> 今日内容
>
> 1. 字符串Hash
> 2. KMP
> 3. 并查集
> 4. ST表
> 5. 带权并查集

字符串Hash

引入问题：给你一个数字x，x<1e10，有q(q<1e6)次询问，每次询问两个区间[a~b] [c~d]，求这段区间的数字是否相同

> 输入
>
> 1114414455 3
>
> 1 2 2 3
>
> 3 4 4 5
>
> 3 4 6 7
>
> 输出
>
> YES
>
> NO
>
> YES

1114414455

hs[1]=1
hs[2]=11
hs[3]=111　　p=10
hs[4]=1114
hs[5]=11144
hs[6]=111441
hs[7]=1114414
[4~5] = h[5]-h[3]*100=11144-11100=44
[6~7] = h[7]-h[5]*100=1114414-1114400=44
44=44 说明这两个字符串相同
[l~r] = h[r]-h[l-1]*p[(r-l+1)]
p[0]=1 p[1]=10 p[2]=100 p[3]=1000

代码：（**本代码有bug**~）（如果字符串数字非常长呢？）

```
10010 1
2 3 5 5
```

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e6+5;
const int P=10;
char s[N];
ll hs[N];

ll getHash(int l,int r){
    int L=hs[l-1],R=hs[r];
    for(int i=1;i<=(r-l+1);i++){//如何改进?
        L*=10;
    }
    return R-L;
}
int main(){
    int q;
    scanf("%s",s+1);
    scanf("%d",&q);
    int ls=strlen(s+1);
    for(int i=1;i<=ls;i++){
        hs[i]=hs[i-1]*P+(s[i]-'0');
    }
    while(q--){
        int a,b,c,d;
        cin>>a>>b>>c>>d;
        if(getHash(a,b)==getHash(c,d)){
            cout<<"YES"<<endl;
        }
        else{
            cout<<"NO"<<endl;
        }
    }
}
```

算法介绍

字符串Hash，把一个任意长度的字符串映射成一个非负整数，并且冲突概率几乎为0。

- 算法步骤:
    1. 取一固定值P,把字符串看成P进制的数，并分配一个大于0的数字，代表每种字符。
    2. p通常取133或13331，模数一般取2^64，避免低效取模，用unsigned long long 代替

问题二：兔子与兔子

算法模板

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
typedef unsigned long long ull ;//2^64-1;
//给你一个数字s，s<1e10，有q(q<1e6)次询问，每次询问两个区间[a~b] [c~d]，求这段区间的数字是否相同;
const int N=1e6+5;
```

```
const int P=131;//131 13331
// mod=2^64  %mod %mod
char s[N];
ull hs[N],p[N]={1}; //p[i]=10^i=p^i
ull getHV(int l,int r ){//得到区间[l~r]的哈希值;
    return hs[r]-hs[l-1]*p[r-l+1];
}
int main(){
    int q;
    scanf("%s",s+1);//下标1开始;
    int n=strlen(s+1);
    for(int i=1;i<=n;i++){
        hs[i]=hs[i-1]*P+s[i];
        p[i]=p[i-1]*P;
    }
    scanf("%d",&q);
    while(q--){
        int a,b,c,d;
        scanf("%d%d%d%d",&a,&b,&c,&d);
        if(getHV(a,b)==getHV(c,d)){
            cout<<"Yes\n";
        }
        else{
            cout<<"No\n";
        }

    }
}
```

问题三：字符串匹配

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
typedef unsigned long long ull ;
const int N=1e6+5;
const int P=131;//进制数 一般131 13331
char s[N],t[N];
ull hs[N];
ull p[N]={1};//p[i]=p^i;
ull getHV(int l, int r){//获取[l~r]区间的哈希值;
    return hs[r]-hs[l-1]*p[r-l+1];
}
int main(){
    scanf("%s",s+1);
    scanf("%s",t+1);
    int ls=strlen(s+1),lt=strlen(t+1);//获取字符串长度;
    if(ls<lt){
        printf("NO\n");
    }
    else{
        for(int i=1;i<=ls;i++){
            hs[i]=hs[i-1]*P+s[i];
            p[i]=p[i-1]*P;
        }
        //计算th哈希值;
        ull th=0;
```

```
        for(int i=1;i<=lt;i++){
            th=th*P+t[i];
        }
        for(int i=1,j=i+lt-1;j<=ls;i++,j++){
            if(getHV(i,j)==th){
                printf("YES\n");
                return 0;
            }
        }
        printf("NO\n");
    }
}
```

KMP

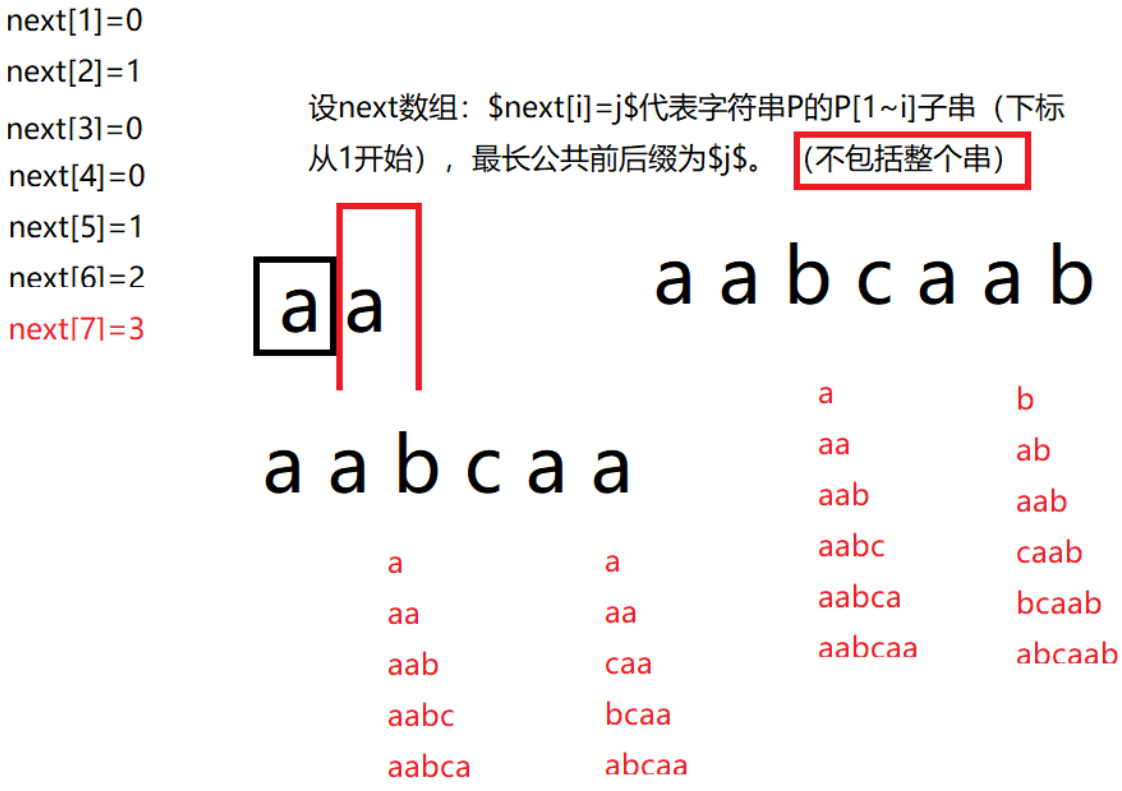
引入：被匹配串$S = ababac$，匹配串$P = abac$，求P在S里出现几次，设字符串p长度为m，s的长度为n。

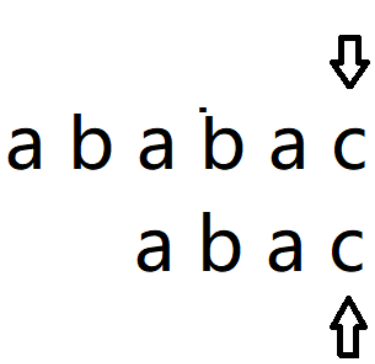
- 暴力解法，时间复杂度$O(n * m)$

如何去高效解决此问题？——KMP算法

利用自身匹配串的特性，减少重复比较。

设next数组：$next[i] = j$代表字符串P的P[1~i]子串（下标从1开始），最长公共前后缀为j。（不包括整个串）





next[1]=0

next[2]=0

next[3]=1

next[4]=0

算法步骤

1. 求出next数组
2. 建立双指针，i指针遍历被匹配串，j指针遍历匹配串，出现匹配失败j指针跳转对应next数组的位置，匹配成功时候也要继续根据next进行跳转。

```
    scanf("%s",s+1);
    scanf("%s",p+1);
    int ls=strlen(s+1),lp=strlen(p+1);
    //先求出匹配串的next数组
    for(int i=2,j=0;i<=lp;i++){
        while(j && p[j+1]!=p[i])j=ne[j];
        if(p[j+1]==p[i])j++;
        ne[i]=j;
    }
    //再用得到的next 去匹配s
    for(int i=1,j=0;i<=ls;i++){
        while(j && p[j+1]!=s[i])j=ne[j];
        if(p[j+1]==s[i])j++;
        if(j==lp){//匹配成功
            cout<<i-lp+1<<endl;
            j=ne[j];
        }
    }
```

- 时间复杂度分析

对于每次i的遍历，j最多增加1，因此不论j怎么根据next跳转，减小的幅度都不会超过增加的幅度。

时间复杂度$O(n+m)$

st表

- 需求：输入一个序列$A[n]$，进行q次查询，每次查询求$[l,r]$区间的最大值。
- 问题：暴力的求法是，每次都遍历一下，求最大值，显然这个非常慢。所以我们需要引入一个高效的数据结构。
- 介绍：st表根据倍增思想实现的数据结构，主要用来求解RMQ问题，$O(1)$的时间复杂度求出某个区间的最大值，最小值，GCD。

算法介绍

预处理

首先，我们设$STMax[i][j]$：从i开始的区间长度为2^j的最大值，即区间范围为$[i, i+2^j-1]$。

如:$STMax[i][0]$ 从i开始区间长度为1的最大值，区间范围[i,i]，求解出[i,i的最大值。

那么很显然，对于原数组$A[]$，我们有$A[i]=STMax[i][0]$。我们直接读入即可。（不用再建立$A[]$数组）

```
    for(int i=1;i<=n;i++){
        scanf("%d",&st[i][0]);
    }
```

我们如何去维护好$STMax[i][j]$呢，很简单，根据max最大值的一个性质，

$max(1,2,3,4)=max(max(1,2),max(3,4))$。是不是看到公式就头晕，让我们来举个**例子**。

假设已知区间$[1,2][3,4]$的最大值分别为a,b，我们要求$[1,4]$的最大值，只需要求$max(a,b)$。

好，我们继续深入，如果要求$[1,8]$的最大值，我们只需要求出$[1,4],[5,8]$的最大值，然后在他们之中再取最大值即可。一直递增，我们可以求出更多的最大值，这个便利用到区间dp的思想。

所以我们可以维护好区间长度，不断递推出更多的区间长度。（看不懂没关系，看下文分析）

我们上文说到，我们对所有$STMax[i][0]$进行赋值，他们的区间长度都是1，最大值都是原数据$A[i]$本身。那么我们是不是得到了所有长度为1的区间的最大值？，因此我们可以去求长度为2的区间的最大值。

接下来我来分析**前三步**。

- 求长度区间为2的最大值

根据$max(max([1])，max([2]))$我们可以得到$[1,2]$这个长度为2的区间的最大值，同样的，我们能得到$[2,3],[3,4][4,5]\dots[N-1,N]$所有长度为2的区间的最大值。

- 求长度区间为4的最大值

根据上面举的例子，求出$[1,4],[2,5],[3,6]\dots$.所有长度区间为4的

- 求长度区间为8的最大值

求出所有长度区间为8，的最大值

.....

我们每一步都可以在上一步的基础上得到更大的区间。区间长度不断翻倍增长(不断乘2)，回到我们上面的定义：

我们设$STMax[i][j]$：从i开始的区间长度为2^j的最大值，所以我们在声明数组的大小的时候，第二维的最大值为整个区间的长度的对数。一般我都设置为21（大概$2^{21}=2e6$）

- 声明ST表

```
int STMax[N][21];
```

- 维护ST表的代码如下

仔细看代码，别看到符号就觉得难。$(1<<j)$代表的是2^j

```
    for(int j=1;j<=log(n);j++){//j代表的是区间长度，n代表原数组的长度，区间长度要
<=log(n),原因查看j的实际含义。
        for(int i=1;i+(1<<j)-1<=n;i++){//右区间不能越界
            STMax[i][j]=max(STMax[i][j-1],STMax[i+(1<<(j-1))][j-1]);
        }
    }
```

查询

我们预处理完$STMax[i][j]$以后，我们就知道了从任意位置i开始，区间**长度**为任意一个2的整数幂的一个区间（$[i,i+2^k-1]$）的最大值。

不妨问问自己，我们建立这样的数据结构有什么意义呢?

假设，我们现在需要求解[1,4]之间的最大值。

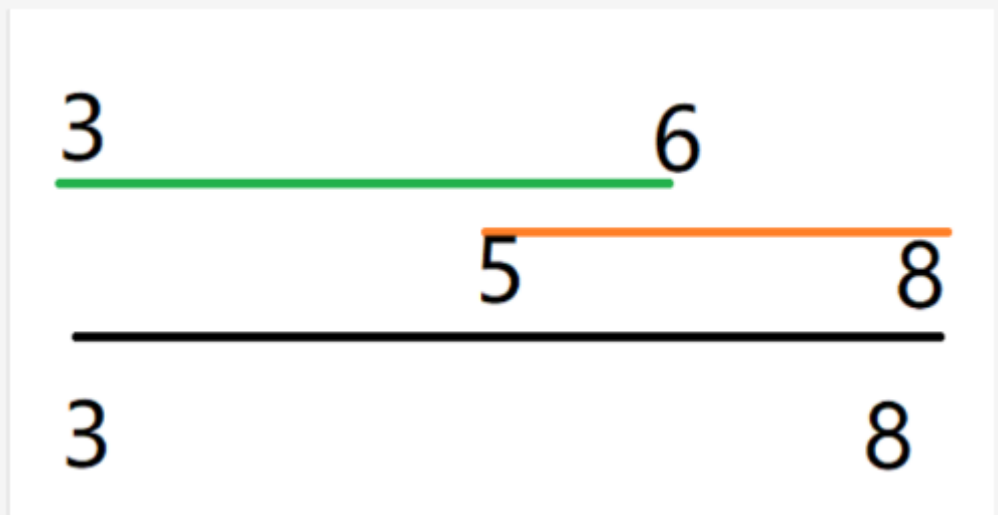
我们可以根据我们的$STMax[i][j]$直接求得，其结果为$STMax[1][2]$。

但是，如果我们需要求解区间长度为6，$[3,8]$的最大值，我们该如何是好?

我们不能根据我们建立的$STMax[i][j]$直接得到答案。因为我们区间的长度只有2的整数次幂，如果从3开始，让$j=2,STMax[3][2]$代表的是$[3,6]$的最大值，如果我们让$j=3,STMax[3][3]$代表的是$[3,10]$的最大值，明显就超出了我们要求的区间。

我们可以根据上文说到的max可以相互嵌套的性质。

如果我们求出$[3,6][5,8]$区间的最大值，然后再取最大值不就是$[3,8]$区间的最大值了吗?



所以我们的策略是这样，对于$[l,r]$的区间，即次区间长度为$le=r-l+1$，我们可以构建两个长度为log_2^{le}的相交区间。这个值很巧妙，会确保两个区间之并集一定覆盖整个区间，还请读者细细体会。

因此对于一个区间的查询

```
int query(int l,int r ){
    int k=log2(r-l+1);
    return max(STMax[l][k],STMax[r-(1<<k)+1][k]);
}
```

我们反复求解$log2(r-l+1)$也挺浪费时间的，所以我们可以预处理，建立$log[]$数组

```
    for(int i=2;i<N;i++){
        lg[i]=lg[i/2]+1;
    }
```

总结

st表是一种能够很好地完成一种**特定**的RMQ的算法。

因为局限性比较大，不能反复修改但是时间复杂度非常优秀。

预处理$O(nlog_2n)$ 查询$O(1)$

算法模板

以洛谷的模板题解完结

模板题

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+5;
int STMax[N][23];
int lg[N];
```

```
int query(int l,int r ){
    int k=lg[r-l+1];
    return max(STMax[l][k],STMax[r-(1<<k)+1][k]);
}

int main() {
    int n,m;
    for(int i=2;i<N;i++){
        lg[i]=lg[i/2]+1;
    }
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&STMax[i][0]);
    }
    for(int j=1;j<=lg[n];j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
            STMax[i][j]=max(STMax[i][j-1],STMax[i+(1<<(j-1))][j-1]);
        }
    }
    while(m--){
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",query(l,r));
    }

}
```

并查集

问题引入：街头打架

核心思想：一个元素代表一个集合

- 查询

```
int find(int x)
{
    if(fa[x] == x)
        return x;
    else
        return find(f[x]);
}
```

- 合并

```
void merge(int i, int j)
{
    fa[find(i)] = find(j);
}
```

模板

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

```
typedef long long ll;
const int N=1e4+5;
int f[N];
int find(int i){
    return f[i]==i? i:f[i]=find(f[i]);
}
void merge(int a ,int b){// a归属b的阵营;
    f[find(a)]=find(b);
}
int main(){
    int n ,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)f[i]=i;
    while(m--){
        int x,y,z;
        cin>>z>>x>>y;
        if(z==1){
            u(x,y);
        }
        else{
            if(find(x)==find(y))cout<<"Y\n";
            else cout<<"N\n";
        }
    }
}
```

带权并查集