```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
	char c[20] = "1234";
	int x = atoi(c);
	cout << x;
	/*int x = 5678;
	itoa(x, c, 10);
	cout << c;*/
	return 0;
}
```

$$2147483647^{2147483647} \% 10007$$

一个小问题

- 计算 $a^b \% c$（a、b ≤ 15）

sample

Input
 	
`15 15 10007`
 	
Output
 	
`6525`

幂运算

- 不得不说这个复杂度为 O(b) 的朴素算法，实在太容易爆了，开 long long 也没什么用
- 怎么破?

```
long long ans = 1;
for (int i = 1; i <= b; ++i)
    ans *= a;
printf("%lld", ans % c);
```

特别提醒：pow（a,b）这个函数精度比较差，除非结果很小否则不建议用

快速幂

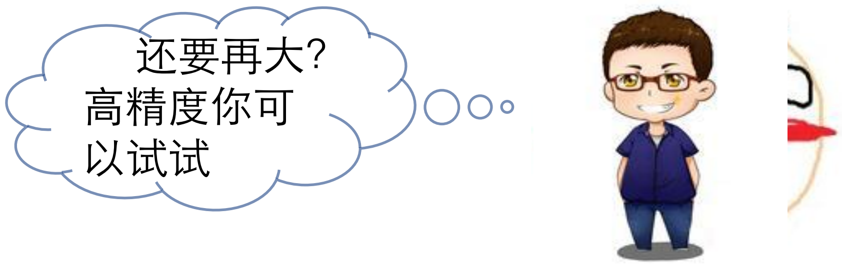
- 先来看一个数学公式：$a^b \% c = (a \% c)^b \% c$
- 于是我们惊喜的发现，a 输入到 2147483647 也没爆，当然 b 还是不能太大

```
long long ans = 1;
a = a % c;
for (int i = 1; i <= b; ++i)
    ans = (ans * a) % c;
printf("%lld", ans % c);
```

快速幂

- 继续来看一个数学公式：$a^b \% c = (a^2)^{b/2} \% c$
- 因为整除的关系，所以当 b 是奇数的时候，$a^b \% c = ((a^2)^{b/2} \times a) \% c$
- 如果我们用 a^2 迭代 a，这样 b 就可以缩减为 b / 2，循环次数减少一半
- 如此往复下去，直到 b == 0 时结束
- 但这样 a 会越来越大？这时候就需要第一个数学公式来配合了

快速幂



- 我们终于可以放心地计算 $2147483647^{2147483647} \% 10007$ 这个原本会炸裂的算式了
- 复杂度？ $O(\log_2 b)$

```
long long ans = 1;
a = a % c;
while (b > 0)
{
    if (b % 2) ans = (ans * a) % c;
    a = (a * a) % c;
    b /= 2;
}
printf("%lld", ans % c);
```

快速幂

- 也可以用位运算，还能再快一点儿

```
long long ans = 1;
a = a % c;
while (b > 0)
{
    if (b & 1) ans = (ans * a) % c;
    a = (a * a) % c;
    b >>= 1;
}
printf("%lld", ans % c);
```

矩阵乘法

- 矩阵A为 m 行 n 列，矩阵B为 n 行 p 列，则矩阵A可以和B相乘得到一个 m 行 p 列的矩阵C

$$\begin{pmatrix} 2 & 3 & 4 \\ 5 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 4 & 2 \\ 2 & 0 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 2\times4+3\times2+4\times3 & 2\times2+3\times0+4\times1 \\ 5\times4+0\times2+1\times3 & 5\times2+0\times0+1\times1 \end{pmatrix}$$

矩阵乘法改写递推式

- 利用矩阵乘法的独特性质，可以通过构造[0 1]型的单位矩阵，把线性递推式改写

- Fibonacci数列问题就是一个典型的递推式问题：Fn=Fn-2+Fn-1

$$\begin{pmatrix} F_{n-2} & F_{n-1} \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} F_{n-1} & F_{n} \end{pmatrix}$$

矩阵乘法快速幂优化递推

- 所以， Fibonacci数列问题可以改造成一个基于矩阵乘法的快速幂问题（矩阵乘法满足结合律）

参考代码

```
matrix pow(matrix a, int n)
{
    matrix ans;
    while (n > 0)
    {
        if (n & 1) ans = mul(ans, a);
        //mul(matrix a,matrix b)为矩阵乘法函数
        a = mul(a, a);
        n >>= 1;
    }
    return ans;
}
```

最大连续子段和

- 求长度为 n （$n \le 10^6$）的数列中，最大的连续子段和

sample

Input
 	
`10`
`3 -8 20 12 5 -17 41 -140 25 30`
 	
Output
 	
`61`

最大子段和

- 显然不能单纯以负数的出现作为评判标准
- 如果一个新数加入到现有的区间和中，使得和为正，那它一定是当前考查目标区间的一部分；反之，则要把该数连同之前的和都舍弃
  – 因为与其保留一个负数，不如从零开始更优
- 怎么理解？就是区间和无论是增长还是减少，但只要还为正，就都是对最终答案有贡献的，应该保留（明显这里有负数也可以，只要不是负得太多）

a[ ]	3	-8	20	12	5	-17	41	-140	25	30
sum[ ]	3	0	20	32	37	20	61	0	25	55

这里虽然和减少，但依然有
可能对最终的答案有贡献

最大子段和

这段代码很简洁，
运行也很高效，然而
有一个隐蔽的bug：
试着输入-1、-2、-3
看看

```
for(int i = 1; i <= n; ++i)
{
    scanf("%d", &a[i]);
    sum[i] = max(0, sum[i-1] + a[i]);
    if (sum[i] > ans) ans = sum[i];
}
printf("%d", ans);
```

最大子段和

- 修改后的代码，供参考

```
int ans = sum[1];
for(int i = 2; i <= n; ++ i)
{
    scanf("%d", &a[i]);
    sum[i] = max(a[i], sum[i-1] + a[i]);
    if (sum[i] > ans) ans = sum[i];
}
printf("%d", ans);
```

最大子矩阵和

- 类似的，最大子段和也可以推广到二维

最大子矩阵和

```
for (int i = 1; i <= n; ++ i)
{
    for (int j = i; j <= n; ++ j)
    {
        t[1] = sum[j][1] - sum[i-1][1];  //初始化第一列
        for (int k = 2; k <= n; ++ k)
        {
            if (t[k-1] > 0) t[k] = t[k-1] + sum[j][k] - sum[i-1][k];
                else t[k] = sum[j][k] - sum[i-1][k];
            if (t[k] > ans) ans = t[k];
        }
    }
 }
```

课堂练习

- 长度为 n （$n \le 10^6$）的数列中，对某指定区间 [L, R] 中的每个数都加上值 x，问这个数列最后长啥样。最多有10000个区间（这些区间可能有重叠的部分）

sample

Input

```
10
2 1 8 9 4 15 7 11 -3 6
3 7 2
6 10 -3
8 9 5
1 3 1
2 4 1
```

Output

```
3 3 12 12 6 14 6 13 -1 3
```

差分序列

- 朴素的算法依然是$O(10^6\times10^4)$的
- 区间可能有重叠，看起来没有更好的办法？
- 我们建立一个 b 数组，b 是原数组 a 中每项与前项的差值，看起来 b 是 a 的“逆前缀和数组”，或者说 a 是 b 的前缀和数组。

a[ ]	2	1	8	9	4	15	7	11	-3	6
b[ ]	2	-1	7	1	-5	11	-8	4	-14	9

差分序列

这样对于每个区间我们只要修改两个值b[L]、b[R+1]

- 然后我们对 b 数组操作：b[L]=+x；b[R+1]=-x；
- 再对 b 数组求前缀和，还原成新的 a 数组

a[ ]	2	1	8	9	4	15	7	11	-3	6
			L=3		x=2		R=7			
b[ ]	2	-1	9	1	-5	11	-8	2	-14	9

a[ ]	2	1	10	11	6	17	9	11	-3	6
			L=3		x=2		R=7			

差分序列

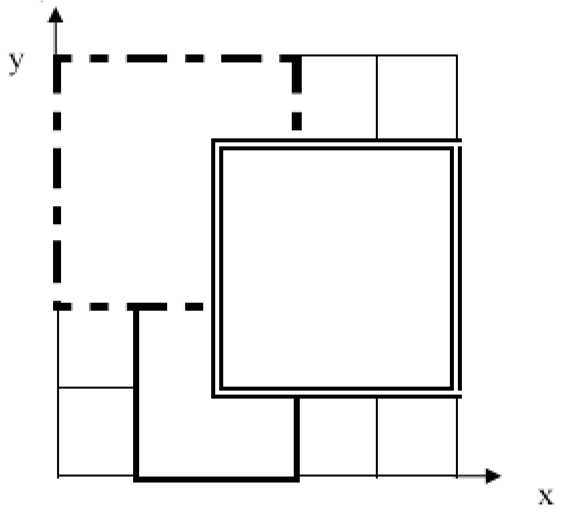
复杂度从朴素算法的$O(10^6 \times 10^4)$，降到$O(10^6+10^4)$

```c
for (int i = 1; i <= n; ++i)
    scanf("%d", &a[i]), b[i] = a[i] - a[i-1];

while (scanf("%d%d%d", &L, &R, &x) == 3)
    b[L] += x, b[R+1] -= x;

for (int i = 1; i <= n; ++i)
    a[i] = a[i-1] + b[i], printf("%d ", a[i]);
```

铺地毯

- 给n×m的矩形区域铺上k块地毯，每块地毯给出左下角、右上角坐标
- 问所有k块地毯铺完之后，还有多少个整点没有被覆盖

- 暴力的复杂度O(nmk)



分析

- 每块地毯所能覆盖的整点都是连续的。
- 利用前缀和+差分序列可以在O(1)时间内实现对某一行一段区间内的修改。
- 比如：0 1 1 1 1 0
- 前缀和的复杂度是O(nm)，整体复杂度O(nk)+O(nm)