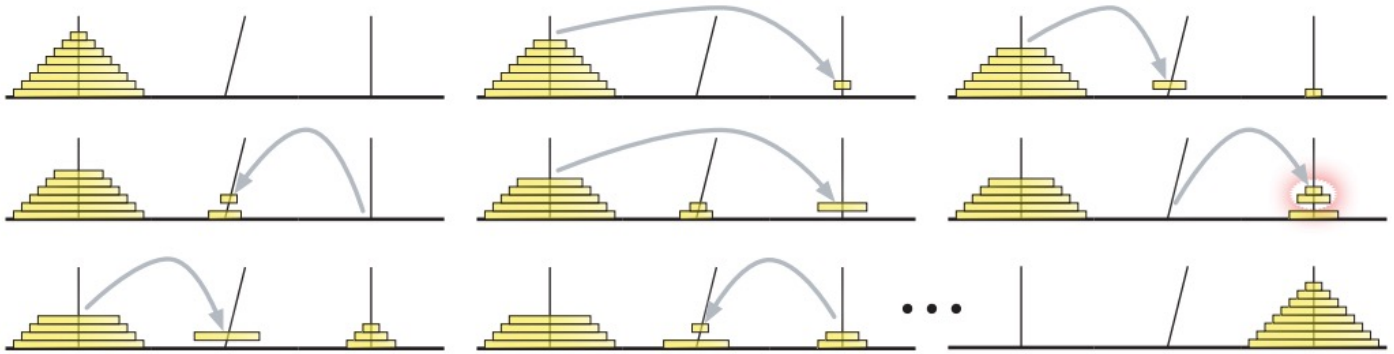
- 刚刚的看法: 相同的 “状态” 归类
- 接下来(的一生)会遇见各种各样的看问题的手段

最大公约数(简介)

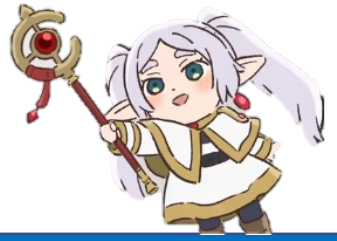
- 最大公约数: $\gcd(a,b) :=$ 能除尽a和b的那个最大的那个整数
- 观察: $\gcd(a,b) = \gcd(b, a\%b)$
  - 证明留到数学部分

Hanoi斜塔

- Hanoi塔, 但是斜着的柱子可以一步移过来



- 如何设计递归算法?

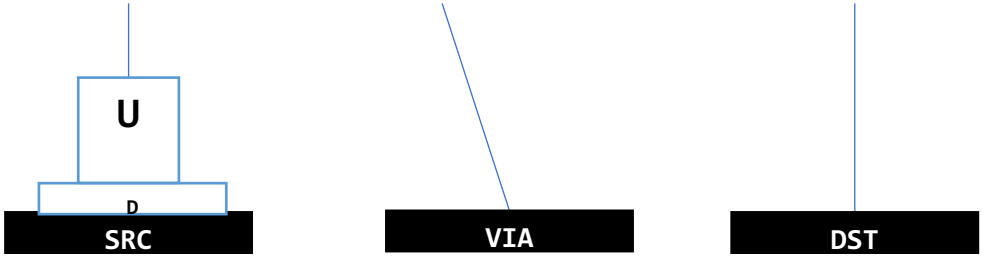


Hanoi斜塔(递归程序)

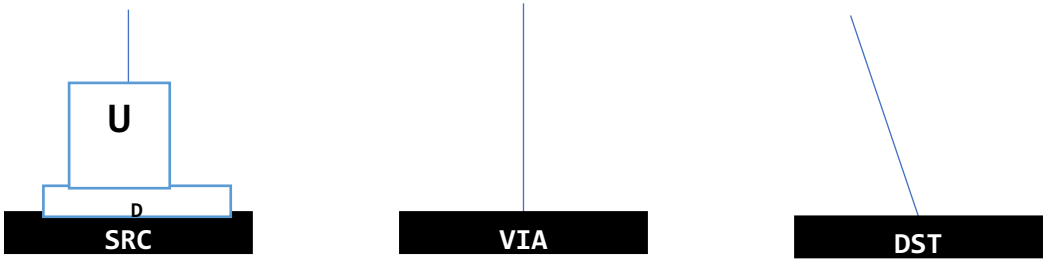
```c
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod){
    if (n == 1){
        printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
        return;
    }

    if(from_rod == 'B'){
        printf("Move all Bs to %c\n", to_rod);
        return ;
    }

    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);

    if(aux_rod != 'B'){
        towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
    }else{
        printf("Moving All Bs  -> %c\n", to_rod);
    }
}
```

Hanoi斜塔(解答)

- $T_{|\to|}(n)$:= 在直着的柱子之间转移n个盘子最小移动次数
- $T_{|\to\backslash}(n)$:= 在斜着的和直着的柱子之间转移n个盘子最小移动次数



$$T_{|\to|}(n) = T_{|\to\backslash}(n-1) + 1 + 1 \qquad T_{|\to|}(1) = 1$$

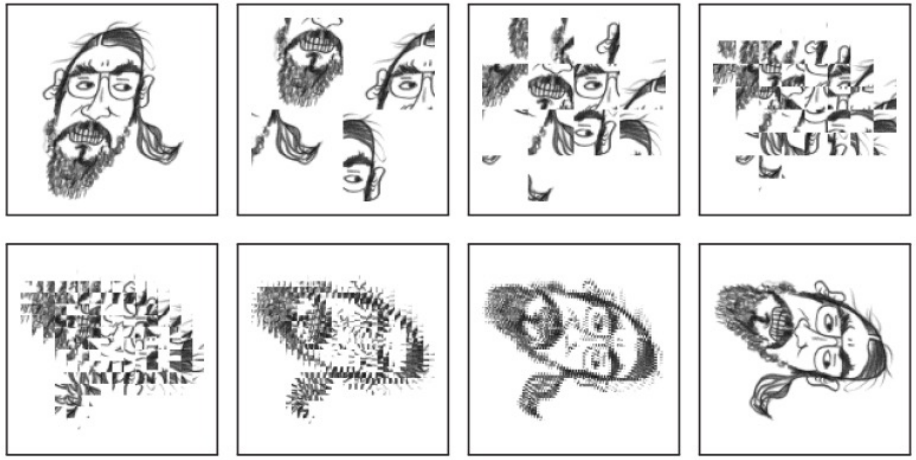


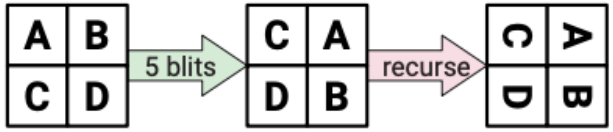
$$T_{|\to\backslash}(n) = T_{|\to|}(n-1) + 1 + T_{|\to\backslash}(n-1) \qquad T_{|\to\backslash}(1) = 1$$

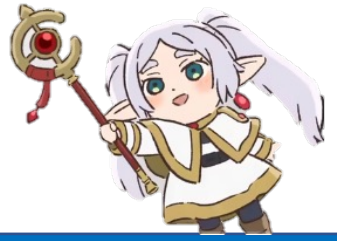
旋转图像

- 处理器的能力: 把一块图形(通过中间变量)复制

假设GPU上~$O(k)$的时间完成$k \times k$的矩形复制







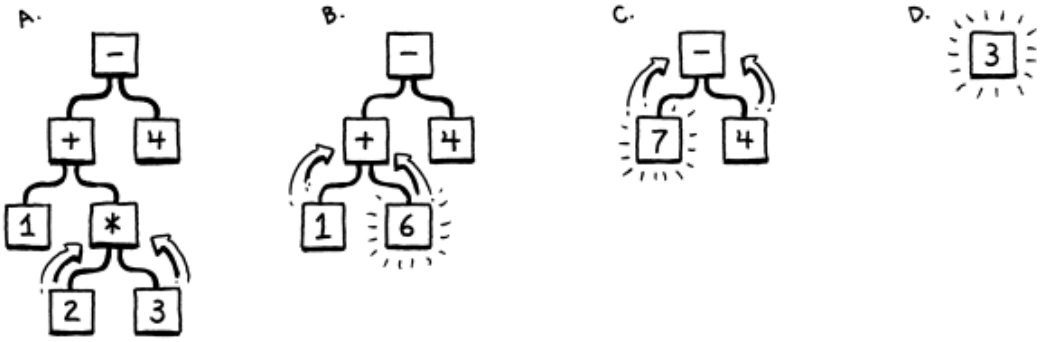
表达式求值

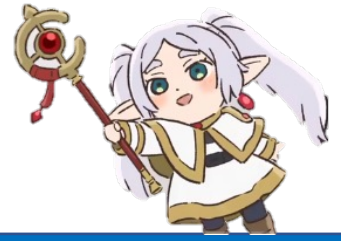
- 阅读表达式

$$1 + (2 \times 3) - 4$$

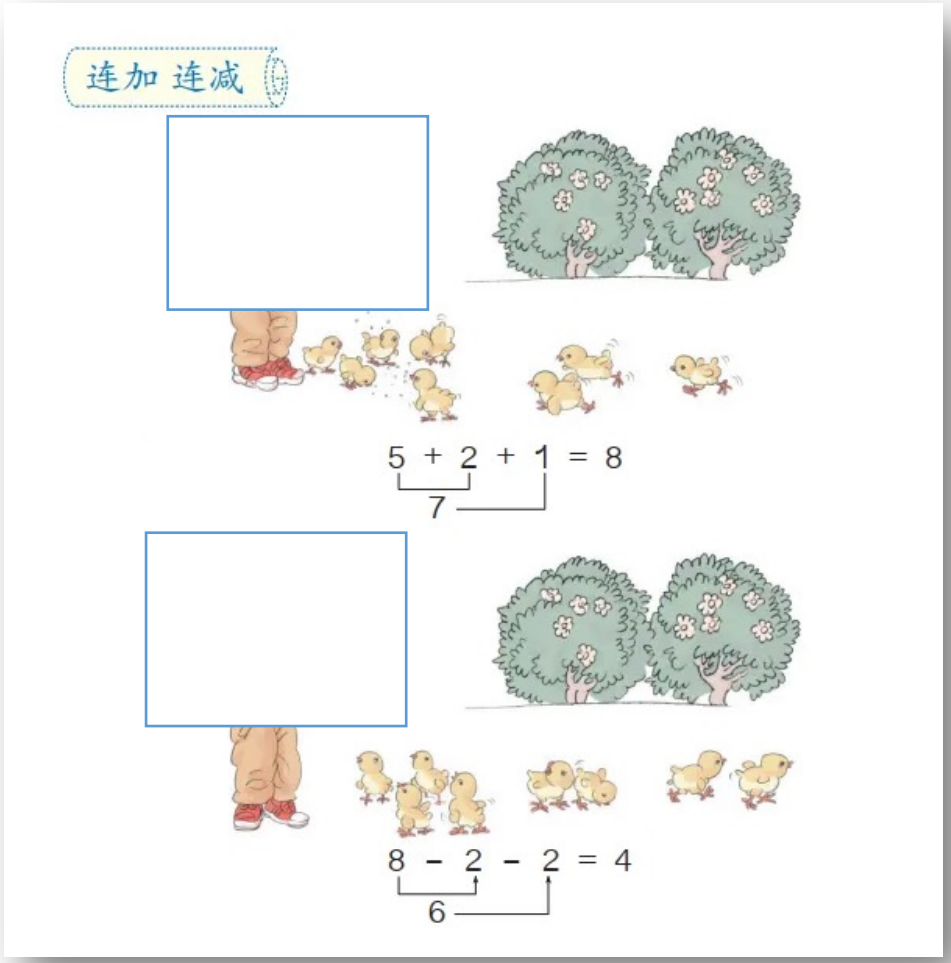
- 什么是一个合法的表达式? (结构上)

```
<expr> ::= <number>
  | "(" <expr> ")"
  | <expr> "+" <expr>
  | <expr> "-" <expr>
  | <expr> "*" <expr>
  | <expr> "/" <expr>
```

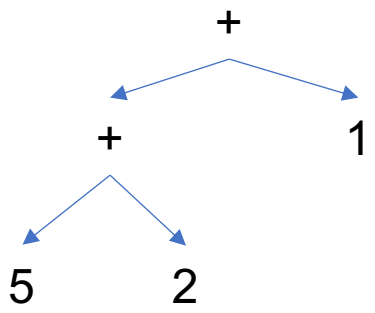




Aside: 小学数学课本



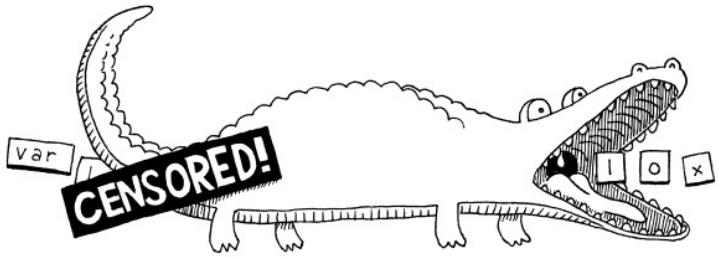
- 这不就是构建表达式树吗?



- 改版的时候人画得太丑了
- 打码处理

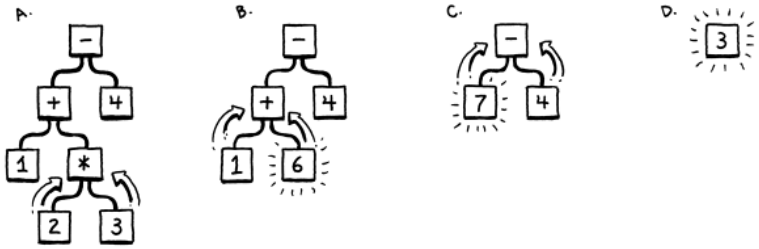
Extra: 编译器怎么工作的?

词法分析



其中一个过程(语法分析)

- 构建表达式树
- 在标记了节点的树上面DFS



BV1sJ4m1e7bM

[南大软院2024] 《编译原理》

(但大概率需要至少了解离散数学的基本概念和OOP才能学)

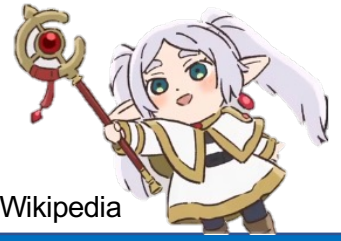
BV14L411F7xR

《离散数学》 (还是魏恒峰)

画三角形(P1498)

- 一个分形图案(**Sierpiński triangle**)

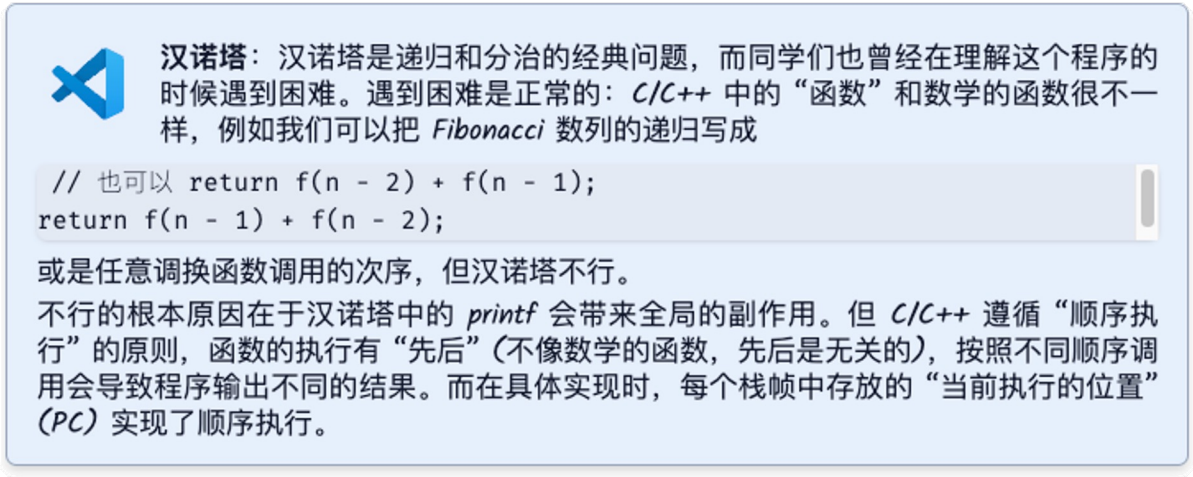
- **Sierpiński**三角形与Hanoi塔的联系(3Blue1Brown)



图片来源: Wikipedia

回答: 数学和计算机中的函数

- 数学中的函数 – 结果仅由输入决定
- 计算机中的函数 – 还有邪恶的全局变量 ← “不纯粹”
  - 给程序的执行带来 “副作用”



汉诺塔：汉诺塔是递归和分治的经典问题，而同学们也曾经在理解这个程序的时候遇到困难。遇到困难是正常的：*C/C++* 中的“函数”和数学的函数很不一样，例如我们可以把 *Fibonacci* 数列的递归写成

```
// 也可以 return f(n - 2) + f(n - 1);
return f(n - 1) + f(n - 2);
```

或是任意调换函数调用的次序，但汉诺塔不行。

不行的根本原因在于汉诺塔中的 *printf* 会带来全局的副作用。但 *C/C++* 遵循“顺序执行”的原则，函数的执行有“先后”（不像数学的函数，先后是无关的），按照不同顺序调用会导致程序输出不同的结果。而在具体实现时，每个栈帧中存放的“当前执行的位置”（*PC*）实现了顺序执行。

----蒋炎岩,南京大学《操作系统》2024

- 递归程序如何执行?

又出现 Segmentation Fault 了?

- 回顾: 程序执行 = 想象成读 “说明书” 操纵 “实在的东西”

- 错误=某一个预期的状态和预想的不一样

牢记:
- (1)机器永远是对的
  - 有0.00001%的概率是编译器错了(但是你能知道)
  - 有0.0000001%的概率是处理器错了(你也可以知道)
- (2) 未经测试的代码永远是错的

(大)程序难调试!
- Bug的触发经过了漫长的过程

需求 → 设计 → 代码 (Fault/bug) → 执行 (Error) → 失败 (Failure)

“调着调着一下午就过去了”

- zgw和他的同学实际生活中的例子


```
81  void init_tsklst(TSKLST *tsklst){
82    tsklst->nr_node = 0;
83    tsklst->dummy.prv = tsklst->dummy.nxt = &(tsklst->dummy);
84    // that is: tsklst->dummy.name = "idle";
85    memcpy(tsklst->dummy.name, "idle", sizeof("idle"));
86    tsklst->dummy.fence1 = STK_FENCE_MAGIC;
87    tsklst->dummy.fence2 = STK_FENCE_MAGIC;
88    // Make the dummy node running idle process
89    tsklst->dummy.context = kcontext((Area){.start = tsklst->du
      stk, .end=tsklst->dummy.stk+STK_SZ}, idle_, NULL);
90  }
```

```
159   if(CPUNOW.last != NULL && CPUNOW.last != CPUNOW.curr_task){
160     // shall unlock
161     if(strcmp(CPUNOW.idle->name, "idle percpu") != 0){
162       unlock_spinlk(&CPUNOW.last->hold);
163     }
164   }
165   CPUNOW.last = CPUNOW.curr_task;
166   panic_on(CPUNOW.curr_task->fence1 != STK_FENCE_MAGIC, "Stack
      corrupted");
167   panic_on(CPUNOW.curr_task->fence2 != STK_FENCE_MAGIC, "Stack
      corrupted~");
```

调试理论

如果我们能判定任意程序状态的正确性，那么给定一个 failure，我们可以通过二分查找定位到第一个 error 的状态，此时的代码就是 fault (bug)。

```
for(int i=1; i<=3; i++)
    for(int j=1; j<=3; i++)
        a[i][j]=1;
```

预期	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	…
实际	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	

- 判定程序的正确性不是简单的事情

方法1: 单步调试

- VSCode用户狂喜
- 当年的我: Dev-C++的调试器坏掉了
- 允许调试:
  - 编译选项加上: -g -O0
- 实在不行可以用gdb
  - start
  - b = break
  - c = continue
  - s = step
  - w = watchpoint
- 精细地控制, 但是很浪费时间

方法2: 观察执行的一个侧面

- 使用printf

小技巧: 宏定义

- `#define DEBUG(...) do{printf(“[DEBUG]”); printf(__VA_ARGS__);printf(“\n”);}while(0)`
  - 宏定义是文本级别的替换
  - 为什么用do while(0)包裹起来?
    - 直接输入语句会有悬挂的if语句的问题

```
#define DEBUG(...) printf(“[DEBUG]”); printf(__VA_ARGS__);printf(“\n”);

if(cond) DEBUG(“If Condition in!”);
else DEBUG(“Else in!”);
```

展开

```
if(cond) printf(“[DEBUG]”); printf(“If Condition in!”);printf(“\n”);
else printf(“[DEBUG]”); printf(“Else Condition in!”);printf(“\n”);
```

- 无法通过编译

遇到任何问题的检查单

着急调试程序检查单

1.是怎样的程序 (状态机) 在运行?
2.我们遇到了怎样的 failure ?
3.我们能从状态机的运行中从易到难得到什么信息?
4.如何二分检查这些信息和 error 之间的关联?

- 防御性编程: 程序不对的时候, 尽早退出
  - 把不变量写在断言里面
  - 双向循环链表: (assert(node->prv->nxt==node);)
- 你将用余下的一生(如果学习CS的话)践行这份检查单

```
void __attribute__((unused)) inspect_tnode(TSKLST
*bd){
  task_t *start = &bd->dummy;
  // Log("%p\n", start);
  int count = 0;
  start = start->nxt;
  while(start != &bd->dummy){
    panic_on(start->prv->nxt != start, "Did not
    maintain the llist well.");
    count++;
    start = start->nxt;
  }
  printf("NIL\n");
  panic_on(count != bd->nr_node, "Did not maintain
  size well.");
}
```

任何学习资料

~~计算机语言~~到底讲了啥?

“ 心智的活动，除了尽力产生各种简单的知识点之外，主要表现在如下三个方面:

1) 将若干简单知识点组合为一个复合知识点，由此产生出各种复杂的知识点。

2) 将两个知识点放在一起对照，不管它们如何简单或者复杂，在这样做时并不把它们合起来。由此得到有关它们的相互关系的知识点。

3) 将有关知识点与和当前知识点不重要的东西隔离开，这就是抽象，抽象可以做一大堆炫酷的事情

---- John Locke, The Essay concerning Human understanding

有关人类理解的随笔

(少儿版)

遇到困难

> … 教科書通りはよく言ったもので、難しい言葉だらけ。…
> 老师常言教科书上的才是正确答案,可上面皆是难懂的话语.

----《空の箱》(TV动画 Girls Band Cry 插曲)

- “翻译”多了自然习惯话术
- 看一看背后哪里有所疏漏?
  - 寻求老师/同学帮助
- 学习是螺旋上升的
  - 实在感觉不行先放一会儿

> 可以理解,“放弃”是面对困难时的本能行为... 教育的根本目不仅是通过外力“逼迫”大家成长,更重要的是教大家如何在面对困难时使用正确的思路解决.

----蒋炎岩,南京大学《操作系统》主讲

Cheers!

信息量最大的部分已经过去了
接下来是一些日常

高精度加减乘

- 问题: 在做乘法的时候, 如果乘的数非常大, 难道每一位的进位不会溢出吗?
  - 考虑2147483647/8=268435455 (1e9)
- 加法, 减法, 乘法
- 分析高精度加减乘的时间复杂度

解线性方程组: 为什么重要(1)

- 生活中到处都有线性方程组
  - 化学式配平
- 方程组 = 约束条件; 在约束条件下求解

$$x_1\mathrm{Cu} + x_2\mathrm{HNO_3}(\text{稀}) \triangleq x_3\mathrm{Cu(NO_3)_2} + x_4\mathrm{NO} + x_5\mathrm{H_2O}$$

- Cu守恒: $x_1 = x_3$
- H守恒: $x_2 = 2\times x_5$
- O守恒: $3\times x_2 = x_3\times 3\times 2 + x_4 + x_5$
- N守恒: $x_2 = 2\times x_3 + x_4$

- 5个未知数, 4个方程 → 无穷多组解
- 一组解 $3\mathrm{Cu} + 8\mathrm{HNO_3}(\text{稀}) \triangleq 3\mathrm{Cu(NO_3)_2} + 2\mathrm{NO} + 4\mathrm{H_2O}$

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \cdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n \end{cases}$$

解线性方程组: 问题转换

- 真正重要的事情

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \cdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n \end{cases}$$

- 实际上线性方程组就是...

$$\left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & b_2 \\ \cdots & & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} & b_n \end{array}\right]$$

解线性方程组: 怎么做?

- 三个基本变换

设$\begin{cases} x-y+2z=5 \\ x+y+z=6 \\ 2x+3y+5z=23 \end{cases}$ 的解是(x,y,z) ⇨ … ⇨ (我们希望) $\begin{cases} 1x+0y+0z=? \\ 0x+1y+0z=? \\ 0x+0y+1z=? \end{cases}$

- 任意交换两行, 解不变 —— $\begin{cases} x+y+z=6 \\ x-y+2z=5 \\ 2x+3y+5z=23 \end{cases}$ 的解也是(x,y,z)

- 某一行乘上一个非0倍数, 解不变 $\begin{cases} x-y+2z=5 \\ 2x+2y+2z=2\times6 \text{的解也是}(x,y,z) \\ 2x+3y+5z=23 \end{cases}$

- 某一行乘上一个倍数加到另一行去, 解不变

$2x+2y+2z=2\times6$ ⇦ 乘2

$2x+3y+5z=23$ ⇨ 相加

$\begin{cases} x-y+2z=5 \\ x+y+z=6 \\ 4x+5y+7z=35 \end{cases}$ 的解也是(x,y,z)

解线性方程组: 矩阵形式(1/2)

化为 “上三角” 形式

- 对于 $\begin{bmatrix}1 & -1 & 2 & 5\\ 1 & 1 & 1 & 6\\ 2 & 3 & 5 & 23\end{bmatrix}$ 第二行+=第一行×(−1) $\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 2 & 3 & 5 & 23\end{bmatrix}$ 第三行+=第一行×(−2) $\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 0 & 5 & 1 & 13\end{bmatrix}$

把这些置为0

0的好处: 后续做线性组合的时候不会产生进一步影响

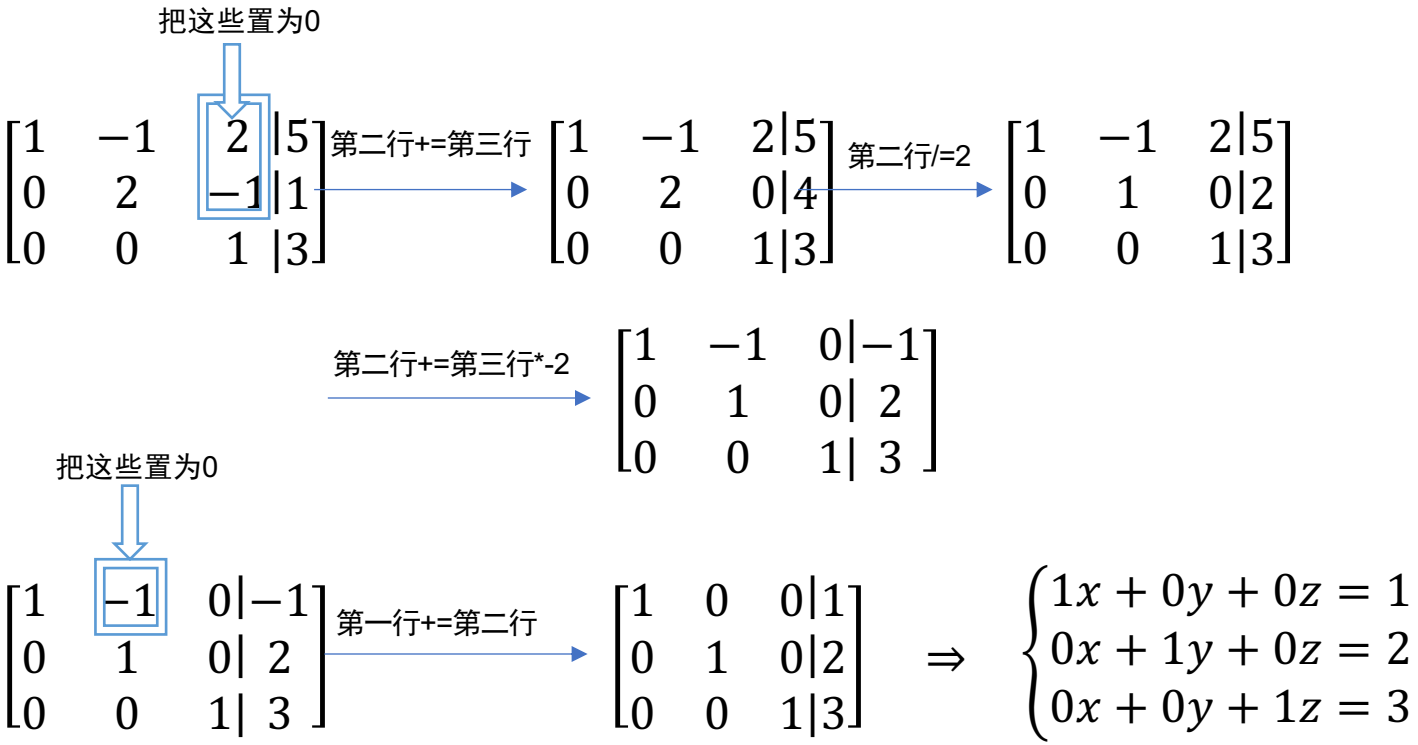
- 然后 $\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 0 & 5 & 1 & 13\end{bmatrix}$ 第三行+=第二行×$\left(-\frac{5}{2}\right)$ $\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 0 & 0 & 7/2 & 21/2\end{bmatrix}$ 第三行*=$\frac{2}{7}$ $\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 0 & 0 & 1 & 3\end{bmatrix}$

把这些置为0

这表明 $\frac{7}{2}z = \frac{21}{2} \Rightarrow z = 3$

解线性方程组: 矩阵形式(2/2)

化为 “对角” 形式



$$\begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & -1 & 1\\ 0 & 0 & 1 & 3\end{bmatrix} \rightarrow \begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 2 & 0 & 4\\ 0 & 0 & 1 & 3\end{bmatrix} \rightarrow \begin{bmatrix}1 & -1 & 2 & 5\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 3\end{bmatrix}$$

$$\rightarrow \begin{bmatrix}1 & -1 & 0 & -1\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 3\end{bmatrix}$$

$$\begin{bmatrix}1 & -1 & 0 & -1\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 3\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 3\end{bmatrix} \Rightarrow \begin{cases}1x + 0y + 0z = 1\\ 0x + 1y + 0z = 2\\ 0x + 0y + 1z = 3\end{cases}$$

线性方程组: 实现这个过程

- 好奇: 矩阵乘法是变量代换

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = c_1, \\ a_{21}x_1 + a_{22}x_2 = c_2, \end{cases} \qquad \begin{cases} b_{11}y_1 + b_{12}y_2 = x_1, \\ b_{21}y_1 + b_{22}y_2 = x_2. \end{cases}$$

将第二个方程带入第一个方程中

$$\begin{cases} (a_{11}b_{11} + a_{12}b_{21})y_1 + (a_{11}b_{12} + a_{12}b_{22})y_2 = c_1, \\ (a_{21}b_{11} + a_{22}b_{21})y_1 + (a_{21}b_{12} + a_{22}b_{22})y_2 = c_2, \end{cases}$$

- 就得到了

$$\begin{array}{cccc} & & b_{11} & b_{12} \\ & & b_{21} & b_{22} \\ & & \downarrow & \downarrow \\ a_{11} & a_{12} \quad \to & a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21} & a_{22} \quad \to & a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{array}$$

- B2105 矩阵乘法

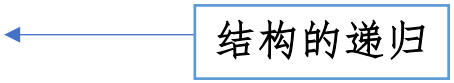
线性方程组: 无解和无穷解的情况

- 出现 $0x = 1$ ⇒无解
- 出现 $0x = 0$ ⇒无数解

- 名词: 矩阵, 矩阵的秩

马上: 结构和它的递归

```
struct Node{
    int val;
    Node *nxt;
}
```



- 方法1: 使用一个 “池子”, 下一个是它的编号

  nxt[m]:=编号为m的节点下一个编号为nxt[m]

- 方法2: 指针(Later)

递归的数据结构 (引言)

- 马上: 链表简介; 如何表示递归的结构
- 考虑场景:

“请你到计算机楼809找我”

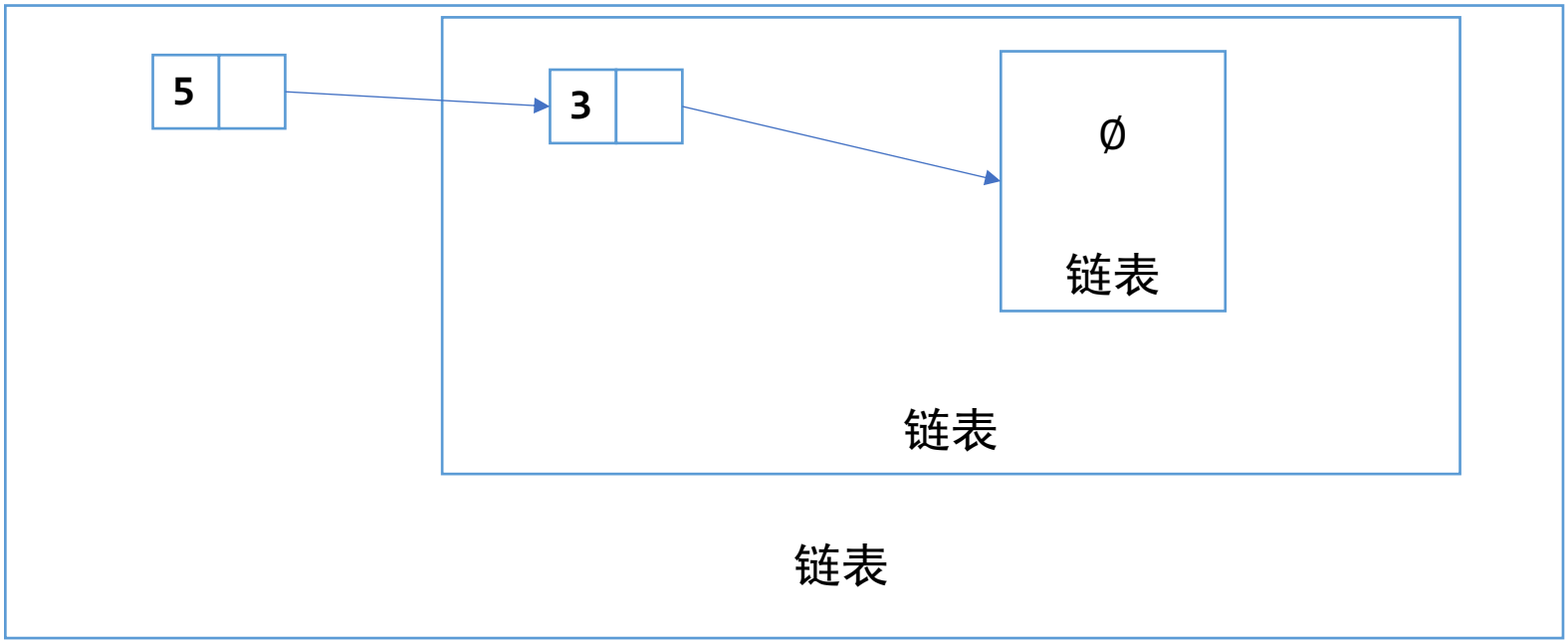
“和系主任有点事, 你来816吧”

(找到目标)

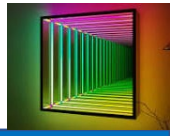
链表定义

- 链表的定义(一般): 用链把数据串起来
- 链表的定义(神金):
  - ø是链表
  - 一个节点, 下一个元素是链表的东西也是链表

马上会看到为啥要神金版本的定义.



- 关键是怎么表示 “下一个元素还是这一类的”?



链表表示(数组版本)

- 把它们像办公室一样组织起来, 通过门牌号表示
- 一个节点u, 它的下一个节点的编号是$\mathrm{nxt}[u]$.

#	1	2	3	4	5	6
val(数据值)	2	0	9	5	7	3
nxt(下一个)	4	3	5	2	6	-1

- 从1开始, 构成的链表: 2 → 5 → 0 → 9 → 7 → 3

现学: 指针(1/4)

- 变量: 类型、里面存的值、地址


```cpp
#include <iostream>
using namespace std;
int main(){
    int x=20;
    cout<<"The value of x is "<<x<<endl;
    cout<<"The address of x is "<<&x<<endl;
    //                              ^^ This is a pointer.

    printf("We use printf for clearer view\n");
    printf("The value of x is %d\n", x);
    printf("The address of x is %p", &x);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
zgw (e) base ~ program cobj  g++ addr.cpp && ./a.out
The value of x is 20
The address of x is 0x16d1c6d4c
We use printf for clearer view
The value of x is 20
The address of x is 0x16d1c6d4c%
```

现学: 指针(2/4)

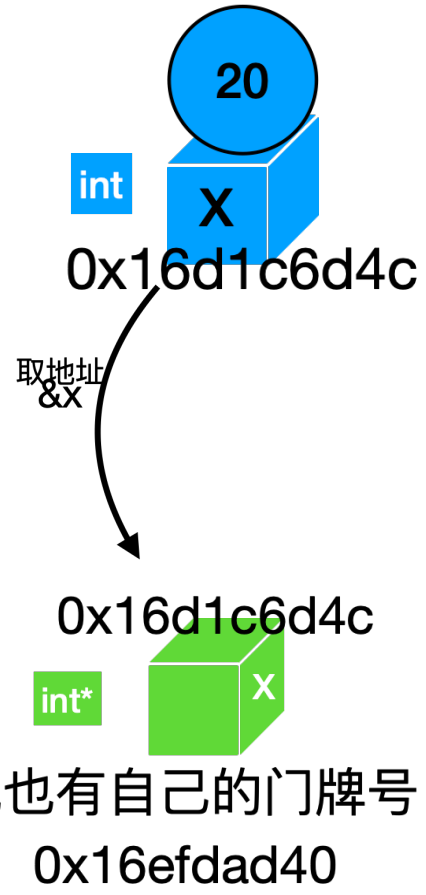
- 指针

  A pointer is a variable that contains the address of a variable.

```
3   int main(){
4       int x=20;
5       cout<<"The value of x is "<<x<<endl;
6       cout<<"The address of x is "<<&x<<endl;
7       //                              ^^ This is a pointer.
8
9       printf("We use printf for clearer view\n");
10      printf("The value of x is %d\n", x);
11      int* p2 = &x;
12      printf("The address of p2 is %p", &p2);
13  }
```

int *p2

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

zgw  (e) base  ~  program  cobj   g++ addr.cpp && ./a.out
The value of x is 20
The address of x is 0x16efdad4c
We use printf for clearer view
The value of x is 20
The address of p2 is 0x16efdad40%
```



现学: 指针(3/4)

- 指针

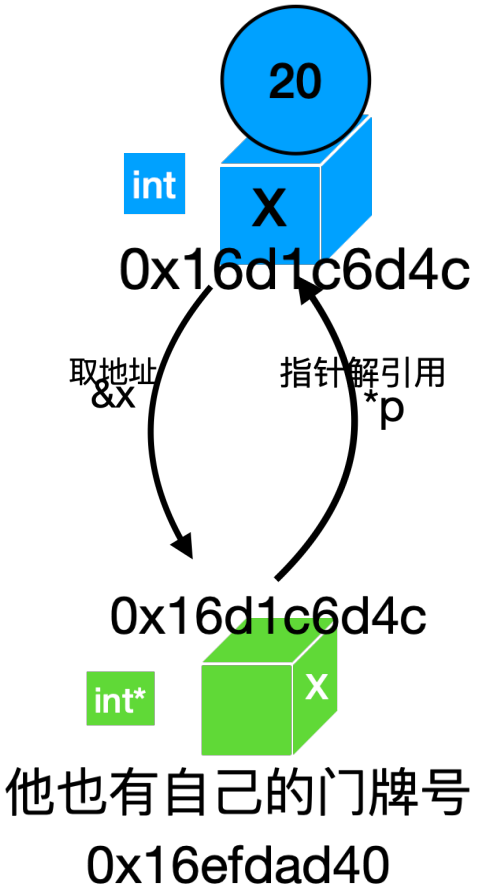
  A pointer is a variable that contains the address of a variable.

addr.cpp > main()

```cpp
#include <iostream>
using namespace std;
int main(){
    int x=20;
    cout<<"The value of x is "<<x<<endl;
    int *p2 = &x;
    int **p3 = &p2;
    **p3 = 1;
    printf("The value of x is %d\n", x);
}
```

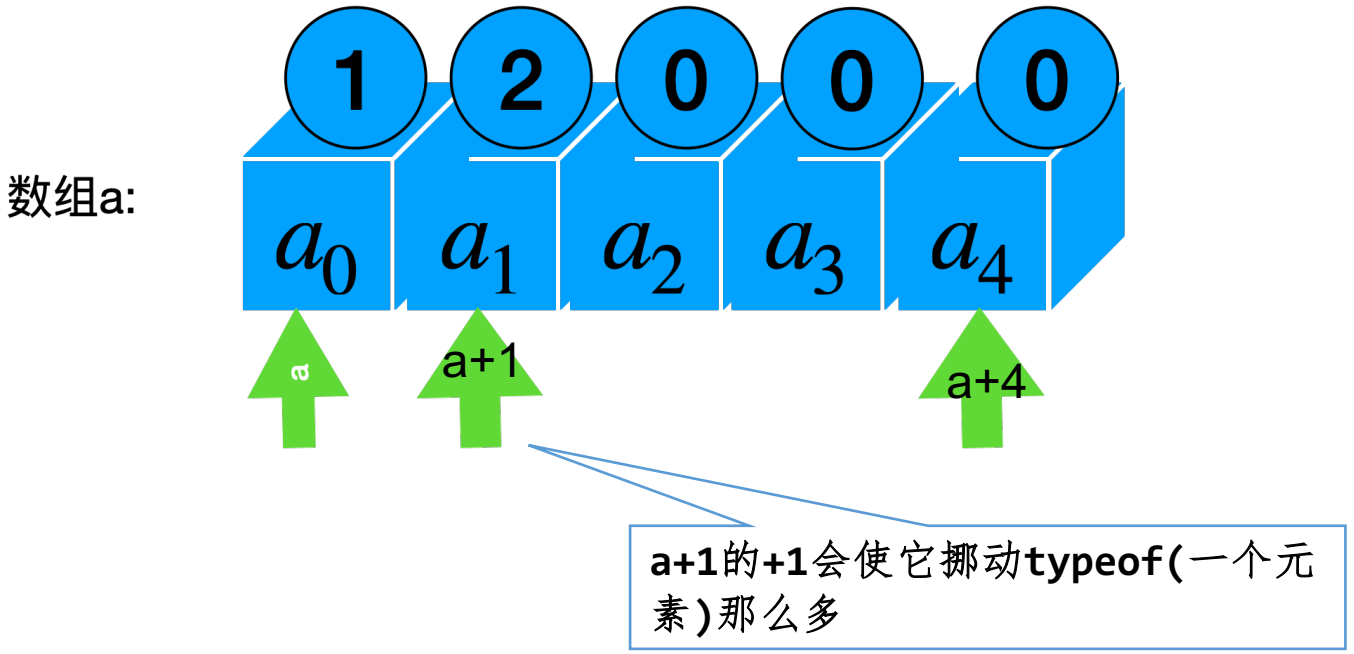
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
zgw (e) base ~ program cobj g++ addr.cpp && ./a.out
The value of x is 20
The value of x is 1
```



现学: 指针与数组(4/4)

- In expressions, the name of the array is a synonym for the address of its first element;
- but an array name is not a variable



链表表示(指针版本)

- 下一个同类在哪? `Node *nxt;`

```
struct Node{
    int val;
    Node *nxt;
}
```

结构的递归

- 思考1(在数据结构上面回答): 如何增/改/删/查链表? 复杂度是多少?
- 思考2: 仿照链表的~~神金~~递归定义, 给出二叉树的定义. 并在计算机中实现其表示.
- 思考3: 一般的树能不能借用二叉树的表示方式?

递归的结构导致递归的过程

- 递归的过程: 到达一个与原问题描述相同的子问题
- 递归的结构: 到达一个与原结构构造相同的子结构

总结

- 算法竞赛程序 ≈ 算法+数据结构
  - 递归的算法/数据结构对大家尤其有趣
- 写复杂程序的时候认清自己的能力
  - 加一些assert
- 程序运行起来是好理解的
  - 观察程序的侧面(调试器/printf)

参考文本和图片引用; 作业

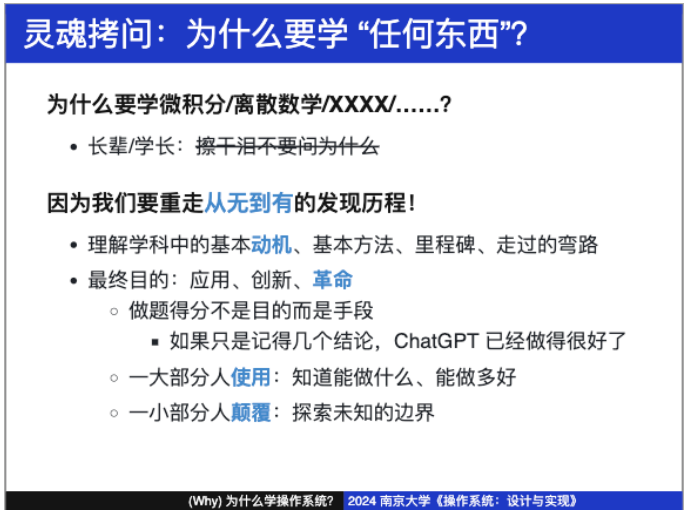
参考文本

- G.J. Sussman et al. *Structure and Interpretation of Computer Programs* Chapter 1.
- Jeff Erickson, *Algorithms.* Chapter 1.
- 李尚志,《线性代数学习指导》, 第一章

图片引用

- 递归精灵: 芙莉莲, TV动画《葬送のフリーレン》小剧场
- Don't Panic: 选自*The Hitchhiker's Guide to the Galaxy*(《银河系漫游指南》)的台词.
- Hanoi斜塔、递归的旋转: Algorithms第一章课后习题
- 表达式树: 选自Crafting Intepreters, NJU ICS 2020 计算机系统基础
- 小学数学课本:选自人教版《数学》一年级上册
- 数学和计算机科学的回答: 选自《操作系统》课程总结, 蒋炎岩
- 调试理论(一辈子基础软件): 改编自TV动画《BanG Dream, It's MyGo!!!!!》第一集

除自己截图外, 其他图片在出现的地方已经标明来源.



作业

- 体会并实现课上的代码片段.