plan

deepwaterooo

October 8, 2024

Contents

1 Notes: 主要是呆校园的时候，小文件方便作点儿笔记

1.1 游戏项目【现，第二第三个】

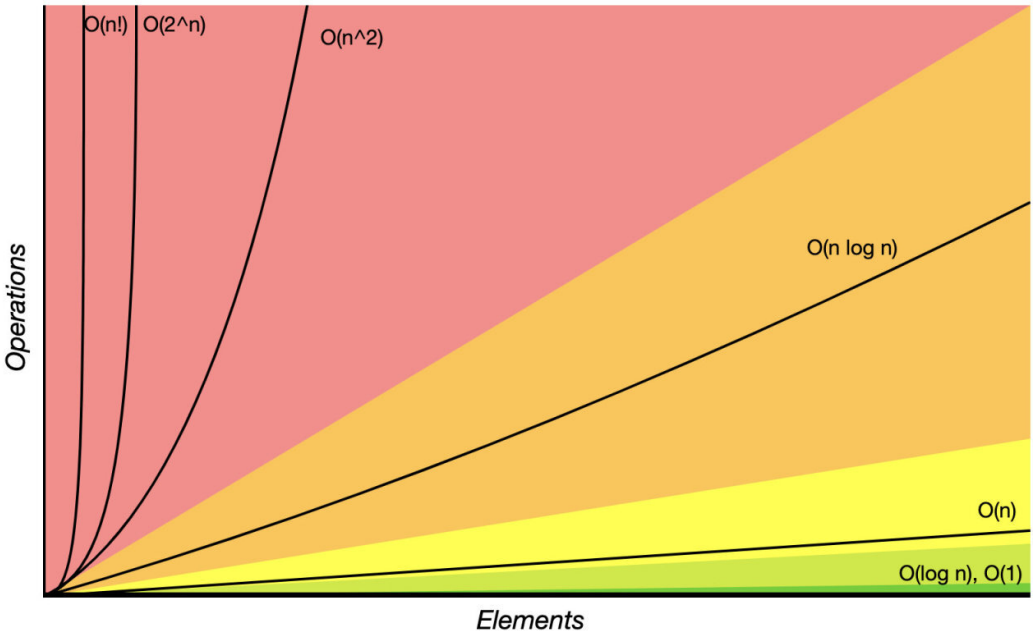
- 现项目：作后期加工，必要的客户端屏幕适配，和自己能够想到的优化
- 构思第二第三个游戏项目

1.2 算法总结题型: 主要是动规，和自己相对陌生的题型

- 动规：不会的题型
- 简单的就不用再浪费时间了

2 数据规模与算法

Input Size	Complexity
50000	O(n)
20000	O(n logn)
1000	O(n ^ 2)
30	O(n ^ 4)
16 (20)	O(2 ^ n)



数据结构	时间复杂度								空间复杂度
	平均				最差				最差
	访问	搜索	插入	删除	访问	搜索	插入	删除	
顺序表	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
栈	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
单链表	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
双链表	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
跳表	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n\log(n))$
散列表	–	$O(1)$	$O(1)$	$O(1)$	–	$O(n)$	$O(n)$	$O(n)$	$O(n)$
二叉搜索树	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
笛卡尔树	–	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	–	$O(n)$	$O(n)$	$O(n)$	$O(n)$
B-树	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
红黑树	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
伸展树	–	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	–	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
AVL 树	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$

算法	时间复杂度			空间复杂度
	最佳	平均	最差	最差
快速排序	$O(n\log(n))$	$O(n\log(n))$	$O(n^2)$	$O(\log(n))$
归并排序	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n)$
Timsort	$O(n)$	$O(n\log(n))$	$O(n\log(n))$	$O(n)$
堆排序	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(1)$
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
希尔排序	$O(n)$	$O((n\log(n))^2)$	$O((n\log(n))^2)$	$O(1)$
桶排序	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
基数排序	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

节点 / 边界管理	存储	增加顶点	增加边界	移除顶点	移除边界	查询
邻接表	$O(\|V\| + \|E\|)$	$O(1)$	$O(1)$	$O(\|V\| + \|E\|)$	$O(\|E\|)$	$O(\|V\|)$
邻接矩阵	$O(\|V\|^2)$	$O(\|V\|^2)$	$O(1)$	$O(\|V\|^2)$	$O(1)$	$O(1)$

类型	时间复杂度						
	建堆	查找最大值	分离最大值	提升键	插入	删除	合并
（排好序的）链表	–	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(m+n)$
（未排序的）链表	–	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
二叉堆	$O(n)$	$O(1)$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(m+n)$
二项堆	–	$O(1)$	$O(\log(n))$	$O(\log(n))$	$O(1)$	$O(\log(n))$	$O(\log(n))$
斐波那契堆	–	$O(1)$	$O(\log(n))$	$O(1)$	$O(1)$	$O(\log(n))$	$O(1)$

3 客户端屏幕适配

4 新项目构思：游戏项目

- 因为狠想要写个【多人网络游戏】，所以，现在能够想到的，不外乎：
  - 两人的五子棋【网络上有了，自己的仓库里也有了】，三人的斗地主【ET 框架有示例】，四人的麻将【其它语言的狠多，主要借视图 UI 图片用】与拖拉机【与三人的斗地主类似】等纸牌或是卡牌游戏, UNO 等卡牌游戏
- 麻将图片：https://github.com/jynnie/majiang

5 双副牌双升 108 张卡牌游戏

- 昨天晚上找见了别人几年前就开发出来的卡五星麻将，所以写麻将游戏的想法就被恶杀在摇篮中。现在再写什么好呢？就只能写【双升拖拉机】了，就是两副牌 108 张来打的拖拉机。现已经 ios iPhone 上有的双升游戏，可能搜索一下设计，写安卓版的双升了，看下能否套用 ET 框架，写成四人网络【客户端与服务器双热更新的】网络游戏
- 现在先搜索必要的框架设计，出版规则比大小算法之类的。
- 【服务器与客户端的同步】：尤其是在分四人牌后，亮主拖底的时候，谁先亮，亮什么主，顺序重要，结果重要。【ET 框架有专用的游戏服，由游戏服来状态同步】在本程序中，采用的是服务器保存所有的状态，处理所有的逻辑。比如，客户端在点击亮主后，做的事情就是发一个消息给服务器，不做任何显示操作，等待服务器传来亮主的消息后再显示
  - 【发牌，公正性】：随机分牌。第一步就是要发牌。需要做到一个完全随机的发牌，就要保证每张牌发到每个玩家手里的概率都是一样的，而且牌的顺序是等概率随机打乱的。程序中采用的是如下的发牌算法（感谢 Dr.Light 提供）：假如有两幅牌，编号从 1 到 108，首先随机选出一个，并且将牌发给玩家，然后将这个编号的牌与 108 号牌交换编号，那

么剩下的牌就是从 1 到 107 号。于是再从中选出一个，重复以上的过程，这样一来，算法的复杂度就是 O(n)。

- 【牌的逻辑 OOD/OOP】设计：三个类，对应单张，拖拉机（对子是长度为 1 的拖拉机），和混合单张与拖拉机，如下图

牌的逻辑

在升级中，牌只有三种形式，一种是拖拉机，一种是单张（对子其实就只是长度为1的拖拉机），另一种就是甩牌时两种牌的混合。在序中，将牌的类型抽象为三个类，如下图所示：（CCardFactory只是创建牌用的，不是具体的牌类型）

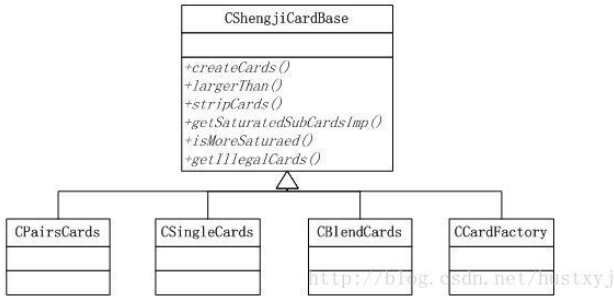


图 1.　　牌的类结构图

里面的几个虚函数主要解决了以下几个问题：

Ø　牌对象的创建。

Ø　两手牌比大小。

Ø　甩牌时判断是否能够甩，即：保证甩出来的各个牌对象在其他玩家中，都是最大的。

Ø　先出拖拉机（对子）时，对方出牌时必须从最长的拖拉机开始出，通俗点的意思就是有对子必须先出对子。在后面的讨论中，我们将其称为出的牌是否饱满。

- 第一手牌与非第一手牌的处理：

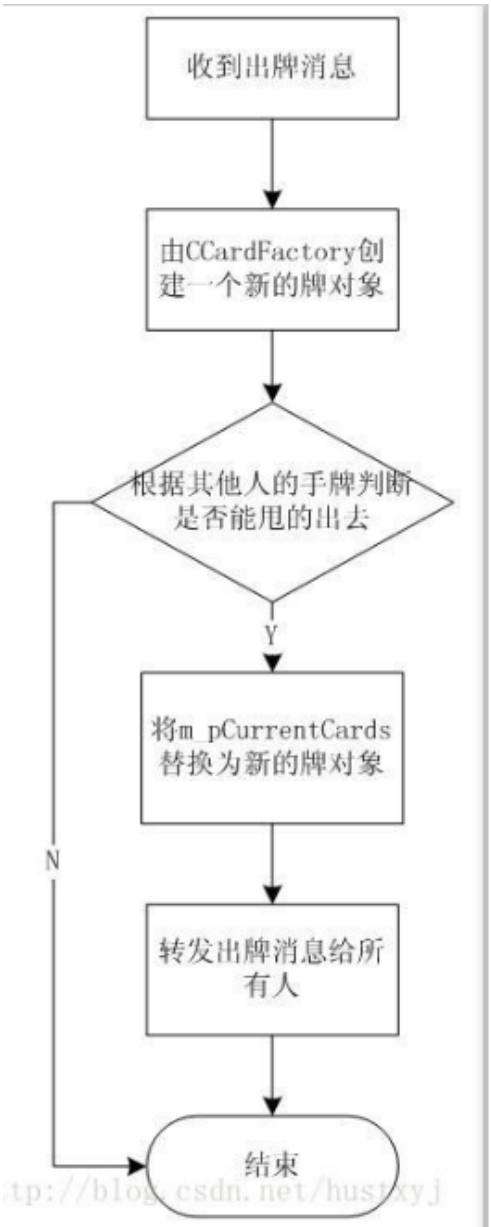


图 9. 第一手牌的处理

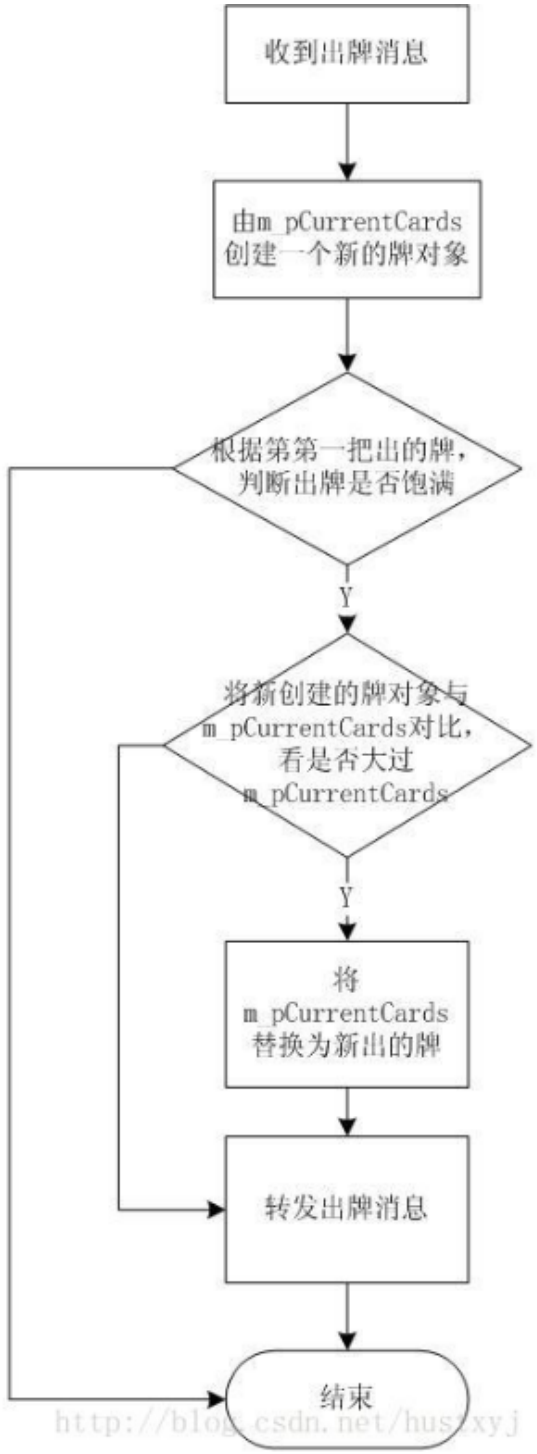


图 10. 非第一手牌的处理

- 参考一个很 Q 的界面：

- 亮着牌打的不好玩，一定会把其它三副牌藏起来的
- http://www.homygame.com/ngscom/help/shengji.htm
- 简易版设计原理：模拟拖拉机（升级）玩法；

  - 1. 创建两副牌的集合：HashMap
  - 2. 创建纸牌：四个花色共 108 张
  - 3. 创建 poker 的 ArrayList 操作集合
  - 4. 创建亮主牌的操作
  - 5. 将所有牌放入牌盒中
  - 6. 创建四个玩家与底牌的集合：HashSet wj1,wj2,wj3,wj4,dipai
  - 7. 洗牌
  - 8. 发牌操作
  - 9. 创建看牌方法
  - 10. 调用方法看牌
- 安桌上的游戏现在是这样的：还要再写一个吗？【活宝妹就是一定要嫁给亲爱的表哥!!!】还是说更为完善或是好玩儿的游戏逻辑？或是 UI 视图画面，或是性能表现？反正一定是套用 ET 框架写得最容易快速方便。【感觉现在这个截图的 UI 长得有点儿丑怪。。】不好看不经典，看了就不想玩儿了。。

- 其它再小的游戏，写都感觉没什么意思：连连看，五子棋、扫雷，祖码。。。

6 其它项目【未必游戏】：安卓蓝牙、安卓视频，post-processing 处理，其它自己能够想到的小项目

7 【算法、快速、归纳、总结】小文件：整理、记载、要点

- **【亲爱的表哥的活宝妹，任何时候，亲爱的表哥的活宝妹，就是一定要、一定会嫁给活宝妹的亲爱的表哥!!! 爱表哥，爱生活!!!】**
- **【亲爱的表哥的活宝妹，任何时候，亲爱的表哥的活宝妹，就是一定要、一定会嫁给活宝妹的亲爱的表哥!!! 爱表哥，爱生活!!!】**

- 亲爱的表哥的活宝妹，惊见一架【刷题机器】。亲爱的表哥的活宝妹，这个周，这个周中、用几天的时间，把亲爱的表哥的活宝妹，感觉生疏、先前没能理解消化透彻的、几个比较难的数据结构、比较难的题型，参看别人的、【刷题机器】的总结、自己手写几个必要的题目、作为理解消化的帮助，亲爱的表哥的活宝妹，自己把这些相对难的题型、解法、总结一遍。
- 亲爱的表哥的活宝妹，感觉生疏、比较喜欢、想要这次总结的题型、主要包括、四大类：
  - **【BIT 与线段树，顺便捡起、差分数组、前缀后缀和等基础】**：这次，能够理解透彻!!
  - **【图：图，总是、狠简单!】**：所有基础概念、基本图论算法、先前不懂的难题，总部理清思路!!
  - **【复杂的、深翻的、各种树!】**：平衡树、四种破烂平衡的情况、左右旋转、使用上下文环境。
  - **【动规题型总结】**：
- 亲爱的表哥的活宝妹，这次，受限于【刷题机器】的总结，是用 C++ 语言写的。亲爱的表哥的活宝妹，这次练习这些复杂题目，可能用 c++ 来写。可是每周比赛的时候，亲爱的表哥的活宝妹，还是喜欢用 java 来刷题。

8 【BIT 树状数组】与【线段树】

- **【亲爱的表哥的活宝妹，任何时候，亲爱的表哥的活宝妹，就是一定要、一定会嫁给活宝妹的亲爱的表哥!!! 爱表哥，爱生活!!!】**
- 【BIT】：【单点修改】【区间查询】【单点查询】【区间修改】等
- 【线段树】：

树状数组是一种支持 **单点修改** 和 **区间查询** 的，代码量小的数据结构。

什么是「单点修改」和「区间查询」？

假设有这样一道题：

已知一个数列 a，你需要进行下面两种操作：

- 给定 x, y，将 $a[x]$ 自增 y。
- 给定 l, r，求解 $a[l \dots r]$ 的和。

其中第一种操作就是「单点修改」，第二种操作就是「区间查询」。

类似地，还有：「区间修改」、「单点查询」。它们分别的一个例子如下：

- 区间修改：给定 l, r, x，将 $a[l \dots r]$ 中的每个数都分别自增 x；
- 单点查询：给定 x，求解 $a[x]$ 的值。

注意到，区间问题一般严格强于单点问题，因为对单点的操作相当于对一个长度为 1 的区间操作。

普通树状数组维护的信息及运算要满足 **结合律** 且 **可差分**，如加法（和）、乘法（积）、异或等。

- 结合律：$(x \circ y) \circ z = x \circ (y \circ z)$，其中 $\circ$ 是一个二元运算符。
- 可差分：具有逆运算的运算，即已知 $x \circ y$ 和 x 可以求出 y。

- lowBit(): 这里注意：lowbit 指的不是最低位 1 所在的位数 k，而是这个 1 和后面所有 0 组成的 2^k。

- 怎么计算 lowbit？根据位运算知识，可以得到 lowbit(x) = x & -x。
- lowbit 的原理
  - 将 x 的二进制所有位全部取反，再加 1，就可以得到 -x 的二进制编码。例如，6 的二进制编码是 110，全部取反后得到 001，加 1 得到 010。
  - 设原先 x 的二进制编码是 (...)10...00，全部取反后得到 [...]01...11，加 1 后得到 [...]10...00，也就是 -x 的二进制编码了。这里 x 二进制表示中第一个 1 是 x 最低位的 1。
  - (...) 和 [...] 中省略号的每一位分别相反，所以 x & -x = (...)10...00 & [...]10...00 = 10...00，得到的结果就是 lowbit。
- 【TODO】：补加，BIT 的什么什么狗屁狗屁性质之类的。。。
- 【TODO】：重点掌握：下午找几个题目、写写玩玩儿，体会差别
  - BIT 适用于、可解的几大题型
  - BIT 与线段树，适用题型的区别、本质区别——算法时间、空间复杂度上的区别
- 【TODO】：应用于数组上的、二维 BIT, 等亲爱的表哥的活宝妹，写熟练了一维、完全掌握后，再扩展