

**算法作业笔记**

**学院： 计算机科学学院**

**班级：计算机科学与技术1802班**

**姓名： 段星雨**

目录

[1. 算法基础 1](#_Toc39789028)

[1.1数据结构 1](#_Toc39789029)

[1.2递归 1](#_Toc39789030)

[1.3蛮力（暴力） 1](#_Toc39789031)

[1.4时间与空间 1](#_Toc39789032)

[2.一般算法 2](#_Toc39789033)

[2.1分治 2](#_Toc39789034)

[2.2贪心 2](#_Toc39789035)

[2.3动态规划 3](#_Toc39789036)

[2.4回溯 3](#_Toc39789037)

[2.5分支限界 4](#_Toc39789038)

[3.其他算法 4](#_Toc39789039)

[3.1双指针 4](#_Toc39789040)

[3.2获取每一位数字或进制转换 5](#_Toc39789041)

[3.3 KMP 5](#_Toc39789042)

[3.4转化 5](#_Toc39789043)

[4.算法优化 5](#_Toc39789044)

[4.1使用空节点 5](#_Toc39789045)

[4.2使用API和简化思路 5](#_Toc39789046)

[4.3转化思路 6](#_Toc39789047)

[4.4模块化编程 6](#_Toc39789048)

[4.5空间与时间互换 6](#_Toc39789049)

# 1. 算法基础

## 1.1数据结构

在数据结构中有许多的概念用于算法，比如以下概念：

栈：先进后出

队列：先进先出

树：既可以储存数据，也可以表示一些算法的过程

图：可以用来表示结点之间的关系

DFS：图或树以深度来搜索，直到搜索到叶子结点才返回该节点的父节点

BFS：图或树以广度来搜索，以层来搜索所有结点

……

## 1.2递归

用方法本身调用自己，递归聚焦于逻辑，而非过程，只要有大体思路即可，不必要追究具体的实现。

递归可以很大程度的简化代码，但相对的，在上一层未结束时再次调用方法，这就导致在某些数据量庞大的算法中，递归占用的内存会较大，但是递归仍然可以简化算法的步骤。

**典例：**

1.汉诺塔

void HanNuo(n, a, b, c)

{ if (n == 1) then print(n, a, c);

else then{HanNuo(n - 1, a, c, b); print(n, a, c); HanNuo(n - 1, b, a, c); }

2.阶乘或数列求和

## 1.3蛮力（暴力）

蛮力法是一种正常逻辑的方法，但并不适用于所有情况。因为在有一些情况下，我们可以转化思路来简化算法。

例如在计算一个数是否为5的倍数，蛮力算法会将每个数都对5取余，但其实只需要判断这个数的个位是否是0或5即可。在数据量小的时候可能两种方法差异不大，但数据量一多，时间复杂度就会上去。

但对于一部分问题来说只能使用蛮力解，其实也就是一种演绎的方法。

## 1.4时间与空间

评价一个算法的好与坏，时间和空间复杂度绝对是考虑的重点。

在有些题目中会限制使用的空间和时间，最好的情况下是两者都变小。但在某些情况下只有其中一种会超过限制，一般会用到两种方法：

空间换时间：例如动态规划或记忆化搜索，使用额外的空间来记录某些数据，用来在之后重复的子问题出现时减少时间

时间换空间：一般很少出现这种情况，因为一般算法还是以时间为重。时间换空间跟多出现在一些限制了空间的情况中

# 2.一般算法

## 2.1分治

分治，分而治之，顾名思义就是拆分，但拆分的子问题与大问题同类，更多情况下可以使用递归。

**典例：**

1. 二分法

二分查找：每次比较中间元素，然后确定新的范围

BinarySearch(a, key) {

// key要查找的数字，a排好序的数组

low = 0; high = a.length - 1;

while (low <= high) do {

mid = low + (high - low) / 2;

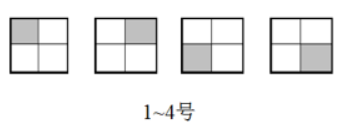
if (key < a[mid]) high = mid - 1;

else if (key > a[mid]) low = mid + 1;

else return mid; }

return -1; }// 未找到或错误返回-1

2.残缺棋盘

残缺棋盘是一个有2^kX2^k. (k≥1)个方格的棋盘，其中恰有一个方格残缺。下图给出k=1时各种可能的残缺棋盘，其中残缺的方格用阴影表示。”

1)两个三格板不能重叠

2)三格板不能覆盖残缺方格，但必须覆盖其他所有方格

解法：将棋盘拆分，直到为2\*2的时候，也就是最小的子问题，然后填充，之后返回上一层，dfs的思想来遍历整个棋盘

## 2.2贪心

贪心算法就是在每一步挑选满足条件的最优情况，而不关注总体情况。所以贪心算法要经过证明才可使用。

在一般问题中，贪心算法找到的虽然不是最优解，但可以找到较优解，贪心算法有时可以估算范围下界。

**典例：**

1.最小生成树：prim算法和kruskal算法

Prim：每一步挑选已选节点可到的相邻未选节点到已选节点最小值对应的点

Kruskal：每一次在所有边中挑选权值最小的边，并且这条边相连的点至少有一个不在已挑选的点集中

2. 假设某社团某一天要 组织n个活动E= {1, 2,3...n}，其中每个活动都要求使用同一礼堂，而且在同一时间内只有一个活动能使用这个礼堂。每个活动i 都有一个要求使用礼堂的起始时间si和结束时间fi,且有si<fi若区间(si, fi) 和(si, fi)不相交，则称活动i与活动j是相容的。

贪心算法：以结束时间排序，从最早结束开始选，依次找到下一个可以相容的点

## 2.3动态规划

动态规划就是每一步决定与之前的子问题有关系，并且在一步中有多种情况时可以使用贪心选出最优情况。在动态规划的搜索树中会出现相同子问题的节点，所以可以提前使用dp数组来储存重复子问题的结果，这样可以大大简化过程。

**典例：**

1.斐波那切数列

Fib(1)=1，Fib (2)=1, Fib (n)=F(n - 1)+F(n - 2)

2.组合数

C(n,m)=C(n-1,m-1)+C(n-1,m)

3.上楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

动态规划：第 i 阶可以由以下两种方法得到：

在第 (i-1) 阶后向上爬1阶。

在第 (i-2) 阶后向上爬2阶。

dp(i)=dp(i-1)+dp(i-2) dp(1)= 1,dp(2)= 2

## 2.4回溯

回溯其实就是暴力法的简化，DFS可以在中途判断找到错误直接返回，而不用找到最后的每种情况再判断。

**典例：**

1.n皇后

N叉树搜索，然后每次新加入棋子后判断是否满足，若不满足直接弹出节点，寻找下一个答案。

2.马踏棋盘

马有8种走法，使用八叉树遍历，当点已经走过则回溯

## 2.5分支限界

分支限界与回溯类似，但使用的是BFS，每个节点只会遍历一次，一般寻找最优解，而不是全部解。

**典例：**

单源最短路径：

问题：给定一个带权有向图G=（V,E），其中每条边的权是一个实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到其他所有各顶点的最短路径长度。这里的长度就是指路上各边权之和。

方法：

1.首先将源点入队，然后找到源点的邻接点，更新邻接点的距离并入队，然后源点出队

2.之后再找队首元素的邻接点，更新邻接点的距离，如果某个点本次距离比之前的距离要长，则“剪枝”，入队未在队列中邻接点，然后队首元素出队

3.重复2，直到所有元素的距离都被更新

# 3.其他算法

## 3.1双指针

反转链表：

删除排序数组中的重复项：

removeDuplicates(nums) {

//nums为已排序数组

flag = 0;//记录之前的不同元素的节点

for i = 1 to nums.length do {if (nums[i] != nums[flag]) nums[++flag] = nums[i]; }

return flag + 1; }

## 3.2获取每一位数字或进制转换

获取每一位数字与进制转换大同小异，都是用数字去对进制取余，得到当前进制下末尾的数字，然后再用数字除进制，则可以消去当前进制下最后一位的数字

## 3.3 KMP

Kmp算法也就是indexOf，即找到次字符串在主字符串第一次出现的索引。

Kmp核心是找到next数组，即次字符串每次遍历回溯长度。因为在字符串匹配时，发现字符串没必要每次都全部回溯到开头。次字符串的不同长度的子串开头结尾可能存在相同公共子串，所以可以只回溯公共子串的长度即可。其实在每次回溯时使用了贪心算法，所以kmp算法大大简化了暴力寻找的难度。

## 3.4转化

判断一个数能否被4,7,9整除。

方法：使用s进行计数，能被4,7,9整除分别用2^0,2^1,2^2进行记录，并加到s上，最后根据s的值进行输出

# 4.算法优化

## 4.1使用空节点

1.在数组中头结点可以空下来，可以做监视哨或者方便数组索引对应实际情况

2.链表可以使用空的头结点，方便在双指针遍历时便于确认临界条件

3.二维数组可以在最外层加一圈元素，方便判断越界条件，也可以避免一些空指针异常

## 4.2使用API和简化思路

使用语言语言准备好的api，

例如：

1.c语言中的memset函数初始化相同格式元素

2.c语言用函数返回值1或0来当判断条件

3.用三元运算符“……？……：……”来替换一些if……else语句

4.java中牢记set，list和collection的子类中不同的数据储存方法，来简化算法。

例如HashSet中会自动排序，并且在数据阈值到8是会自动转换为红黑树方便查找。

5.java使用map来储存键值对，方便寻找一些元素的对应关系

6.c语言中可以使用宏定义来定义一些语法或一些变量，方便调用和检查程序

7.调试时使用debug监视变量的变化情况，更易于发现错误

8.有时可以使用一些位运算来提高程序的效率，方便指数增长和越界判断

……

## 4.3转化思路

转换思维，使用数学思路简化计算，例如矩阵计算，整除等情况

## 4.4模块化编程

使用模块化编程，可以使程序更加直观，更加方便debug，之后如果有其他情况用到该算法方便修改和调用

## 4.5空间与时间互换

见1.4.