上半节 审题+矩阵快速幂

下半节 字符串匹配+复习

①形式化（能不看就不看）

②审题（花足够的时间看题，不要急着写代码）

最小数＋k

如果加到了第二小的数，停止

此时，最大值减k，最大值减k后可能会小于最小值加k后的值

考虑最小值增加后不能超过(最小值＋最大值)/2

min（第二小的值，(min+max)/2）

sort(a, a + n);

cout << setprecision(1) << ans;

快速幂

求一个乘法 2^128

方法一：把2乘128次（128次乘法）

方法二：2→2^2→2^4→2^8→2^16→…→2^128（7次平方）

为什么可以快速幂

快速幂：(2^2)^2＝2^2×2^2 ← 乘法结合律

因为乘法有结合律，所以才可以对乘法使用快速幂，让原本n次计算简化为log n次

矩阵快速幂

矩阵乘法：结合律

如果有一个新的矩阵运算，不是矩阵乘法，但具有结合律（类乘法（具有结合律的运算））

线性递推 → 构造矩阵乘法

任务：高考数学数列应用题

寻找一个线性递推关系（用n-1的答案来描述n的答案）

答案：数列的通项公式

增加状态（类比高考数学求an通项公式时增加bn、cn状态来构造递推关系）

an：表示红和绿都为偶数的方案数

bn：表示红和绿都为奇数的方案数

cn：表示红和绿一偶的一奇的方案数

an=2\*an-1+cn-1

线性递推式 → 常矩阵左乘列向量 → 对常矩阵使用快速幂 → 答案

类矩阵乘法

max 加法 具有结合律

n太大 → 必须缩短时间（让n→log n）→考虑使用快速幂→考虑递推运算是否具有结合律

例4中把矩阵乘法的加变成max，矩阵乘法的乘变成加

字符串匹配：从一个长字符串里找一个短字符串

例：

长字符串：1234abcd567

短字符串：abc

匹配成功

目标：bool match(string longStr, string shortStr){ }

设长字符串长度为n，短字符串长度为m

暴力：O(mn)

优化：O(m+n)

（KMP算法）

长：11111111111111

短：11112

1111

只有当我们的短字符串的(前4位这个子串)对应的部分匹配值为3（2）时

我们才需要考虑从长字符串的第2（3）位开始继续尝试匹配

↓

根据短字符串预处理得到的部分匹配值来决定

下一次在长字符串中尝试匹配时，将首指针往后移动多少位

第一轮成功匹配了4位，如果我们要把首指针向后移动1位，部分匹配值可以大到3

根据部分匹配值

第一轮：成功匹配了短字符串的前4位（说明长2-4和短2-4匹配）

第二轮：如果能从长字符串的第二位开始匹配成功

那么长字符串的2-4位一定和短字符串的1-3位匹配

综上：短1-3和短2-4匹配才能使从长第二位开始有可能匹配成功

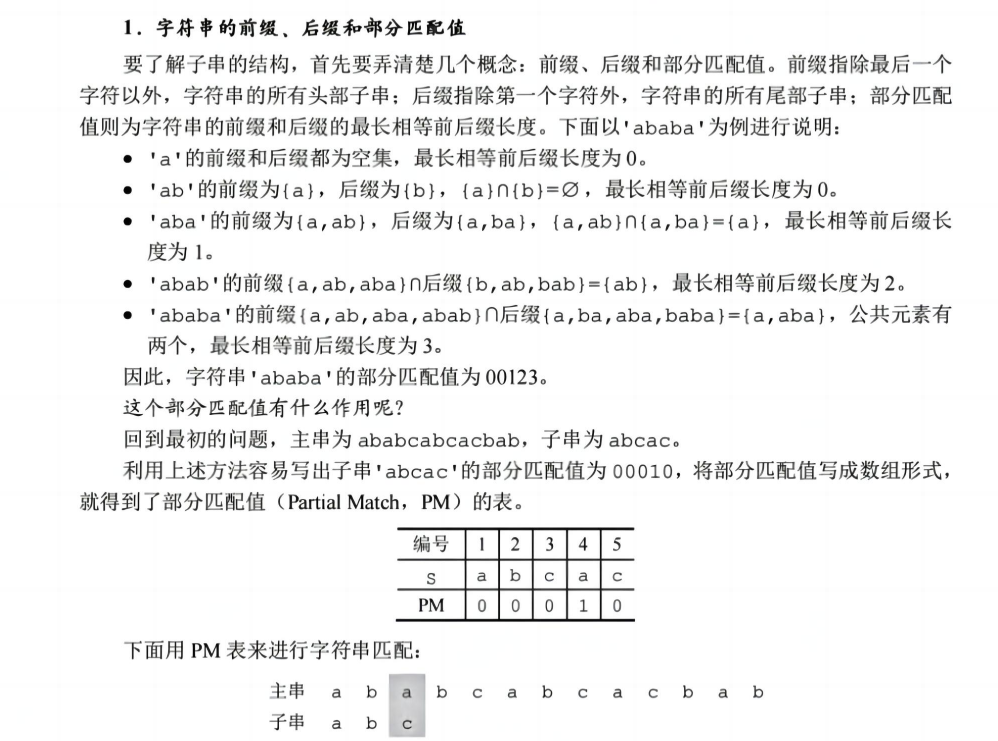
上面这条性质只和短字符串本身有关，而和长字符串无关

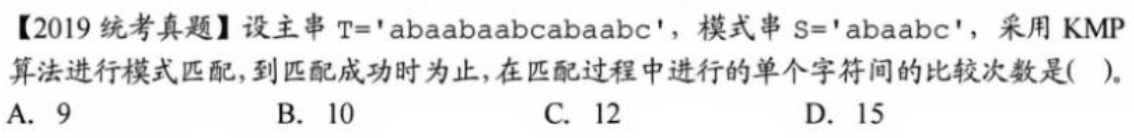
在长短匹配之前，做一个短字符串的预处理

对短字符串的每一个子串，求他的最大前后缀匹配

（abcd的子串：a、ab、abc、abcd）

（aba的最大前后缀匹配：第一位a=第三位a（1√），前两位ab≠ba（2×））





2019考研数据结构真题

abaabc预处理：

子串a：部分匹配值为0

子串ab：部分匹配值为0

子串aba：部分匹配值为1

子串abaa：部分匹配值为1

子串abaab：部分匹配值为2

子串abaabc：部分匹配值为0

abaabaabcabaabc

abaabc 比较6次，失配，成功部分长度为5，移动首指针3位

abaabc 比较4次，成功

一共比较了10位，选B

KMP算法√

T1送分（审题）

T2优化

容器的选择/差分和前缀和/树状数组和线段树（n→log）/二分（n→log）/尺取/单调栈或队列/快速幂（n→log）

T3图论

BFS/DFS/树的直径/并查集/最小生成树/最短路/拓扑排序/强连通分量（SCC）

T4策略

贪心/动态规划

贪心（混分）：调整（贪心算法领先），反证法

动态规划（正解）：线性（一维状态），坐标（多维状态），背包（把值作为状态），区间（左右指针），状压（多维→一维），树形（特定节点作为根的树），矩阵

先考虑答案需要怎么递推，怎么通过子问题来转化，根据这些来设置状态

如果转化出现了问题，考虑怎么设置辅助状态，帮助完成转化过程

动态规划优化：

1.空间：滚动数组（一个数组（正序、逆序），两个数组（交替更新））

2.时间：最优容器，前缀和/线段树，单调栈/单调队列，快速幂

BFS技巧：使用的场景（最短路，迷宫问题），超级源点，双向BFS

DFS剪枝：可行性剪枝，最优性剪枝

树的直径：两遍搜索，DP

并查集：O(1)合并，O(树高)查找

最小生成树：Kruskal，Prim

多源点最短路Floyd：基于矩阵DP

单源点最短路Dijkstra：基于贪心，要求所有边必须是非负的

单源点边权可负Bellman-Ford：基于松弛，每一个点的最短路长度不超过n-1，经过n-1次松弛一定能使每个点的距离都是最短距离

SPFA：Bellman-Ford的优化，基于队列，优化掉了一些显然没有用的松弛操作

差分系统：基于最短路问题，把最短路问题转换为不等式约束关系

Kahn算法：不断地取出度（入度）为0的点，并且删除其所有边，（判断）得到一个拓扑序列

DFS求拓扑序列算法

在图中以任意的顺序进行DFS，得到了一个DFS森林

树1：2、3、4

树5：6、7

//假设一定存在拓扑排列

树内部一定满足拓扑序

1一定不能到5、6、7，否则树1里面就会有5、6、7中的点

但是5、6、7可能可以到1

1→2→3→4→5→6→7

5 6 7 1 2 3 4

SCC：两遍DFS（Kosaraju模拟）

1 5

2 3 4 6 7

合并：只将far[5]置为1，6和7的far依然是5，当查找6的时候一路查上去查到1，此时再将far[6]置为1

sort：必须是重载过小于的结构体或类才能使用，如何反向（直接取负）

pair：嵌套，来方便的进行不同权重的比较（重载了先比较第一个再比较第二个）

线性容器vector/string

树形容器set/map（基于红黑树，维护了有序关系，遍历必须通过迭代器，O(logn)插入和查找）

如果不需要维护有序关系，只当成集合来用（unorderd\_map/set，基于哈希表，O(1)插入和查找）

容器适配器stack，queue，priority\_queue（O(n)初始化，语法）

数组本身：优势在于更新（O(1)），劣势在于查找区间和（O(n)）

前缀和数组：优势在于查找区间和（O(1)），劣势在于更新（O(n)）

树状数组/线段树：二者的折中

二分：整数二分（有序序列的下标，查找目标位置），浮点二分（查找一个满足单调的具体的值）

尺取：查找一个目标区间，同时移动区间的左右指针，O(n2)→O(n)

单调栈和单调队列：维护区间内的最值，使区间最值的查找在O(1)时间完成