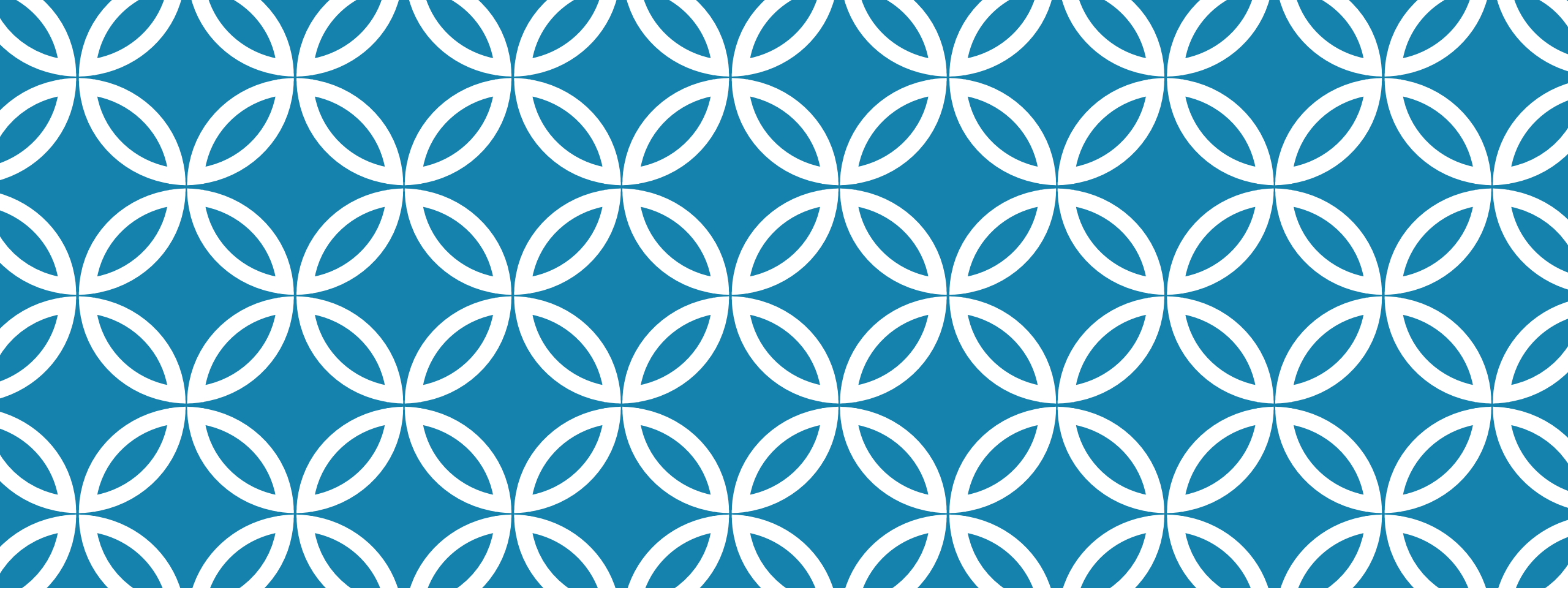
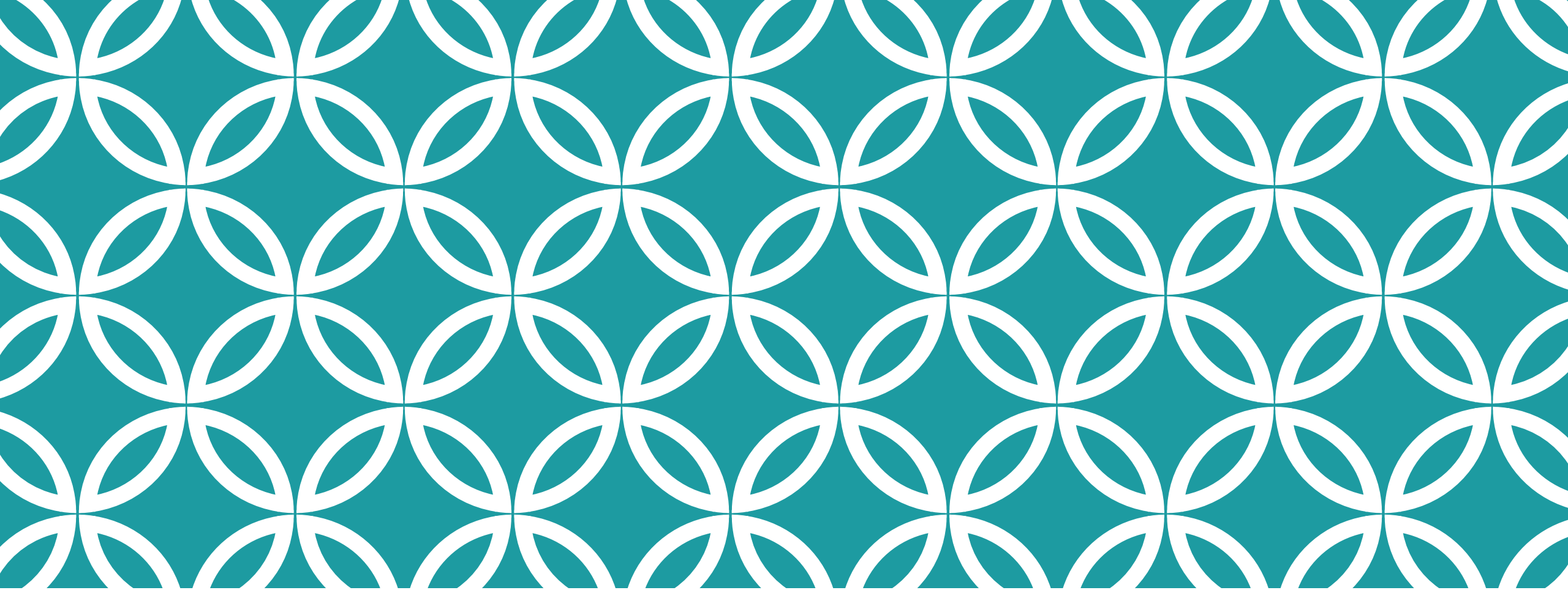


基础算法专题选讲2

字符串算法



基础知识

库函数、表达式

字符串

C语言

- C语言中没有真正意义的字符串，而是使用char数组或char指针代替
- 定义方式：char str[len]；char *str;

输入输出接口

条件	普通	整行	缓冲区
输入	scanf	gets	fread
输出	printf	puts	fwrite

- 注意：用scanf时，空格、Tab等分隔符会切断字符串，puts会额外输出一个换行符，使用fread一次可能读入多行

字符串

<cstring>(string.h)

函数	功能
memcpy	复制一段内存
memset	给一段内存赋值
strcpy	复制字符串
strncpy	复制字符串中的若干字符
strstr	求一段子串
strcmp	比较两个字符串
strlen	求字符串长度
NULL	空指针（0）

字符串

C++语言

- C语言中字符串变成了一个类
- 定义方式： **string str;**

输入输出接口

条件	普通	整行	char数组
输入	cin	getline	read
输出	cout	puts	write

- 注意：用cin时，空格、Tab等分隔符会切断字符串，最好把同步关掉，不然会很慢

字符串

<string>

操作	功能
=	拷贝字符串
length	求字符串长度
reserve	反转
compare	比较大小
insert	插入子串
c_str	转成char数组
copy	复制部分字符
substr	求子串

表达式

表达式

- 算术表达式通常有前缀、中缀、后缀三种表示方式
- 中缀表达式即我们书写用的表达式
- 前缀表达式被称为波兰式，后缀表达式被称为逆波兰式

条件	示例
前缀表达式	3 * (1 - 2)
中缀表达式	* 3 - 1 2
后缀表达式	3 1 2 – *

表达式

后缀表达式求值

- 后缀表达式可以利用堆栈求值，与求前缀表达式方法类似

求解步骤

- 1. 建立一个用于存数的栈，逐一扫描该后缀表达式中的元素
- (1) 如果遇到一个数，则把该数如栈
- (2) 如果遇到运算符，就取出栈顶的两个数进行运算，把结果入栈
- 2. 扫描完毕后，栈顶只剩一个元素，就是结果

表达式

中缀表达式求值

- 中缀表达式可以先转为后缀表达式再求值

求解步骤

- 1. 建立一个用于存运算符的栈，逐一扫描中缀表达式
- (1) 如果遇到一个数，则输出该数
- (2) 如果遇到左括号，则左括号入栈
- (3) 如果遇到右括号，则不断取出栈顶并输出，直到栈顶为左括号，然后将左括号出栈
- (4) 如果遇到运算符，只要栈顶的运算符优先级不低于新符号，就不断取出并输出，最后把新符号入栈
- 2. 依次输出剩余符号，就能得到等价后缀表达式

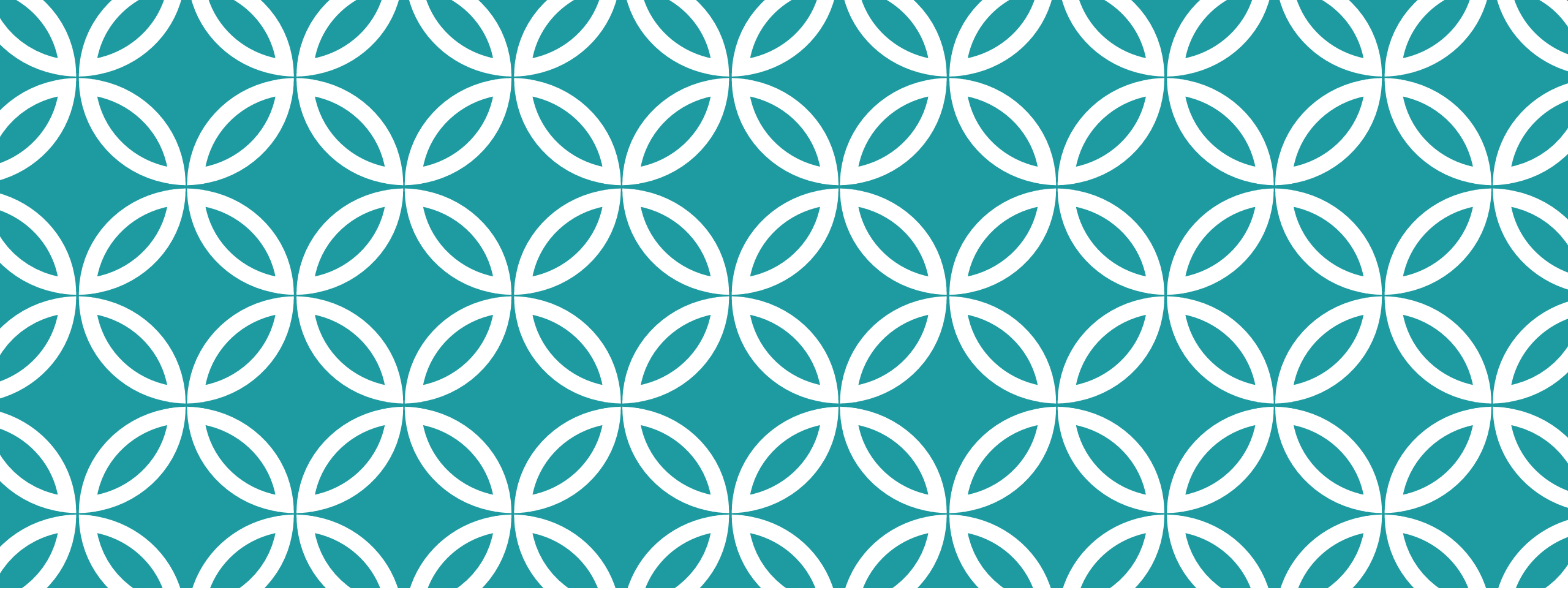
表达式

中缀表达式求值

- 中缀表达式也可以直接递归求解，复杂度$O(n^2)$

求解步骤

- 考虑子区间[L,R]的值
- 1. 在L~R中考虑没有被任何括号包含的运算符
- (1) 若存在加减号，选最后一个，分成左右两半递归，计算结果并返回；
- (2) 若存在乘除号，选最后一个，分成左右两半递归，计算结果并返回；
- 并输出，最后把新符号入栈
- 2. 如果不存在没有被任何括号包含的运算符
- (1) 若首位都是括号递归求解区间[L+1,R-1]，返回结果
- (2) 首尾不是括号，说明只有一个数字，直接返回



HASH

Hash函数、字符串hash、冲突

HASH函数

Hash表

- Hash表又称散列表，一般由hash函数与链表结构共同实现
- 简单的说就是将任何数据类型与一定范围内的整数形成了映射

支持操作

- 计算hash函数并插入hash表
- 计算hash函数并在表中查找该元素

HASH函数

Hash函数

- 将一组数据映射为一个有限范围内的整数
- 尽量避免冲突（即数据不同时函数值尽可能不同）

例

- 最简单的hash函数f(x) = (x mod P) + 1
- 这个函数将一个大整数x映射为了1~P范围内的较小整数，只要x随机生成，hash表中的元素也是均匀分布的
- 但是很多题目数据往往都是人工构造的，所以需要更好的hash函数

字符串HASH

字符串hash

- 对于一个字符串，我们可以将其看成一个数组，因此求字符串hash就是求数列hash
- 当然字符串中不同的元素个数一般只有26个，所以字符串hash的性质和一般的数列hash还是有一定区别的

字符串HASH

自然溢出hash

- 我们将字符串看成一个P进制数，一般P取131或13331（容易称为原根）
- 我们可以直接将这个P进制数对2^{64}取模(即自然溢出)的结果作为字符串的hash值

- 这样字符串的hash就满足如下性质：
- $H(S+T)=H(S)*P^{|T|}+H(T)$
- 自然溢出hash可以
- 我们可以预处理P的幂或者使用快速幂迅速求解字符串的hash值

字符串HASH

例题：兔子与兔子

- 长度为n的DNA序列（字符串），Q组询问，每组询问给出l1,r1,l2,r2，判断区间[l1,r1]和区间[l2,r2]的区间是否相等

字符串HASH

例题：兔子与兔子

- 长度为n的DNA序列（字符串），Q组询问，每组询问给出l1,r1,l2,r2，判断区间[l1,r1]和区间[l2,r2]的区间是否相等

- 我们可以维护整个字符串所有前缀的hash值F[i]，然后计算子串
- Hash[l1,r1] =(F[r]-f[l])*P^{-l}
- P的-l次方虽然不能直接计算，但是我们只要知道了l1和l2直接的差
- 将次数低的hash值乘上二者的次数之差就可以进行比较了
- 时间复杂度O(n)

字符串HASH

例题： palindrome(poj3974)

- 给定一个长度为n的字符串S，求它的最长回文子串

字符串HASH

例题：palindrome(poj3974)

- 给定一个长度为n的字符串S，求它的最长回文子串

- 我们设字符串正向的hash值为H[S]，反向的hash值为R[S]
- 如果一个字符串是回文串，那么有R[S]=H[S]成立
- H[S]和R[S]可以通过O(n)预处理O(1)求出
- 现在问题就变成如何快速求最长回文

字符串HASH

例题：palindrome(poj3974)

- 给定一个长度为n的字符串S，求它的最长回文子串

- 回文串分奇数长度和偶数长度两类
- 只要在每两个字符之间插入一个新的字符构造出一个长度为2n+1的串就可以了，这样都变成了奇数长度的回文
- 我们可以枚举字符串的对称中心，然后二分答案求解
- 时间复杂度O(nlogn)
- 事实上，使用Manacher算法可以O(n)求解此问题

HASH冲突

hash冲突

- 一般来说，Hash函数的定义域是远远大于值域，一个长度为字符串1000的字符串有127^1000种不同的形态，而自然溢出hash值只有2^64种不同的值
- 所以只要取的字符串够多，总能找到两个hash值相同但内容不同的字符串，我们称之为冲突

HASH冲突

生日问题

- 一个班有n个人，那么这个班中存在两个学生在同一天生日的概率是多少？

HASH冲突

生日问题

- 一个班有n个人，那么这个班中存在两个学生在同一天生日的概率是多少？

- 假设生日各不相同，则这n个人的生日相当于365天的一个排列，因此所有人生日各不相同的概率为A(365,n)/356^n
- 当n等于23时，这个值大约为0.5；当n=30时，这个值大约为0.3
- 也就是说，23个人中存在两个人同一天生日的概率就达到了50%，而30个人时就达到了70%

HASH冲突

生日攻击

- 有n个函数取值的hash函数，在$\sqrt{n}$规模的数据是就有50%的概率出现至少一对冲突
- 在密码学领域，利用这一现象寻找碰撞的方法被称为生日攻击

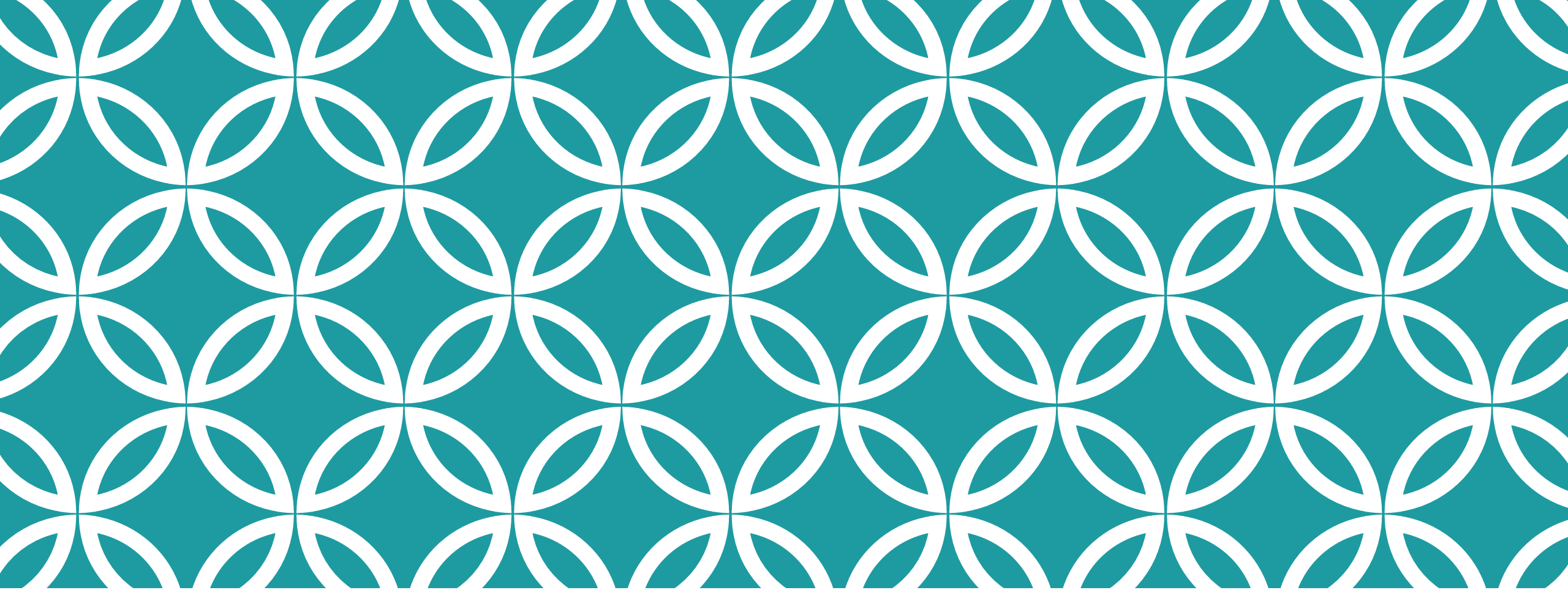
hash冲突的处理

- 基本上来说hash冲突时不可避免的，除非能够提供运行时间的平方以上空间，这并不现实
- 因此所有的hash表都需要解决冲突问题

HASH冲突

hash冲突的解决方法

- 1. 挂链：直接在每个hash值后面挂一个链表
  - 优点方便，缺点模数没选好容易被卡
- 2. 线性探查法：如果一个格子被占，则放到离他距离最近的空格里
  - 优点点方便，缺点数据容易集中
- 3. 二次探查法：将线性探查法的跳的步数从$\pm x$改为$\pm x^2$
  - 比线性探查法数据分布更均匀
- 4. 双哈希：用两个不同的hash值来标定一组数据
  - 最好两个hash算法也不同



TRIE

字典树Trie

TRIE

Trie

- Trie又称为字典树，是一种用于实现字符串检索的数据结构

初始化

- 包含一个根节点，所有字符指针指向空节点

插入操作

- 每次插入时从根节点P开始，扫描S中的每个字符c:
- 1. 若P的c字符指针指向一个已经存在的节点Q，令P=Q;
- 2. 若P的c字符指针指向空，则新建一个节点Q，然后令P的c字符指针指向Q，然后令P=Q。
- 在最后一个字符节点加上一个结尾标记（比如在S末尾加个标记字符）

TRIE

检索操作

- 检索时从根节点P开始，扫描S中的每个字符c:
- 1. 若P的c字符指针指向一个已经存在的节点Q，令P=Q;
- 2. 若P的c字符指针指向空，则说明该字符串不存在，结束检索。
- 检索时不要忘记判断结尾标记

复杂度分析

- 时间复杂度$O(\Sigma len)$
- 空间复杂度$O(\Sigma len)$

可持久化

- 每次如果插入时从根节点开始就新建所有的节点，就可以实现可持久化Trie

TIRE

例题： The XOR Largest Pair

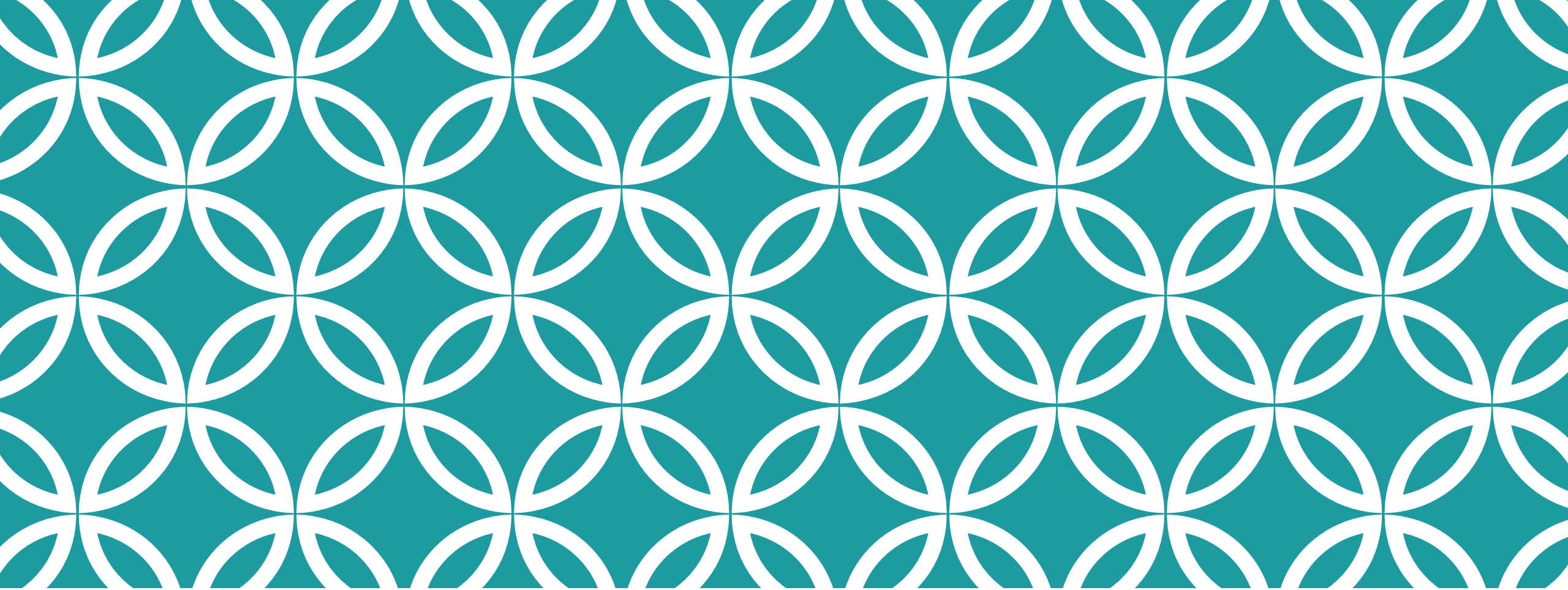
- 给定N个整数A_1到A_n，选出两个进行xor运算，问结果最大是多少。
  $N \leq 10^5$，$0 \leq A_i < 2^{31}$

TIRE

例题： The XOR Largest Pair

- 给定N个整数A_1到A_n，选出两个进行xor运算，问结果最大是多少。$N \leq 10^5$，$0 \leq A_i < 2^{31}$

- 我们可以将整数转为二进制，看成01串，然后就可以用一颗Trie存储所有的整数了
- 对于每个整数可以使用贪心策略求出和它异或起来最大的值：
  - 从最高位开始，如果某一位上存在一个与其相反的数就优先进入这个分支，若不存在则进入另一个分支
- 时间复杂度O(NlogAi)



其他算法

KMP、Manacher 、最小表示法

KMP

KMP模式匹配

- KMP算法能在线性时间内判定字符串A是否为字符串B的子串，同时求出具体的出现位置

*AC自动机

- AC自动机能实现多个字符串Ai对字符串B的匹配
- AC自动机=KMP + Trie

KMP

next数组

- 记next[i]表示A中以i结尾的非前缀子串与A所能匹配的最大长度，换句话说就是一旦匹配出现失败最少需要移动到哪里
- 已知next数组的情况下进行匹配，如果a[i]=b[j]，那么只要将二者右移一格；如果a[i]!=b[j]，那么需要将i调整为next[i]
- 每次操作i减少和j增加1这两个事件至多发生1次，而i总共只能增加n，因此总共只能减少n次，总的匹配次数不超过2n次
- 时间复杂度O(n)

KMP

KMP过程

```
for(int i = 1, j = 0; i <=m; ++i) {
        while(j>0 && (j==n || b[i] != a[j+1])) j = next[j];
        if (b[i]==a[j+1]) ++j;
        f[i] = j;  if (f[i] == n) { 匹配成功}
}
```

KMP

求Next数组过程

- 求next数组相当于对自己进行KMP

```
next[1] = 0;
for(int i = 2, j = 0; i <=n; ++i) {
        while(j>0 && (j==n || a[i] != a[j+1])) j = next[j];
        if (a[i]==a[j+1]) ++j;
        next[i] = j;
}
```

KMP

例题：Period（poj1961）

- 一个字符串S等于另一个串T重复K次，现已知长度n的串S，求最小循环节。数据范围：$n \leq 18$，$W \leq 10^9$

KMP

例题：Period（poj1961）

- 一个字符串S等于另一个串T重复K次，现已知长度n的串S，求最小循环节。数据范围：n≤18，W≤10^9

- 我们可以随机几组样例找规律
- 如果字符串出现了周期为K的循环，那么i-next[i]应该是周期K的约数，因为i-next[i]表示匹配失败时的最小位移
- 进一步可以发现：n-next[n]就是循环的最小正周期，即所求答案

MANACHER

Manacher算法

- Manacher算法能在线性时间求出字符串S中以每一个字符为中心的最大回文长度R[a]
- 为了方便我们在原串中每两个字符中间插入一个标志字符，这些所有的回文串都变成了奇数长度
- manacher算法的核心思想
- 若以一个字符a为中心的回文串被包含在另一个更大的回文串（以b为中心）当中，那么R[a]应该等于a关于b对称点的值R[2*b-a]
- 如果a超过了b的半径覆盖范围，那么a成为了新的对称中心，即令b等于a

MANACHER

Manacher算法

```
inline void manacher() {
    int maxright=0,mid;
    for(int i=1; i<n; i++) {
        if(i<maxright)
            hw[i]=min(hw[(mid<<1)-i],hw[mid]+mid-i);
        else
            hw[i]=1;
        while(s[i+hw[i]]==s[i-hw[i]])
            ++hw[i];
        if(hw[i]+i>maxright) {
            maxright=hw[i]+i;
            mid=i;
        }
    }
}
```

MANACHER

例题：对称正方形（ZJOI2008）

- 已知一个n*n的矩阵，求其中有几个上下左右都对称的正方形区域。数据范围：n≤1000

MANACHER

例题：对称正方形（ZJOI2008）

- 已知一个n*n的矩阵，求其中有几个上下左右都对称的正方形区域。数据范围：n≤1000

- 二阶回文问题，可以分别求出以每个点为中心水平方向以及数值方向的最大回文串长度
- 然后用ST表维护处水平方向的区间最小值以及竖直方向的区间最小值
- 最后以每个点为中心二分最大半径

- （~~当然这道题也能用字符串hash卡过~~）

最小表示法

最小表示法

- 如果字符串是环形结构的，为了方便，我们用字典顺序最小的一个字符串来表示这个环
- 直接爆了显然是$O(n^2)$的
- 事实上存在$O(n)$算法

最小表示法

算法步骤

- 1. 令i=1 j=2
- 2. 比较B[i]和B[j]
  - (1) 若相等，则字符串s全是相同字符，直接输出
  - (2) 若在i+k和j+k出发现不等：若s[i+k]>s[j+k]，令i=i+k+1，若此时i=j，则i++；若s[i+k]<s[j+k]，令j=j+k+1，若此时i=j，则j++；
- 若i>n，则输出B[j]；若j>n，则输出B[i]；否则重复2过程

- 显然每次扫描i+j单调递增，因此时间复杂度为O(n)