字符串哈希

hash，就是将一样东西映射成另一个东西，类似map。字符串hash，就相当于将字符串映射成唯一表示的数字。通过字符串的hash值，我们可以达到$O(1)$比较两个串是否相等。

字符串哈希

- 构造方法
  - 自然溢出
  - 单哈希法
  - 双哈希法
- 模板
  - 子串查找
- 最长公共子字符串
- 确定字符串中不同子字符串数量
- Rabin-Karp
- 例题
  - 最短公共超串(优秀的双模数hash模板)
- 树哈希
  - 树哈希的表示方法
  - [NOIP2018 普及组] 对称二叉树
  - 【模板】树同构
  - [JSOI2016]独特的树叶

构造方法

自然溢出

通过自然溢出，不需要定义mod，而是利用unsigned long long的自然溢出处理长串的哈希值。

```
#define ull unsigned long long
ull base,hash[N],p[N]; //其中p[i]表示base^i
```

单哈希法

有了mod的情况下，开long long求余也可以构造字符串的哈希值。

双哈希法

为了避免哈希冲突的情况(即两个字符串阴差阳错地得到了同样的哈希值)，我们可以取两个不同的mod，这样构造的hash值就相当安全。

模板

子串查找

给定字符串A,B，求B在A中出现的次数。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
typedef long long ll;
```

```
const int base=131;
const int mod=1e16+7;
int bb[1000005],sum[1000005],hs;

int get_hs(int i,int l){ //获取A串从i开始到i+l位的哈希值
    return sum[i+l-1]-sum[i-1]*bb[l];
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    string A,B;
    cin>>A;
    cin>>B;
    int lena=A.length(),lenb=B.length(),ans=0;
    for(int i=0;i<lenb;i++){
        hs=(hs*base+B[i]-'A'+1);
    }
    bb[0]=1;
    for(int i=1;i<=lena;i++){
        bb[i]=bb[i-1]*base;
        sum[i]=(sum[i-1]*base+A[i-1]-'A'+1);
    }
    for(int i=lenb;i<=lena;i++){
        if(sum[i]-sum[i-lenb]*bb[lenb]==hs){
            ans++;
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

最长公共子字符串

如果存在长度为k的最长公共子字符串，那么k-1长的公共子字符串也必定存在。因此可以二分最长公共子字符串。我们将所有长度为k的子串分别进行哈希，将哈希值放到n个哈希表中存储，之后求交集即可。复杂度$O(nlog_2\frac{n}{m})$

确定字符串中不同子字符串数量

遍历所有长度为l=1,...,n的子串，对于所有长度l，将其Hash值乘以相同的b的幂次方并存入数组，数组中不同元素的数量等于字符串中长度不同的子串数量。

我们记$h[i]$为$Hash$的前缀字符，并定义$h[0]=0$

```
int count_unique_substrings(string const& s) {
  int n = s.size();
  const int b = 31;
  const int m = 1e9 + 9;
  vector<long long> b_pow(n);
  b_pow[0] = 1;
```

```
  for (int i = 1; i < n; i++) b_pow[i] = (b_pow[i - 1] * b) % m;

  vector<long long> h(n + 1, 0);
  for (int i = 0; i < n; i++)
    h[i + 1] = (h[i] + (s[i] - 'a' + 1) * b_pow[i]) % m;

  int cnt = 0;
  for (int l = 1; l <= n; l++) {
    set<long long> hs;
    for (int i = 0; i <= n - l; i++) {
      long long cur_h = (h[i + l] + m - h[i]) % m;
      cur_h = (cur_h * b_pow[n - i - 1]) % m;
      hs.insert(cur_h);
    }
    cnt += hs.size();
  }
  return cnt;
}
```

Rabin-Karp

使用哈希思想，对T串计算哈希值，并以相同的方式计算S串每一个位置对应子串的HASH值。

解决哈希溢出的方法：大数取模

解决哈希冲突的方法：直接匹配字符串

例题

最短公共超串(优秀的双模数hash模板)

给定串T,S，求一个长度最短的串A使得T和S均为A的子串。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e6+7;
typedef long long ll;
const int mod1=1e9+7,mod2=1e9+9;

struct Hash{
    ll has1[N],has2[N];
    ll bas1[N],bas2[N];
    ll p1,p2;
    void init(string s){
        p1=2333;//如有误差手动修改
        p2=17;
        has1[0]=has2[0]=0;
        bas1[0]=bas2[0]=1;
        int len=s.length();
        for(int i=1;i<=len;++i){//多串hash可以这块预处理
            bas1[i]=(bas1[i-1]*p1)%mod1;
```

```
            bas2[i]=(bas2[i-1]*p2)%mod2;
        }
        for(int i=1;i<=len;++i){
            has1[i]=(has1[i-1]*p1+s[i-1])%mod1;
            has2[i]=(has2[i-1]*p2+s[i-1])%mod2;
        }
    }
    ll gethash1(int r,int len){//返回以r结尾长度len的子串has1值
        return ((has1[r]-has1[r-len]*bas1[len])%mod1+mod1)%mod1;
    }
    ll gethash2(int r,int len){//返回以r结尾长度len的子串has2值
        return ((has2[r]-has2[r-len]*bas2[len])%mod2+mod2)%mod2;
    }
}HS1,HS2;


signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    //srand(time(0));
    string A,B;
    cin>>A;cin>>B;
    if(A.find(B)!=A.npos){
        cout<<A<<"\n";
        return 0;
    }
    if(B.find(A)!=B.npos){
        cout<<B<<"\n";
        return 0;
    }
    int lena=A.length(),lenb=B.length();
    HS1.init(A);HS2.init(B);
    string ans=A+B;
    for(int len=min(lena,lenb);len>=0;len--){
        ll x1=HS1.gethash1(lena,len),x2=HS2.gethash1(len,len); //A+B
        ll y1=HS1.gethash1(len,len),y2=HS2.gethash1(lenb,len); //B+A
        ll a1=HS1.gethash2(lena,len),a2=HS2.gethash2(len,len); //A+B
        ll b1=HS1.gethash2(len,len),b2=HS2.gethash2(lenb,len); //B+A
        if(y1==y2&&b1==b2){
            ans=B.substr(0,lenb-len)+A;
            break;
        }else if(x1==x2&&a1==a2){
            ans=A.substr(0,lena-len)+B;
            break;
        }
    }
    cout<<ans<<"\n";
    return 0;
}
```

树哈希

可用来判断树是否同构，判断无根树同构的时候，可以比较重心为根的Hash值或者比较每个点为根的Hash值。

树哈希的表示方法

$f_x = 1 + \sum_{y \in son_x} f_y \times prime(size_y)$

[NOIP2018 普及组] 对称二叉树

给定一颗有根二叉树，点有点权。如果将二叉树左右儿子交换，结构相同且对应位置点权相同，则称为对称二叉树。问最大对称二叉子树的节点数。

```
#include<bits/stdc++.h>
#define int long long
#define V1 (999999751)
#define V2 (299999827)
#define V3 (100000007)
#define md (89999794200117649)
#define mdd (999999786000011449)
using namespace std;
const int maxn=1e6+7;

int n,ans,val[maxn],rs[maxn],ls[maxn];
int he[maxn],hal[maxn],Hal[maxn],har[maxn],Har[maxn];

void dfs(int x,int fa){
    if(ls[x]>0) dfs(ls[x],x);
    if(rs[x]>0) dfs(rs[x],x);
    he[x]=he[ls[x]]+he[rs[x]]+1;
    if(he[ls[x]]==he[rs[x]]&&hal[ls[x]]==har[rs[x]]&&Hal[ls[x]]==Har[rs[x]]){
        ans=max(ans,he[x]);
    }
    hal[x]=hal[ls[x]]*V1+val[x]*V2+hal[rs[x]]*V3;
    Hal[x]=Hal[ls[x]]*V1+val[x]*V2+Hal[rs[x]]*V3;
    hal[x]%=md;
    Hal[x]%=mdd;
    har[x]=har[rs[x]]*V1+val[x]*V2+har[ls[x]]*V3;
    Har[x]=Har[rs[x]]*V1+val[x]*V2+Har[ls[x]]*V3;
    har[x]%=md;
    Har[x]%=mdd;
}

signed main(){
    cin>>n;
    for(int i=1;i<=n;i++) cin>>val[i];
    for(int i=1;i<=n;i++){
        cin>>ls[i]>>rs[i];
    }
    dfs(1,0);
    cout<<ans;
    return 0;
}
```

【模板】树同构

给定m棵树，问与每个数同构的树的最小编号。

判断无根树同构，通过两遍dfs树形dp，求出每个点为根时的Hash值，排序后比较即可。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int maxn=2e5+7;
const int mod=1e9+7;
const int N=55;

bool is_pri[maxn];
int pri[maxn];
int sz[N],f[N],g[N];
int n,m;
vector<int>hs[N];
vector<int>G[N];

void get_pri(int N){
    int cnt=0;
    memset(is_pri,1,sizeof(is_pri));
    for(int i=2;i<N;i++){
        if(is_pri[i]) pri[++cnt]=i;
        for(int j=1;j<=cnt&&pri[j]*i<=N;j++){
            is_pri[i*pri[j]]=0;
            if(i%pri[j]==0) break;
        }
    }
}

void dfs1(int x,int fa){
    sz[x]=f[x]=1;
    for(int i=0;i<G[x].size();i++){
        int to=G[x][i];
        if(to==fa) continue;
        dfs1(to,x);
        f[x]+=f[to]*pri[sz[to]];
        sz[x]+=sz[to];
    }
}

void dfs2(int x,int fa,int fa_f){
    g[x]=f[x]+fa_f*pri[n-sz[x]];
    fa_f*=pri[n-sz[x]];
    for(int i=0;i<G[x].size();i++){
        int to=G[x][i];
        if(to==fa) continue;
        dfs2(to,x,fa_f+f[x]-f[to]*pri[sz[to]]);
    }
}

bool Equal(int x,int y){
    if(hs[x].size()!=hs[y].size()) return false;
    for(int i=0;i<hs[x].size();i++){
```

```
        if(hs[x][i]!=hs[y][i]) return false;
    }
    return true;
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    get_pri(10000);
    cin>>m;
    int fa;
    for(int i=1;i<=m;i++){
        cin>>n;
        for(int j=1;j<=n;j++) G[j].clear();
        for(int j=1;j<=n;j++){
            cin>>fa;
            if(fa){
                G[j].push_back(fa);
                G[fa].push_back(j);
            }
        }
        dfs1(1,0);
        dfs2(1,0,0);
        for(int j=1;j<=n;j++) hs[i].push_back(g[j]);
        sort(hs[i].begin(),hs[i].end());
    }
    puts("1");
    for(int i=2;i<=m;i++){
        for(int j=1;j<=i;j++){
            if(Equal(i,j)){
                printf("%d\n",j);
                break;
            }
        }
    }
    return 0;
}
```

[JSOI2016]独特的树叶

给两棵树，树B为树A加上一个叶节点然后将结点编号打乱得到的，问多余的叶子是树B中的那一个结点。

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
using namespace std;
const int N=100010;
bool is_pri[2000010];
int pri[N];
int in[N],near[N];

void get_pri(int MX){
    int cnt=0;
    memset(is_pri,1,sizeof(is_pri));
```

```
    for(int i=2;i<MX;i++){
        if(is_pri[i]) pri[++cnt]=i;
        for(int j=1;j<=cnt&&pri[j]*i<MX;j++){
            is_pri[i*pri[j]]=0;
            if(i%pri[j]==0) break;
        }
    }
}

struct Edge{
    int nxt,to;
};

struct Tree{
    int f[N],g[N],sz[N],head[N];
    int n,tot;
    Edge edge[N<<1];
    void init(int nn){
        n=nn;
        tot=1;
        memset(head,0,sizeof(head));
    }
    void addedge(int u,int v){
        edge[tot].to=v;
        edge[tot].nxt=head[u];
        head[u]=tot++;
    }
    void dfs1(int x,int fa){
        sz[x]=f[x]=1;
        for(int i=head[x];i;i=edge[i].nxt){
            int y=edge[i].to;
            if(y==fa) continue;
            dfs1(y,x);
            f[x]+=f[y]*pri[sz[y]];
            sz[x]+=sz[y];
        }
    }
    void dfs2(int x,int fa,int fa_f){
        g[x]=f[x]+fa_f*pri[n-sz[x]];
        fa_f*=pri[n-sz[x]];
        for(int i=head[x];i;i=edge[i].nxt){
            int y=edge[i].to;
            if(y==fa) continue;
            dfs2(y,x,fa_f+f[x]-f[y]*pri[sz[y]]);
        }
    }
}tr1,tr2;

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    get_pri(2000000);
    int n;cin>>n;
    tr1.init(n);
    tr2.init(n+1);
    memset(in,0,sizeof(in));
    unordered_set<int>se;
    for(int i=0,u,v;i<n-1;i++){
```

```
        cin>>u>>v;
        tr1.addedge(u,v);
        tr1.addedge(v,u);
    }
    for(int i=0,u,v;i<n;i++){
        cin>>u>>v;
        tr2.addedge(u,v);
        tr2.addedge(v,u);
        ++in[u],++in[v];
        near[u]=v,near[v]=u;
    }
    tr1.dfs1(1,0);
    tr1.dfs2(1,0,0);
    for(int i=1;i<=n;i++) se.insert(tr1.g[i]);
    tr2.dfs1(1,0);
    tr2.dfs2(1,0,0);
    for(int i=1;i<=n+1;i++){
        if(in[i]!=1) continue;
        if(se.count(tr2.g[near[i]]-2)){
            printf("%d\n",i);
            break;
        }
    }
    return 0;
}
```