1004 I love counting

Mr W likes interval counting.

One day,Mr W constructed a sequence of length n, each position of this sequence has a weight c ($c \leq n$).

There are a total of Q queries, and each query is given an interval (l, r) and two parameters a, b, and ask how many $kinds\ of\ weights$ of this interval satisfy $c \bigoplus a \leq b$ where $\bigoplus$ is the binary Bitwise XOR operation.

In the first line contains a positive integer n ($n \leq 100000$) represents the length of the sequence.

In the second line contains n positive integers, The i-th number in the sequence represents the weight c_i ($1 \leq c_i \leq n$)of the i-th position.

In the third line, a positive integer Q ($Q \leq 100000$) represents the number of queries.

In the next Q line, each line has four positive integers l, r, a, b ($1 \leq l \leq r \leq n, a \leq n+1, b \leq n+1$), which represent the parameters of the query.

n棵字典树？莫队+字典树？可持久化字典树？树套树？

- n棵字典树:开log棵字典树，分别维护2^i到2^i+1个数，bitse优化
- 可持久化字典树：同上，只不过套模板来的方便
- 莫队+字典树：在字典树上维护当前询问区间的数字集合，对于询问的B,C在字典树上跑，统计可行的叶子节点个数
- 树套树：我不知道

莫队+分块！

- 字典树只适用于小规模数据，当数据多，数据大时需要动态开点删点，与此相比，分块占用空间小
- 通过调节块的大小，分块的常数可以非常灵活的调整(大雾)，满足你的多种暴力姿势，更有莫队的块大小可以调整，双重调参，双重惊喜！

- 或者您也可以使用树套树，解决字典树做法和分块做法的缺陷

迈向分块带师之路

找到[L,R]内所有的C，使得C XOR A<=B

- 莫队维护分块，分块维护对于当前询问[L,R]，a[i](L<=i,i<=R)的出现种数
- 附加维护个出现次数数组
- a[i]<1e6 开1~1e6的分块即可
- 插值O(1) 更新出现次数和当前块信息即可
- 查询O($\sqrt{n}$)

迈向分块带师之路

找到[L,R]内所有的C，使得C XOR A<=B

- 对于位2^j，假设在比他高的所有位上，我们都已经使得C XOR A的结果与B相同
- 那么我们只要使得B的第J位与C对应的位相同，即可使得C XOR A<=B

- 询问总复杂度 $q\sum_{d=0}^{\lfloor \log_2 n \rfloor}\frac{2^d}{block}$ <=O($\frac{nq}{block}$)

```
  int now = 0;
  for (int j = 22; j >= 0; j--) {
    if (q[i].b >> j & 1) {
      int l = now;
      if (q[i].a >> j & 1) {
        l += 1 << j;
      }
      ans[q[i].id] += ask(l, l + (1 << j) - 1);
    }
    if ((q[i].a ^ q[i].b) >> j & 1) {
      now |= 1 << j;
    }
  }
  ans[q[i].id] += b[q[i].a ^ q[i].b];
}
```

1011. I love max and multiply

```
#include<bits/stdc++.h>
using namespace std;

int t;
int n;
long long a[1048576],zad[1048576],zax[1048576],fad[1048576],fax[1
048576];
long long b[1048576],zbd[1048576],zbx[1048576],fbd[1048576],fbx[1
048576];
long long c[1048576];
long long mod=998244353;
long long const inf=1e20;

int main(){
    cin>>t;
    while(t--){
        cin>>n;
        //for(int i=0;i<n;++i) cout<<i<<" ";
        //cout<<"\n";
        //cout<<inf<<"\n";
        for(int i=0;i<n;++i) scanf("%lld",&a[i]);
        for(int i=0;i<n;++i) scanf("%lld",&b[i]);
        for(int i=0;i<n;++i){
            c[i]=-inf;
            if(a[i]>=0){
                zad[i]=a[i];zax[i]=a[i];fad[i]=1;fax[i]=-inf;
            }
            if(a[i]<0){
                zad[i]=-1;zax[i]=inf;fad[i]=a[i];fax[i]=a[i];
            }
        }

        for(int i=0;i<n;++i){
            if(b[i]>=0){
                zbd[i]=b[i];zbx[i]=b[i];fbd[i]=1;fbx[i]=-inf;
            }
            if(b[i]<0){
                zbd[i]=-1;zbx[i]=inf;fbd[i]=b[i];fbx[i]=b[i];
            }
        }

        int z=n-1,k=0;
        while(z){
            z/=2;
```

```
            k++;
        }
        //for(int i=0;i<n;++i) cout<<zad[i]<<" "<<zax[i]<<" "<<fa
d[i]<<" "<<fax[i]<<"\n";
        for(int i=n-1;i>=0;--i){
            for(int j=0;j<k;++j){
                if((i|(1<<j))>=n) break;
                zad[i]=max(zad[i],zad[i|(1<<j)]);
                zax[i]=min(zax[i],zax[i|(1<<j)]);
                fad[i]=min(fad[i],fad[i|(1<<j)]);
                fax[i]=max(fax[i],fax[i|(1<<j)]);
                zbd[i]=max(zbd[i],zbd[i|(1<<j)]);
                zbx[i]=min(zbx[i],zbx[i|(1<<j)]);
                fbd[i]=min(fbd[i],fbd[i|(1<<j)]);
                fbx[i]=max(fbx[i],fbx[i|(1<<j)]);
            }
        }
        //for(int i=0;i<n;++i) cout<<zad[i]<<" "<<zax[i]<<" "<<fa
d[i]<<" "<<fax[i]<<"\n";
        //for(int i=0;i<n;++i) cout<<zbd[i]<<" "<<zbx[i]<<" "<<fb
d[i]<<" "<<fbx[i]<<"\n";
        for(int i=0;i<n;++i){
            if(zad[i]!=-1&&zbd[i]!=-
1)         c[i]=max(c[i],zad[i]*zbd[i]);
            if(fad[i]!=1&&fbd[i]!=1)         c[i]=max(c[i],fad[i]
*fbd[i]);
            if(zax[i]!=inf&&fbx[i]!=-
inf)     c[i]=max(c[i],zax[i]*fbx[i]);
            if(fax[i]!=-
inf&&zbx[i]!=inf)     c[i]=max(c[i],fax[i]*zbx[i]);
        }
        //for(int i=0;i<n;++i) cout<<c[i]<<" ";
        //cout<<"\n";
        for(int i=n-2;i>=0;--i) c[i]=max(c[i],c[i+1]);
        long long ans=0;
        for(int i=0;i<n;++i) ans+=c[i]%mod+mod,ans%=mod;
        printf("%lld\n",ans);
    }

    return 0;
}
```

个人赛D题解题报告

演讲人

郁青陟

个人赛D题解题报告

【题目大意】

给定n，H，将n袋沙包摆成t堆，每堆高度为h[i]，使其满足

1.h[1] ≤ H;

2. 对于任意i∈[1,t]满足|h[i] - h[i+1]| ≤ 1;

3.所有h[i]之和为n。

问满足要求的m最小为多少。

数据范围：1 ≤ n, H ≤ 10^18

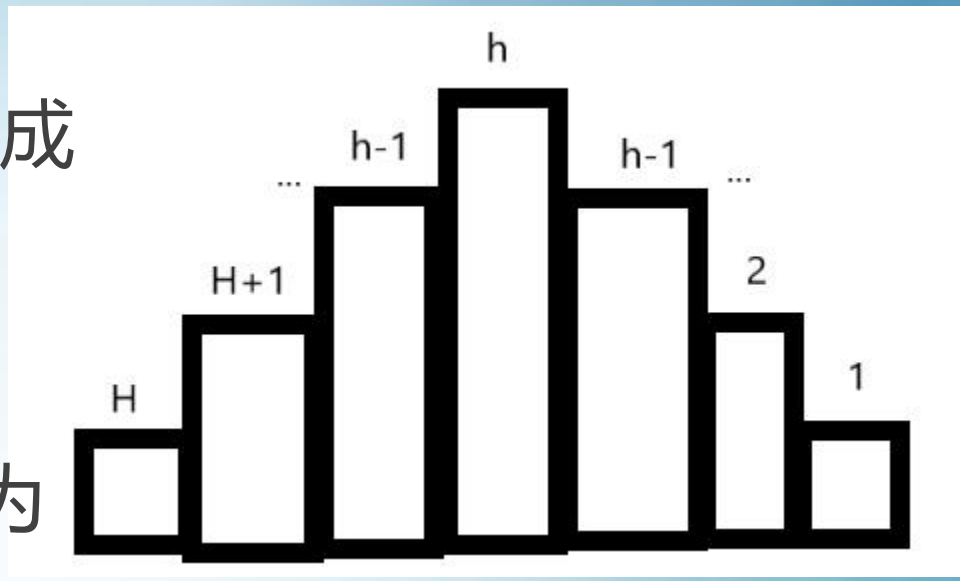
注：因为|h[t]-0|≤ 1,所以h[t]=1

【思路一】

首先分析得高度先递增再递减时能将沙包分成最少的堆数。

简化问题，假设h[i]先严格增再严格减

那么当最高高度为h时，所需要的沙包数即为

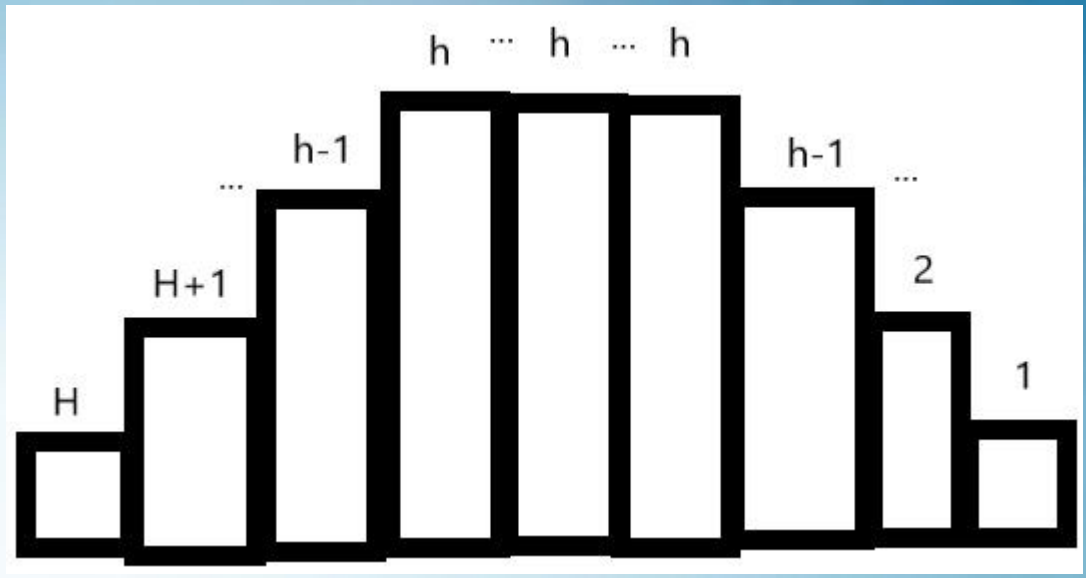


$$sum[h]$$

$$= \begin{cases} (1+h)*h/2 & ,h \le H \\ (1+h)*h/2 + (H+h-1)*(h-H)/2 & ,h > H \end{cases}$$

占用堆数为$\begin{cases} h & ,h \le H \\ h+h-H & ,h > H \end{cases}$

【思路一】



然后再考虑一般情况

当n>sum[h]且n<sum[h+1]时

比sum[h]多的沙包最多一堆可以放h个

不足h个的放在右边和他同样高的一堆旁边

那么额外占用的堆数就为(n-sum)/h上取整。

然后现在就可以二分答案了，二分最高高度

求出满足n>sum[h]且n<sum[h+1]的h，那么

答案就是$\begin{cases} h+(n-sum[h])/h\text{上取整} \\ 2*h-H+\ \ (n-sum[h]/h\text{上取整} \end{cases}$

【补充】

现在思路已经明确了，但注意一个细节，上述思路中的sum[h]已经超过了long long的范围怎么办？

开double，double的精度比long long高

再注意：double中的sum=n 需要写为abs(sum-n)<某个非常小的数，如0.0001。

【核心代码】

二分h

```
ll cx(ll l,ll r){
    if (l==r) return l;
    ll h=(l+r)/2;double sum=0;
    if (h<=m) sum=1.0*(1+h)*h/2;
        else sum=1.0*(1+h)*h/2+(m+h-1)*(h-m)/2;
    if (abs(sum-n)<0.000001) return h+1;
    else if (sum<n) cx(h+1,r);
    else cx(l,h);
}
```

主程序

```
int main(){
    scanf("%lf%lld",&n,&m);
    if (n==1){
        printf("1\n");
        return 0;
    }
    h=cx(1,n)-1;
    if (h<=m) sum=1.0*(1+h)*h/2;
    else sum=1.0*(1+h)*h/2+(m+h-1)*(h-m)/2;
    if (abs(n-sum)<0.00001) ans=0;
    else  ans=(n-sum-1)/h+1;
    if (h<m) ans+=h;
    else ans+=h+h-m;
    printf("%lld\n",ans);
}
```

【思路二】

将思路一化简，由于题目是要求最少摆几堆，而我们有n个沙包，那么很显然答案在1~n之间，我们直接选择二分堆数。

根据法一的分析，我们应求出当堆数为mid时最多可消耗的沙包数sum[mid]，中间变量为最高高度h，而这种情况必然是先严格递增再严格递减。

首先考虑mid ≤ H,此时h=mid,sum[mid]=(1+mid)*mid/2。

再考虑mid > H，此时h=(mid-H-1)/2,$sum[mid] =$

$$\begin{cases} (H-1)*H/2+(H+H+h-1)*h & ,(mid-H-1)\text{为偶} \\ (H-1)*H/2+(H+H+h-1)*h+H+h & ,(mid-H-1)\text{为奇} \end{cases}$$

那么只要二分堆数，满足sum[mid-1]<n<sum[mid]的mid就是答案

【核心代码】

```
ll cx(ll l,ll r){
    if (l==r) return l;
    ll mid=(l+r)/2;double sum=0;ll h=0;
    if (mid>m){
        sum=1.0*(m-1)*m/2;
        h=(mid-m+1)/2;
        sum+=1.0*(m+m+h-1)*h;
        if ((mid-m-1)%2==1) sum+=m+h;
    }
    else sum=1.0*(mid+1)*mid/2;
    if (abs(sum-n)<0.0001) return mid;
    else if (sum<n) cx(mid+1,r);
    else cx(l,mid);
}
```

感谢聆听

个人赛H (CF1326D)Prefix-Suffix Palindrome

题意:
给你一个由小写英文字母组成的字符串s，请找到满足以下条件的最长的字符串t:
- t的长度不超过s
- t是一个回文串
- t=a+b，a是s的前缀，b是s的后缀，a，b可以为空

分析:
- 将两侧对称的部分去掉，问题就可以转化成从剩下的字符串中找从某一边出发的最长回文子串
- 最长回文子串就可以用马拉车或者哈希来做了。

前置知识——字符串哈希

- 什么是哈希？
- 将输入映射到一个值域较小、可以方便比较的范围。
- 核心代码：

```
for (register int i = 1; i <= n; i++) h[i] = (h[i - 1] * base + s[i]) % mod;
p[0] = 1;for (register int i = 1; i <= n; i++) p[i] = p[i - 1] * base % mod;
hash_s(i,j) = (h[j] - h[i - 1] * p[j - i + 1] % mod + mod) % mod;
```

- 其中 $h[i]$ 表示字符串前 i 位的hash值，$p[i]$ 表示 $base^i$，$hash\_s(i,j)$ 是 s 的子串 $s[i \cdots j]$ 的hash值。
- $base$ 随意取，比如131、233、666、114、514……
- mod 需要特别选择，否则容易被卡（哈希碰撞），比如本题卡了998244353、1e9+7以及自然溢出（2^{64}）。

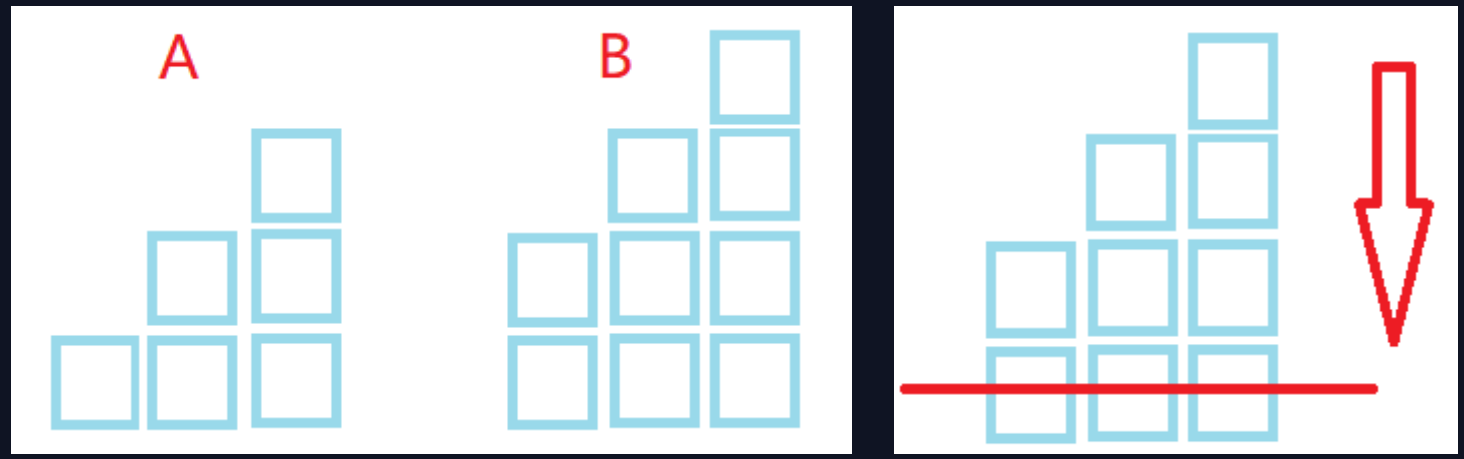
```cpp
#include <cstdio>
#include <cstring>
using namespace std;
const long long maxn = 1e6 + 7, mod = 1e8 + 7, base = 233;
long long nn, t, n, h[maxn], _h[maxn], st, ansl, ansr, anslen, p[maxn];
char s[maxn], ss[maxn];
int main()
{

    p[0] = 1;
    for (long long i = 1; i <= 1e6 + 1; i++) p[i] = p[i - 1] * base % mod;
    for (scanf("%lld", &t); t--;)
    {
        ansl = 0, ansr = -1, anslen = 0;
        scanf("%s", &ss);
        nn = strlen(ss);
        for (st = 0; st < nn - st - 1; ++st)
            if (ss[st] != ss[nn - st - 1]) break;
        n = nn - (st << 1);
        for (long long i = 1; i <= n; ++i) s[i] = ss[i - 1 + st];
        for (long long i = 1; i <= n; ++i) h[i] = (h[i - 1] * base + s[i]) % mod;
        for (long long i = 1; i <= n; ++i) _h[i] = (_h[i - 1] * base + s[n - i + 1]) % mod;
```

```
        for (long long ii = 1; ii <= n; ++ii)
        {
            long long i = 1, j = ii;
            long long x = (h[j] - h[i - 1] * p[j - i + 1] % mod + mod) % mod;
            i = n - ii + 1, j = n;
            long long y = (_h[j] - _h[i - 1] * p[j - i + 1] % mod + mod) % mod;
            if (x == y) anslen = ii, ansl = 1, ansr = ii;
        }
        for (long long ii = 1; ii <= n; ++ii)
        {
            long long i = 1, j = ii;
            long long x = (_h[j] - _h[i - 1] * p[j - i + 1] % mod + mod) % mod;
            i = n - ii + 1, j = n;
            long long y = (h[j] - h[i - 1] * p[j - i + 1] % mod + mod) % mod;
            if (x == y && anslen < ii) anslen = ii, ansl = n - ii + 1, ansr = n;
        }
        for (long long i = 0; i < st; ++i) printf("%c", ss[i]);
        for (long long i = ansl; i <= ansr; ++i) putchar(s[i]);
        for (long long i = st - 1; i > -1; --i) printf("%c", ss[i]);
        puts("");
    }
    return 0;
}
```

I:MUH and Cube Walls

题目大意：给你两堵墙A，B，墙可以任意抬高，放低，求墙A与墙B的匹配次数。

样例：



思路：因为墙可以任意抬高放低，所以墙的高度并不重要，我们只需记录相邻墙体的高度差，然后KMP找匹配即可。

样例解释

A：2 4 5 5 4 3 2 2 2 3 3 2 1

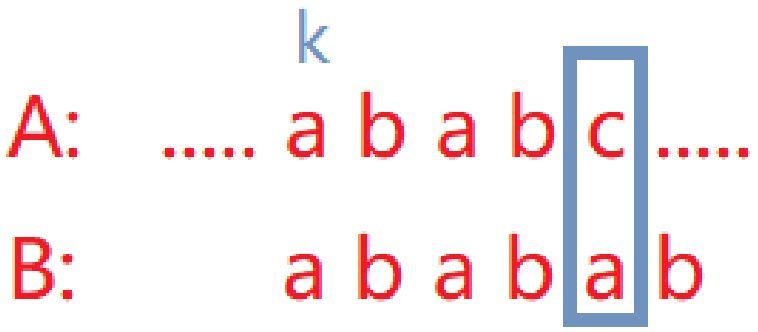
B：3 4 4 3 2

A高度差：2 1 0 -1 -1 -1 0 0 1 0 -1 -1

B高度差：1 0 -1 -1

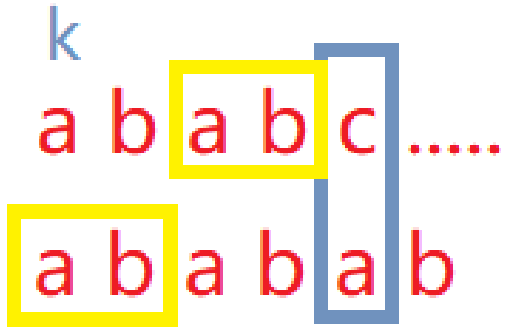
输出：2

KMP

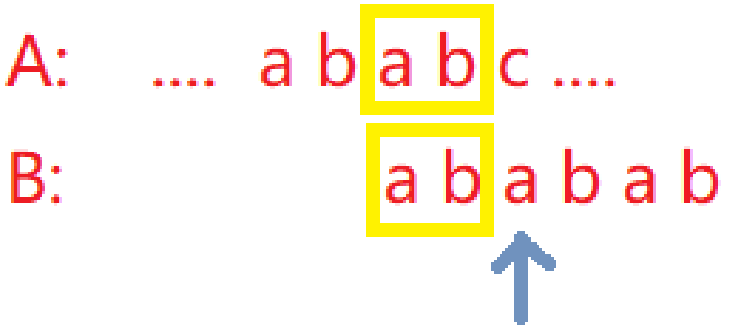


此时 c != a, 那么如果是暴力做法的话，就需要 k ++ ,从k + 1的位置从头开始匹配。

但是我们可以发现一些额外信息：



我们可以发现在匹配过的序列中有一部分（黄框）序列是相同的，我们可以直接将A，B串的黄框框中的序列对其，再继续往后匹配，不需要将匹配起点k后移一位从头开始匹配。

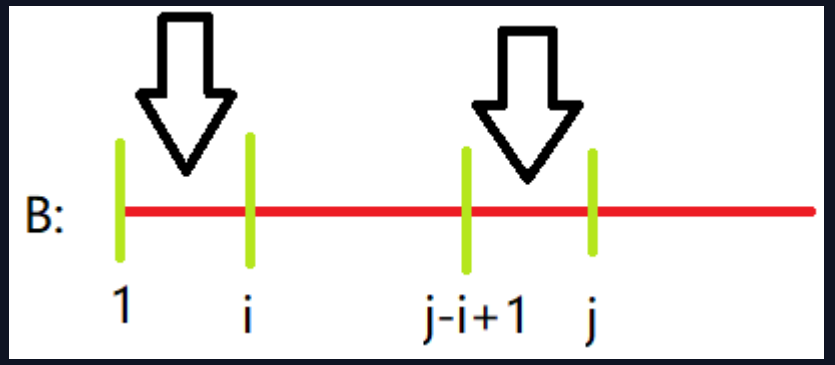


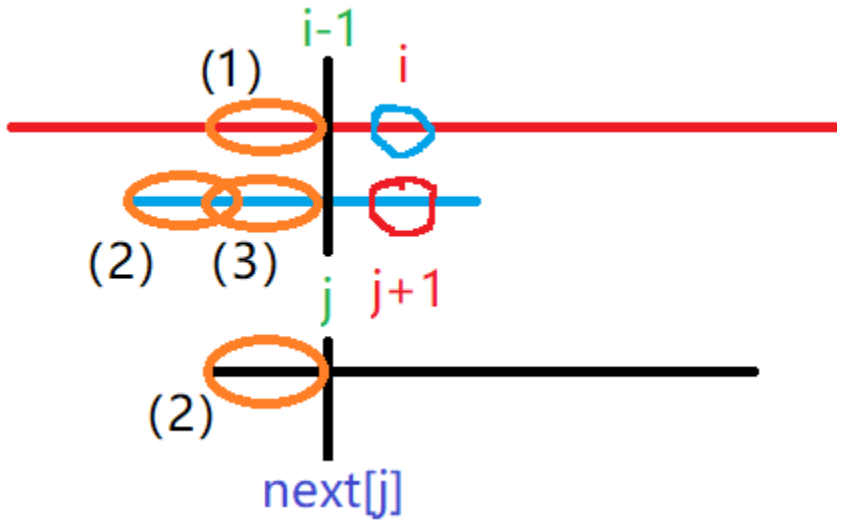
KMP next数组 假设求A串中与B串匹配的子串

next数组含义： next[j] = i 表示

B[1 ~ j] == B[j – i + 1 ~ j]即以B[j]结尾的

后缀与前缀所匹配的最大长度为i：





假设匹配到A[i-1]到B[j]都是相同的，但是A[i] != B[j+1]

那么匹配失败，B串要移动位置重新匹配。

根据求出的next数组，直接将j = next[j]，接着从A串的A[i]位置与B串的B[next[j] + 1]继续往后对比。假如还是匹配不上，那么就迭代如上过程继续去访问next数组移动B串。

Code假设已经求得了next数组后

```
int ans = 0;//存答案个数
for (int i = 1, j = 0; i <= n; i ++ )
{
    while(j && a[i] != b[j + 1]) j = ne[j];//如果匹配失败且j没有退回起点
    if(a[i] == b[j + 1]) j ++ ;//a[i]与b[j+1]匹配成功,就匹配下一位
    if(j == m)//匹配成功
     {
         ans ++ ;
         j = ne[j];//与匹配失败一样,都移动到next[j]继续去匹配
     }
}
```

Code 求next数组

```
//统计非平凡前缀和后缀即前缀和后缀都不是B[1] ~ B[i]这整个字串,所以i要从2开始
for (int i = 2, j = 0; i <= m; i ++ )
    {
        while(j && b[i] != b[j + 1]) j = ne[j];//匹配失败，移动到next[j]
        if(b[i] == b[j + 1]) j ++ ;//匹配成功就继续往后匹配
        ne[i] = j;//记录next[i]的值
    }
```

```
//最后不要忘记特判一下本题的特殊情况
if (m == 1)
{
    printf("%d\n", n);
    return 0;
}
```