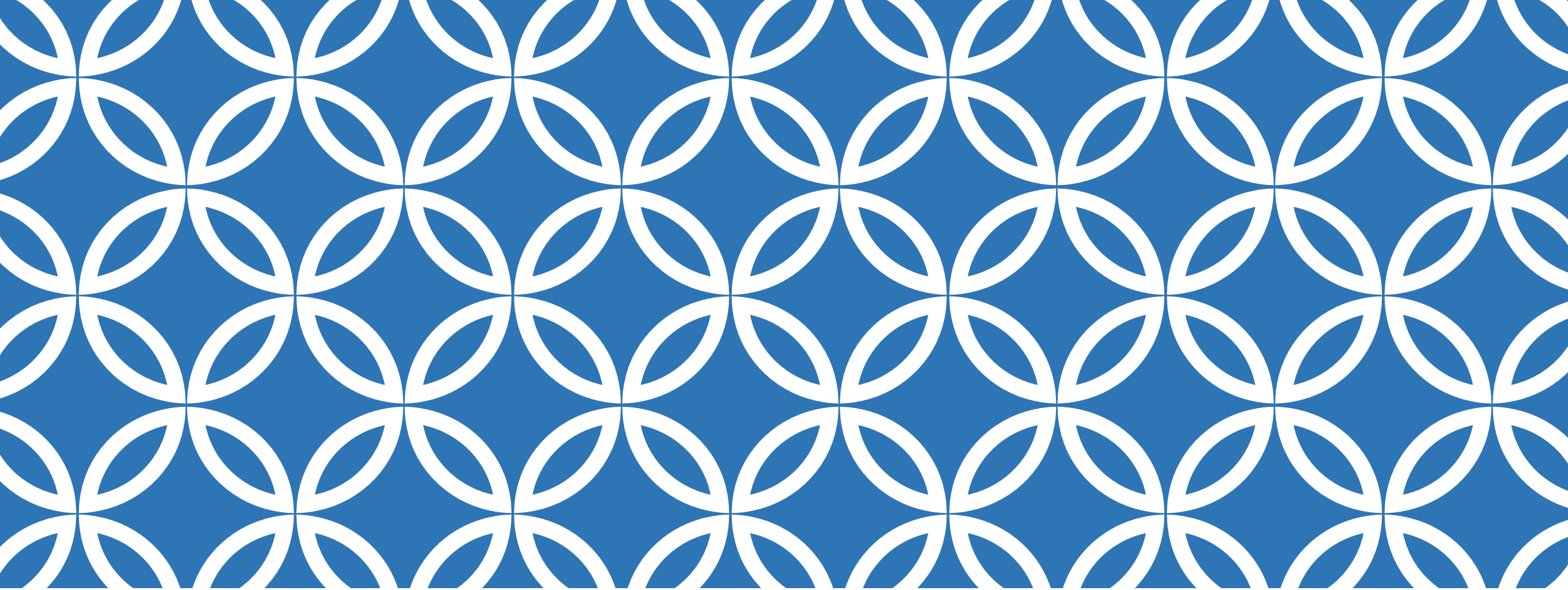


DP入门

洛谷网校
ruanxingzhi

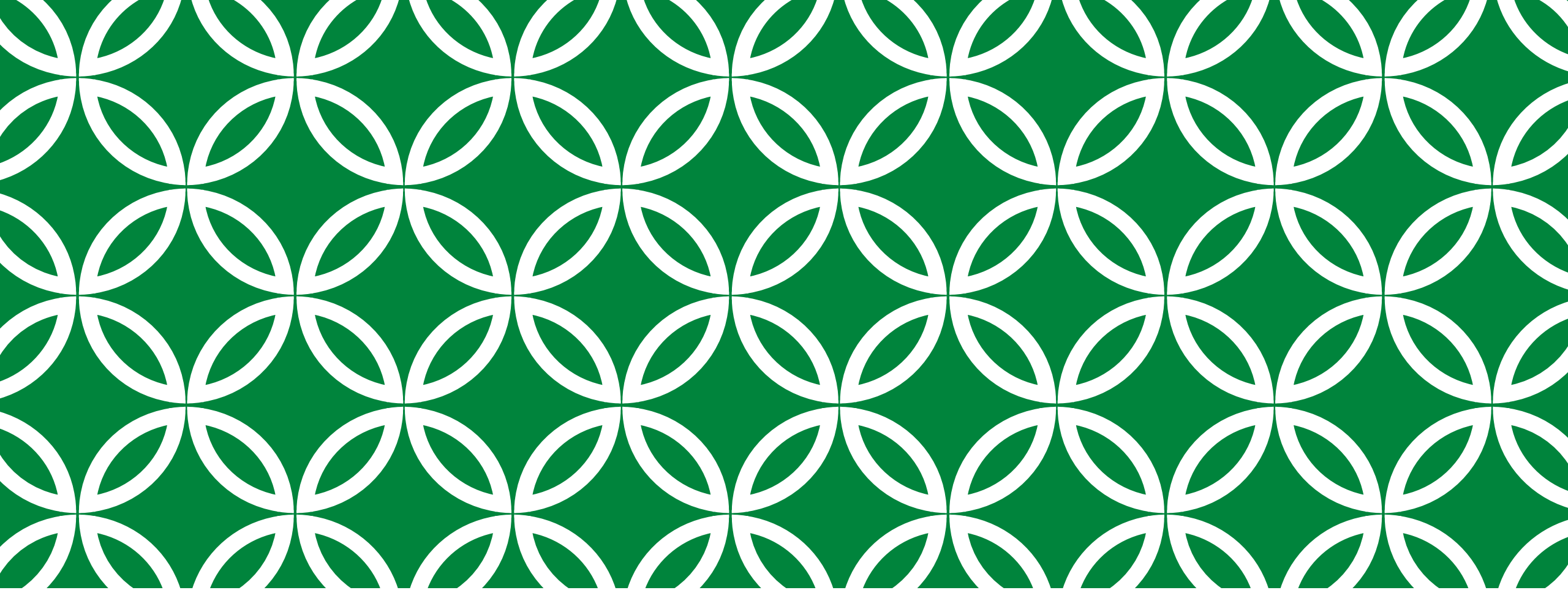
INTRO

这篇课件用于DP入门。

前置技能：小学数学知识

约定几个英文缩写：

e.g.	例如	动物都是生物，e.g. 猫是生物。
etc.	等等	动物中有猫，狗，etc.
P.S.	备注	P.S. 这篇课件是rxz做的。



DP入门

硬币问题

您有无限多的硬币，硬币的面值为1,5,10,20,50,100。

给定一个数额w，问您最少用多少枚硬币可以凑出w.

硬币问题

容易看出，这些硬币的面值和人民币一致。

依据生活经验，我们可以采用这种策略：
先尽量用100的，然后尽量用50的……以此类推。

e.g. 666 = 6*100+1*50+1*10+1*5+1*1,共用10枚硬币。

贪心

这就是贪心了。

我们每次使用一个硬币，总能最大程度地解决问题（把剩下要凑的数额变小）。

贪心是一种只考虑眼前情况的策略。

尽管这一套硬币面值可以采用贪心策略，但是迟早要栽跟头的。

贪心

我们考虑一组新的硬币面值：1,5,11.

于是有了一个反例：如果我们要凑出15，贪心策略是：

15 = 11+4*1,共用5枚硬币。

而最佳策略是：

15 = 3*5,共用3枚硬币。

怎么办办

贪心策略自此陷入困境：鼠目寸光。

在w=15时，贪心策略选择了面值11的硬币（因为这样可以尽可能降低要凑的数额）。

在选择了面值为11的硬币之后，我们只好面对w=4的处境。

怎么办办

那该怎么避免贪心的“鼠目寸光”呢？

强行枚举使用的硬币？
复杂度太高。

观察一波性质？

性质

我们重新分析刚刚的情况：

w=15时，我们取了11，接下来面对w=4的情况。
w=15时，如果我们取5，接下来就面对w=10的情况。

我们记“凑出n需要用到的最少硬币数量”为$f(n)$.

性质

那么，如果我们取了11，则：

$\text{cost} = f(4) + 1 = 4 + 1 = 5.$

解释：我们用了一枚面值为11的硬币，所以加一；

接下来面对的是w=4的情况。$f(4)$我告诉你等于4.

相应地，如果我们选择取5，则：

$\text{cost} = f(10) + 1 = 2 + 1 = 3.$

性质

那么，w=15时，我们选哪枚硬币呢？

cost最低的那一个！

11： $\text{cost} = f(4) + 1 = 4 + 1 = 5.$

5： $\text{cost} = f(10) + 1 = 2 + 1 = 3.$

1： $\text{cost} = f(14) + 1 = 4 + 1 = 5.$

选择5，$f(15) = 3,$即为答案！

性质

我们注意到了一个很棒的性质：

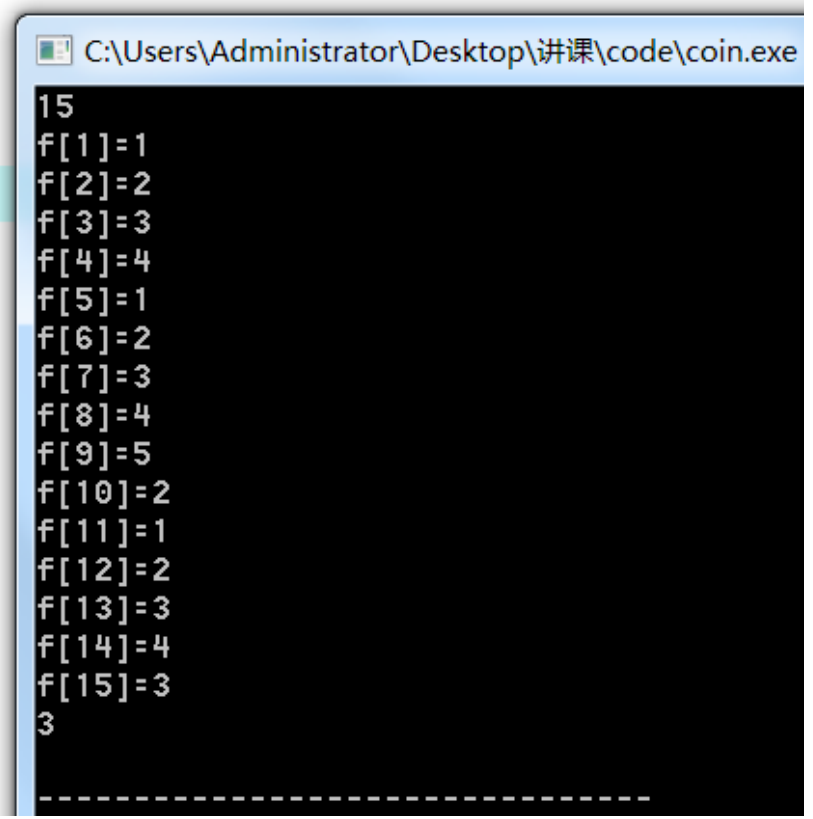
$f(n)$只与$f(n-1), f(n-5), f(n-11)$相关。

更确切地说：

$$f(n) = \min\{f(n-1), f(n-5), f(n-11)\} + 1$$

代码

```
int f[105],i,n,cost;
scanf("%d",&n);

f[0]=0;

for(i=1;i<=n;i++)
{
    cost=INF;
    if(i-1>=0) cost=min(cost,f[i-1]+1);
    if(i-5>=0) cost=min(cost,f[i-5]+1);
    if(i-11>=0) cost=min(cost,f[i-11]+1);
    f[i]=cost;
    printf("f[%d]=%d\n",i,f[i]);
}
```



原理

这个做法的原理是：

- $f(n)$只与$f(n-1), f(n-5), f(n-11)$相关。
- 我们只关心$f(w)$的值，不关心是怎么取到w的。

原理

这个做法和贪心的区别是：

这个算法对给定的w，会算出取1、5、11的代价，从而确定最终答案。

而贪心直接选择可选的最大硬币，一条路走到黑。

原理

这个算法的时间复杂度显然是$O(n)$.为什么比暴力要快呢？

我们的暴力枚举了“使用的硬币”，然而这属于冗余信息。
我们要的是答案，根本不关心这个答案是怎么凑出来的。

要求出$f(15)$，只需要知道$f(14), f(10), f(4)$的值。
其他信息不需要。

原理

可见，我们的做法比暴力快，是因为我们舍弃了冗余信息。
我们只记录了对解决问题有帮助的信息—$f(n)$.

我们能这样干，取决于问题的性质：
求出$f(n)$，只需要知道几个更小的$f(c)$.

我们将求解$f(c)$称作求解$f(n)$的“子问题”。

DP

这就是DP（动态规划，dynamic programming）.

将一个问题拆成几个子问题，分别求解这些子问题，即可推断出大问题的解。

无后效性

一旦$f(n)$确定，“我们如何凑出$f(n)$”就再也用不着了。

要求出$f(15)$，只需要知道$f(14), f(10), f(4)$的值，而$f(14), f(10), f(4)$是如何算出来的，对之后的问题没有影响。

“未来与过去无关”，这就是无后效性。

（其严格定义：如果给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响。）

最优子结构

回顾我们对$f(n)$的定义：
我们记“凑出n需要用到的最少硬币数量”为$f(n)$.

$f(n)$的定义就已经蕴含了“最优”。
利用w=14,10,4的最优解，我们即可算出w=15的最优解。

大问题的最优解可以由小问题的最优解推出，这个性质叫做“最优子结构性质”。

DP的条件

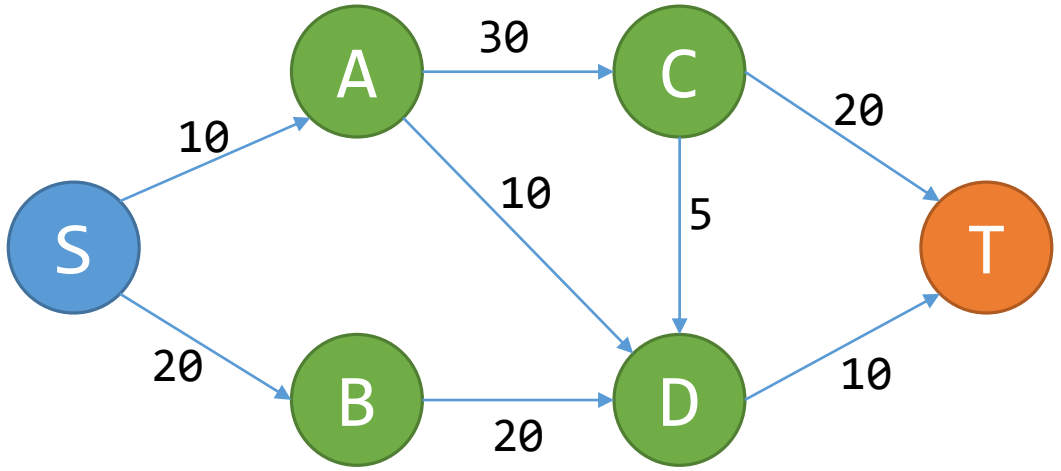
什么情况下我们能使用DP呢？

我们能将大问题拆成几个小问题，且满足

- 无后效性
- 最优子结构性质

DAG最短路

给定一个城市的地图，所有的道路都是单行道，而且不会构成环。每条道路都有过路费，问您从S点到T点花费的最少费用。



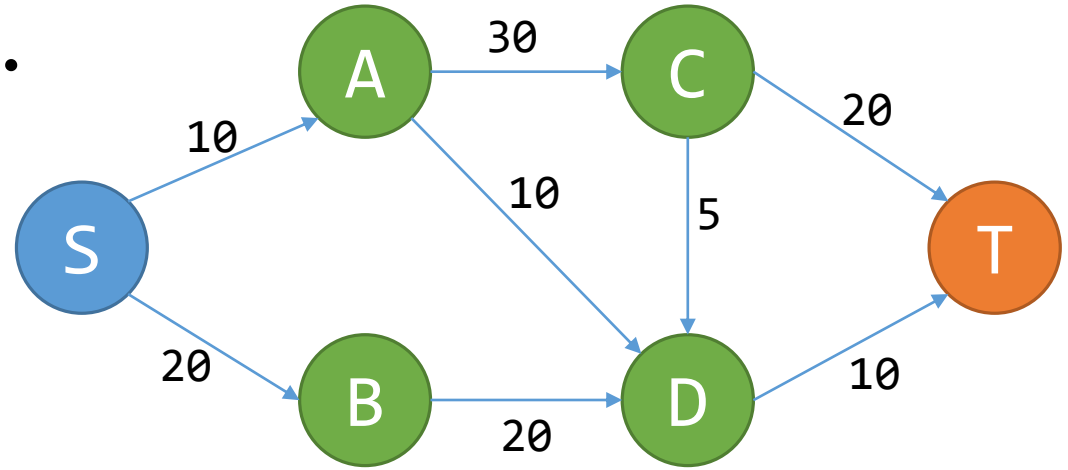
DP

能DP吗？

记从S到P的最少费用为$f(P)$.

想要到T，

要么经过C，要么经过D。



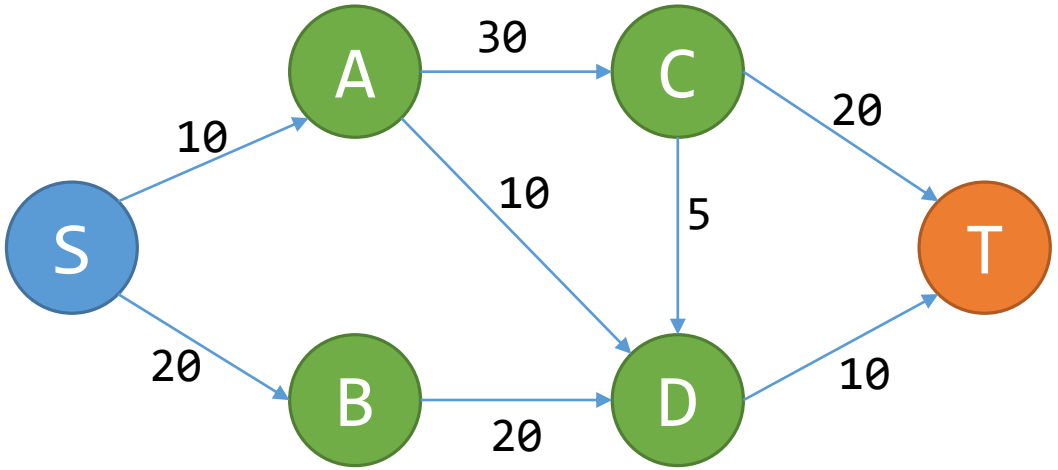
$f(T) = \min\{f(C) + 20, f(D) + 10\}$.好像看起来可以DP？

DP

来看看两个性质。

无后效性：

对于点P，一旦$f(P)$确定，以后就不关心怎么去的。



最优子结构：对于P，我们当然只关心到P的最小费用，即$f(P)$，并以此来解决之后的问题。

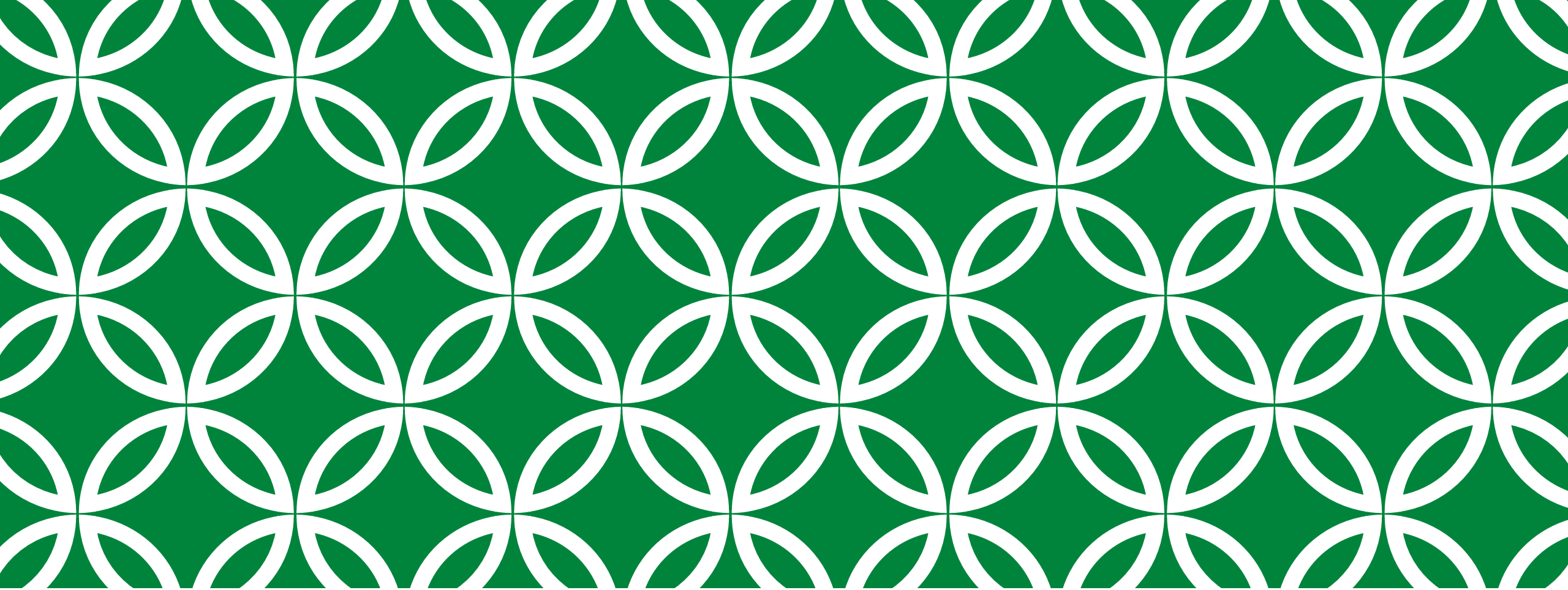
DP

两个性质都满足，可以DP！

$$f(P) = \min\{f(R) + w_{R\to P}\}$$

其中R为有路通到P的所有的点，$w_{R\to P}$为R到P的过路费。

代码怎么写先不管，理解这个算法的意思就行。



关于DP的本质

DP为什么会快

DP为什么会快？

无论是DP还是暴力，我们的算法都是在可能解空间内，寻找最优解。

来看硬币问题。

暴力做法是枚举所有的可能解，这是最大的可能解空间。

DP是枚举有希望成为答案的解。这个空间比暴力的小得多。

DP为什么会快

也就是说：DP自带剪枝。

DP舍弃了一大堆不可能成为最优解的答案。譬如硬币问题：

15 = 5+5+5 被考虑了。

15 = 5+5+1+1+1+1+1 从来没有考虑过，因为这不可能成为最优解。

DP的核心

从而我们可以得到DP的核心思想：

尽量缩小可能解空间。

在暴力算法中，可能解空间往往是指数级的大小；如果我们采用DP，那么有可能把解空间的大小降到多项式级。

一般来说，解空间越小，寻找解就越快。这样就完成了优化。

DP的操作过程

大事化小，小事化了。

将一个大问题转化成几个小问题；

求解小问题；

推出大问题的解。

设计DP算法

首先，把我们面对的局面表示为x.这一步称为设计状态。

对于状态x，记我们要求出的答案(e.g. 最小费用)为$f(x)$.

我们的目标是求出$f(T)$.

找出$f(x)$与哪些局面有关（记为p），写出一个式子（称为状态转移方程），通过$f(p)$来推出$f(x)$.

DP三连

设计DP算法，往往可以遵循DP三连：

我是谁？ ——设计状态，表示局面

我从哪里来？

我要到哪里去？ ——设计转移

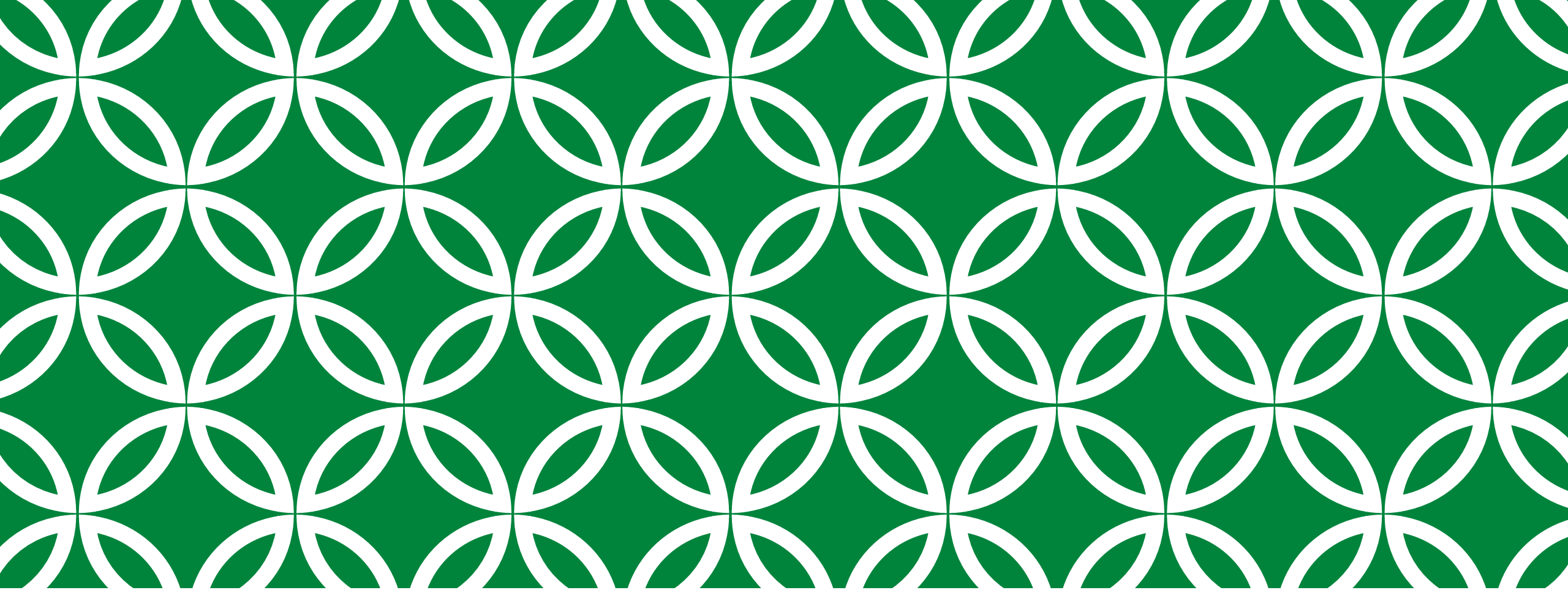
DP_EX

未来将会学习到DP的各种优化。

e.g. 数据结构优化、斜率优化。

一般而言，DP的难点，在初学时是如何设计状态；在学习深入一些之后，变成了如何设计转移；在省选/NOI级别，又变成了如何设计状态。

学习DP主要靠做题练习。有一些设计状态的思想，需要在具体题目中总结。



例题选讲（一）

数字三角形

http://poj.org/problem?id=1163

最大子段和

给出一个整数序列，问最大子段和。

e.g. [2,-1,3,-5,3]的最大子段和是2-1+3=4.

要求一个$O(n)$的算法。

最大子段和

如何设计状态？

dp[i]表示[1,i]位的最大子段和。

状态i从哪里来？

要么合并上之前的最大子段和(dp[i-1])，要么另起炉灶。

```
dp[i]=max(dp[i-1]+a[i],a[i])
```

最长上升子序列

最长上升子序列（LIS）问题：

给定长度为n的序列a，从a中抽取出一个子序列，这个子序列需要单调递增。问最长的上升子序列（LIS）的长度。

e.g. 1,5,3,4,6,9,7,8

LIS为1,3,4,6,7,8，长度为6。

最长上升子序列

如何设计状态？

我们记$f(x)$为以a_x结尾的LIS长度，那么答案就是$\max\{f(x)\}$.

状态x从哪里推过来？

考虑比x小的每一个p：如果$a_x > a_p$，那么$f(x)$可以取$f(p)+1$.

解释：我们把a_x接在a_p的后面，肯定能构造一个以a_x结尾的上升子序列，长度比以a_p结尾的LIS大1.

最长上升子序列

那么，我们可以写出状态转移方程了：

$$f(x) = \max_{p<x\,,\,a_p<a_x} \{f(p)\} + 1$$

两层for循环，复杂度$O(n^2)$.

最长上升子序列

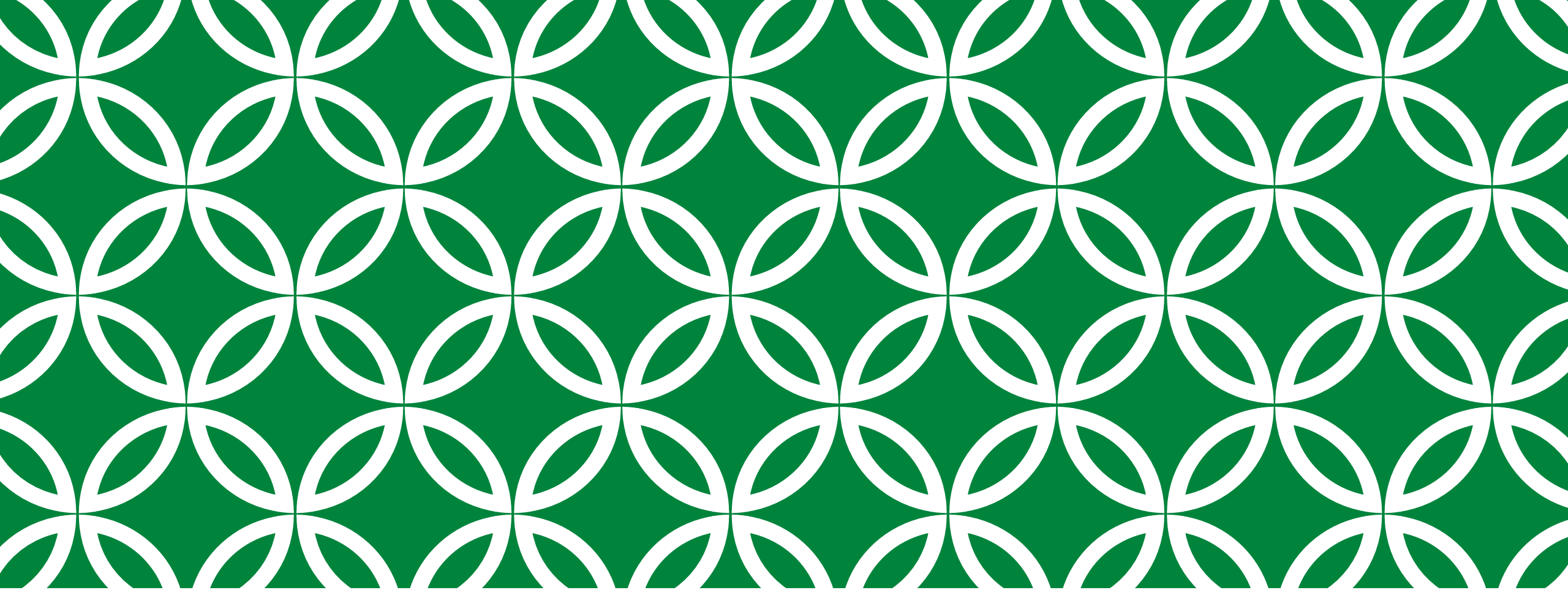
```
int main(void)
{
    int f[105]={0},a[105]={0},i,x,p,n,ans=0;
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        scanf("%d",&a[i]),f[i]=1;

    for(x=1;x<=n;x++)
    {
        for(p=1;p<x;p++)
            if(a[p]<a[x]) f[x]=max(f[x],f[p]+1);
        printf("f[%d]=%d\n",x,f[x]);
    }

    for(x=1;x<=n;x++)
        ans=max(ans,f[x]);

    printf("%d\n",ans);
```

优化

此问题可以优化到$O(n \log n)$.这里安利一份阅读材料:

《动态规划初步·各种子序列问题》
此文对初学者很有帮助。作者是Flower_pks.

https://pks-loving.blog.luogu.org/junior-dynamic-programming-dong-tai-gui-hua-chu-bu-ge-zhong-zi-xu-lie



记忆化搜索

斐波那契数列

我们来看一段求斐波那契数列的代码：

```
int fib(int n)
{
    if(n==1 || n==2) return 1;
    printf("Calc fib[%d]\n",n);
    return fib(n-1)+fib(n-2);
}
```

记忆化搜索

在刚刚的例子中，fib函数有很多不必要的调用。

例如，如果已经知道f(5)是5，那么以后查询f(5)的时候返回5即可，无需进一步递归。

我们用一个记忆数组bin来解决问题。

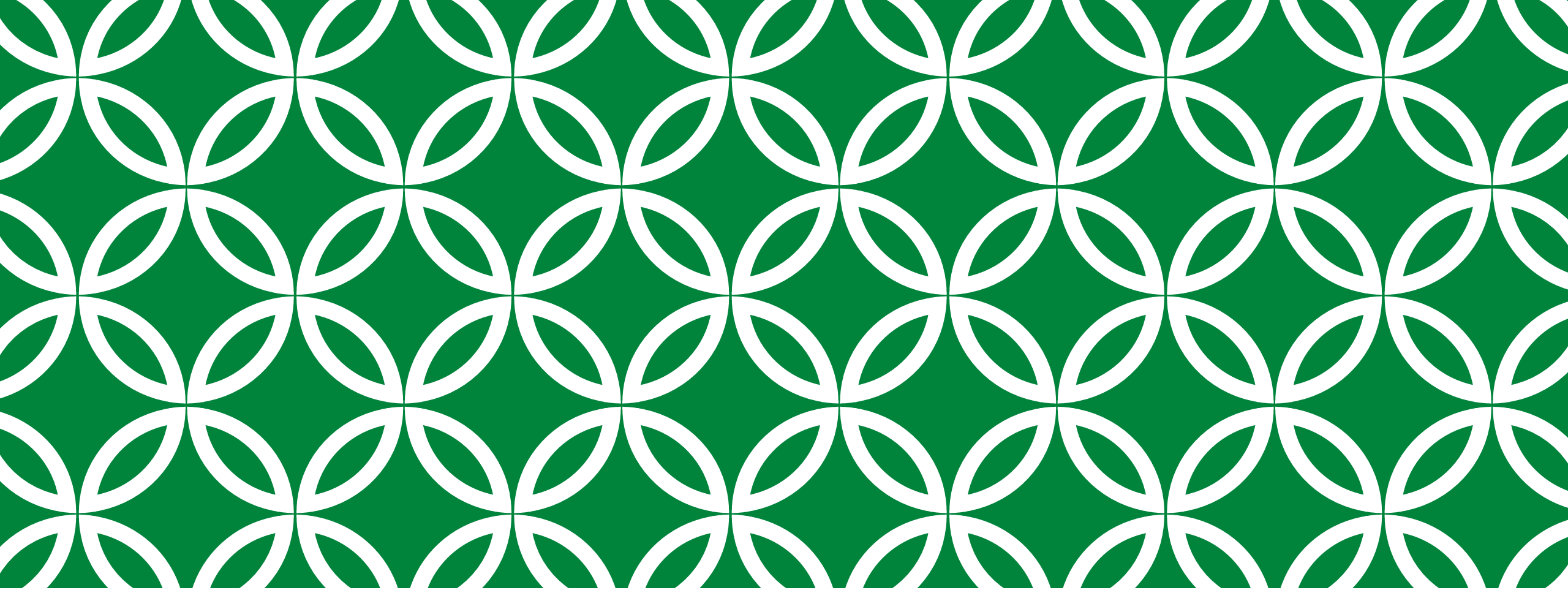
记忆化搜索

做了记忆化之后，每个斐波那契数只被计算一次。

```
int fib(int n)
{
    if(n==1 || n==2) return 1;
    if(bin[n]) return bin[n];
    printf("Calc fib[%d]\n",n);
    return bin[n]=fib(n-1)+fib(n-2);
}
```

例题

https://www.luogu.org/problemnew/show/P1464



计数类DP

数楼梯

楼梯有n阶，上楼可以一步上1阶，也可以一步上2阶。
编一个程序，计算共有多少种不同的走法。

$n \leq 5000$.

P.S. 来源：洛谷P1255

数楼梯

想题先想暴力。

暴力怎么做呢？
枚举每一步是上1阶还是上2阶，递归下去即可。（DFS）

哪些信息是冗余的？

数楼梯

记“走上x阶的方案数”为$f(x)$.答案就是$f(n)$.

我们是从哪里走上x阶的呢？
从$x-1, x-2$这两种情况。

因此有：$f(x)=f(x-1)+f(x-2)$.
$O(n)$美滋滋。

数楼梯

这个例子和以往的DP不同，以前一般都是要作出最优决策。

但是由于其代码实现和DP相似，也有状态转移方程，所以我们仍然把这个做法算作DP.

过河卒

棋盘上A点有一个过河卒，需要走到目标B点。

卒行走的规则：可以向下、或者向右。

同时在棋盘上 C 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。

棋盘用坐标表示，A点是$(0,0)$,B点是(n,m)，保证$n,m \leq 20$.

马的位置坐标(p,q)是给定的。

现在要求你计算出卒从A点能够到达B点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。

P.S. 来源：洛谷P1002

过河卒

我们记走到(x,y)的路径数为$f(x,y)$.答案即为$f(n,m)$.

不考虑那只马，如何设计状态转移方程？

卒只能向下或者向右！不能走回头路！ 

$$f(x,y)=f(x-1,y)+f(x,y-1)$$

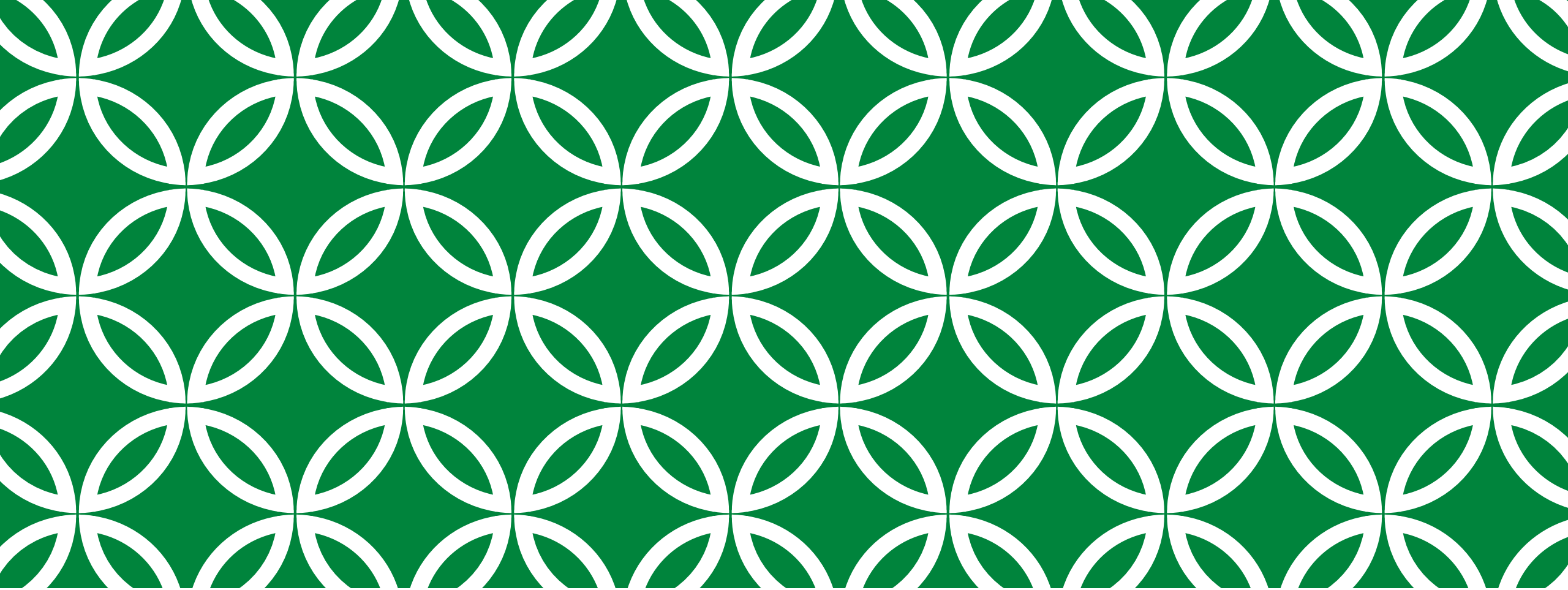
过河卒

现在考虑那只马，怎么办呢？

特判！钦定$f(mark)=0$.其中mark为被马控制的点和马本身的点。

最终的方程是：

$$f(x,y)=\begin{cases}0 & (x,y)\text{不能走}\\ f(x-1,y)+f(x,y-1) & \text{其余情况}\end{cases}$$



例题选讲（二）

装箱问题

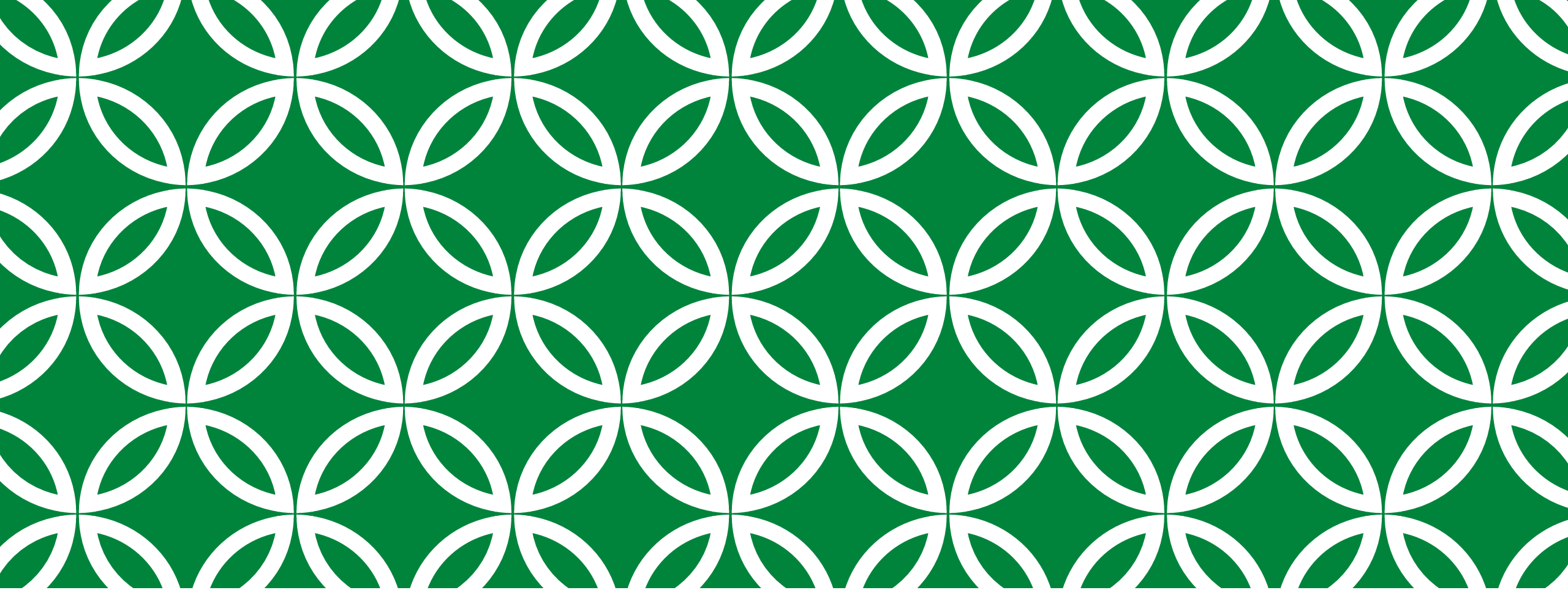
https://www.luogu.org/problemnew/show/P1049

乌龟棋

https://www.luogu.org/problemnew/show/P1541

采药

https://www.luogu.org/problemnew/show/P1048



有趣的东西

sth. excited

区间数颜色

给定一个长度为n的序列，每个元素有颜色。应对m个询问：

Ask(l,r)：询问$[l,r]$区间内有多少种颜色。

允许离线。

暴力A

暴力A：直接跑一遍寻问区间去统计。

单次询问复杂度$O(n)$，取决于数据，无法改变。

总复杂度$O(n^2)$

复杂度完全取决于数据，我们无法改进。

暴力B

暴力B：维护数组w，w[i]表示i这个颜色出现了多少次。

记cnt为当前处理的[l,r]这个区间的颜色数。

现在如果我们手上已经有了一个区间的信息，如何求出下一个区间的信息？

移动l,r指针，使之走到新的询问。每次移动指针的时候就更新信息。l,r走到之后，cnt即为答案。

暴力B

这个暴力算法的复杂度取决于什么?
取决于我们需要移动多少次指针。

我们需要移动多少次指针，和我们处理询问的顺序有关!
比如，三个询问[1,2],[10,10000],[5,6]
明显a->c->b这个处理顺序，移动指针的次数小于a->b->c.

莫队算法

国际上叫做MO's Algorithm.

提出者是2010年集训队莫涛（长郡中学）。

基本思路：改变这些询问的顺序，使之对我们有利。

P.S. 学术上的名称大概是“Query square root decomposition”.

莫队算法

考虑所有询问。

先分$\sqrt{n}$个桶，把询问按照l扔进$\frac{l}{\sqrt{n}}$这个桶。

然后，针对每个桶：将其中的询问按照r排序。

这套事情做完之后，直接跑暴力B。

复杂度：$O(m\sqrt{n})$.

复杂度分析

为什么复杂度就可以$O(m\sqrt{n})$?

考虑l指针：

每个桶内的元素，l相差不会超过$\sqrt{n}$，故l指针在每次询问的时候至多移动$\sqrt{n}$次。故l指针在整个程序中移动$O(m\sqrt{n})$次。

考虑r指针：

每个桶内，r都是有序的，所以在每个桶内r最多移动n次。

一共有$\sqrt{n}$个桶，故r移动$O(n\sqrt{n})$次。

代码实现

写两个函数：add,del用于跳指针。

排序：不需要显式地执行分块操作，只需要在排序的时候，以l所在的块作为第一关键字，以r作为第二关键字。

其它题目

区间询问出现次数多于3的颜色的个数。

区间询问出现次数多于k的颜色的个数。

区间询问众数。

[AHOI2013]作业

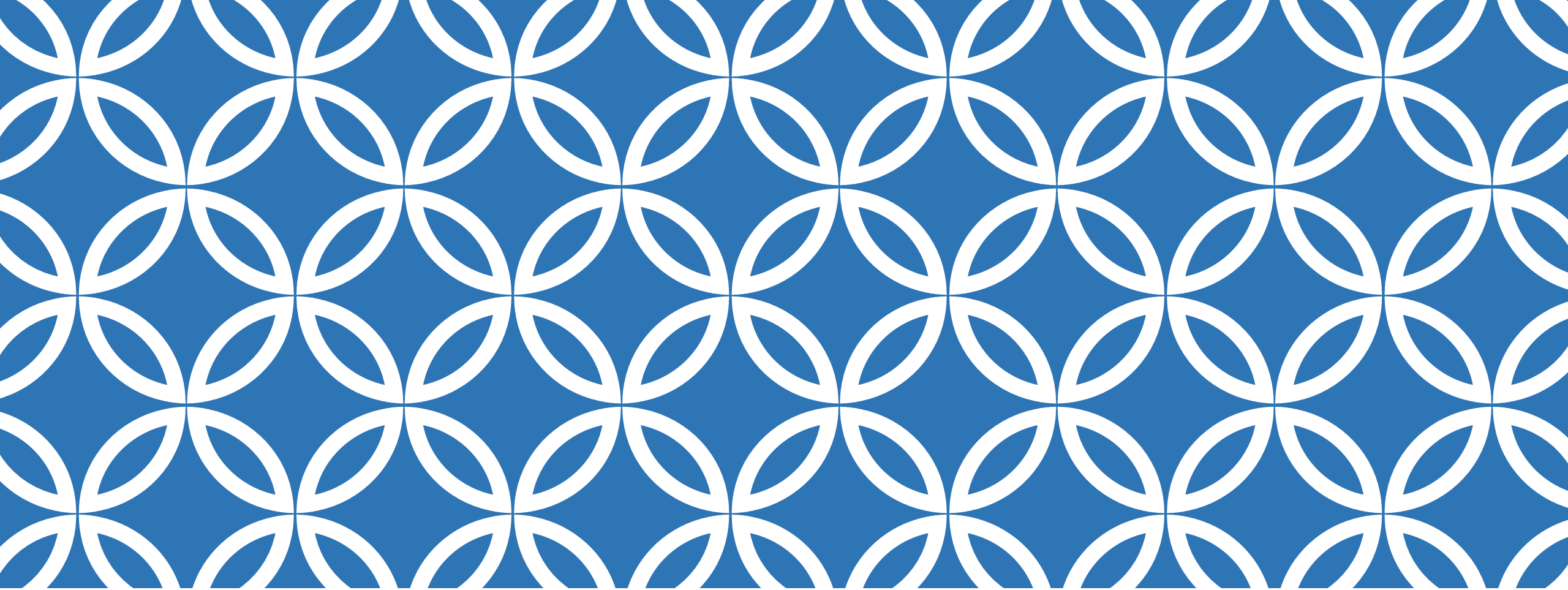
https://www.luogu.org/problemnew/show/P4396

莫队算法的适用范围

允许离线；

可以写出复杂度较好的add和del函数；

没有修改操作。



END

rxz@luogu.org