JYCの算法竞赛模板

求个 **star** (っ•ω•)っ)) (っ•ω•)っ)) (っ•ω•)っ))

目录

JYCの算法竞赛模板

- 缺省源
- 数据结构
  - 树状数组
  - 线段树
  - 树链剖分（重链剖分）（LCA模板）
  - 平衡树
- 字符串
  - 字符串哈希（多重哈希）
  - Z函数（扩展KMP）
- 数学
  - 高精度
  - 快速幂&矩阵乘法
  - 扩展欧几里得
  - 线性筛欧拉函数
  - 组合数学
    - 阶乘相关的初始化
    - 错位排列
    - 卡特兰数
    - 斯特林数
      - 第二类斯特林数
      - 第一类斯特林数
- 杂项
  - 火车头
  - 写在最后（来自Benq大神的几句话）

缺省源

模板正式开始

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

//#define ONLINE
#ifndef ONLINE
#define debug(...) fprintf(stderr,##__VA_ARGS__)
#else
#define debug(...) ;
#endif

using LL=long long;
using PII=pair<int,int>;

```

```cpp
const int mod=998244353;
mt19937 rng(chrono::system_clock::now().time_since_epoch().count());

//double关键字比大小
#define eps (1e-8)
inline int sign(const double& x){
    if(fabs(x)<eps) return 0;
    return x>0.0?1:-1;
}
inline int dcmp(const double& x,const double& y){
    return sign(x-y);
}

//快读
template<typename T>
inline T READ(){
    T x=0; bool f=0; char c=getchar();
    while(c<'0' || c>'9') f|=(c=='-'),c=getchar();
    while(c>='0' && c<='9') x=x*10+c-'0',c=getchar();
    return f?-x:x;
}
inline int read(){return READ<int>();}
inline LL readLL(){return READ<LL>();}
```

数据结构

树状数组

```cpp
template<typename T>//T is int or LL
class FenTree{
private:
    int n;
    vector<T>c;
    inline int lowbit(const int& x){
        return x&(-x);
    }
public:
    FenTree(int n_){
        c.assign(n=n_,T(0));
    }
    void init(int n_){
        c.assign(n=n_,T(0));
    }
    void add(int i,int x){
        for(;i<n;i+=lowbit(i)) c[i]+=x;
    }
    T query(int i){
        T res=0;
        for(;i;i-=lowbit(i)) res+=c[i];
        return res;
    }
};
```

线段树

封装的不是很好，具体 $Info$ 的传递还是得自己修改源码，没法直接传个 $class$ 进来

初始化时，传入一个一维 $vector\ a$ 以及它的长度 n，数据存储在下标 $1\sim n$

```
#define ls (id<<1)
#define rs (id<<1|1)
class SGT{//以线段树维护区间最大值为例，支持区间修改
private:
    struct Node{
        int l,r;
        int mx; bool tag;
        friend Node operator +(const Node& A,const Node& B){
            Node ret;
            ret.l=A.l,ret.r=B.r;

            //update the main content
            ret.mx=max(A.mx,B.mx);
            ret.tag=0;

            return ret;
        }
    };
    vector<Node>q;
    void spread(int id){
        if(q[id].l==q[id].r) return;
        //spread the lazy tag
        if(q[id].tag){
            q[ls].mx=q[id].mx;
            q[rs].mx=q[id].mx;
            q[ls].tag=1;
            q[rs].tag=1;
            q[id].tag=0;
        }
    }
    void build(const vector<int>& a,int l,int r,int id=1){
        if(l==r){
            q[id].l=l,q[id].r=r;

            //init value
            q[id].mx=a[l],q[id].tag=0;

            return;
        }
        int mid=l+r>>1;
        build(a,l,mid,ls);
        build(a,mid+1,r,rs);
        q[id]=q[ls]+q[rs];
    }
public:
    SGT(const vector<int>& a,int n){
        q.resize(n*5);
        build(a,1,n);
    }
    void init(const vector<int>& a,int n){
        q.resize(n*5);
```

```
        build(a,1,n);
    }
    void modify(int l,int r,int val,int id=1){
        if(q[id].l==l && q[id].r==r){
            //modify the value
            q[id].mx=val,q[id].tag=1;
            return;
        }
        spread(id);
        int mid=q[id].l+q[id].r>>1;
        if(r<=mid) modify(l,r,val,ls);
        else if(l>mid) modify(l,r,val,rs);
        else modify(l,mid,val,ls),modify(mid+1,r,val,rs);
        q[id]=q[ls]+q[rs];
    }
    int query(int l,int r,int id=1){
        if(q[id].l==l && q[id].r==r) return q[id].mx;
        spread(id);
        int mid=q[id].l+q[id].r>>1;
        if(r<=mid) return query(l,r,ls);
        else if(l>mid) return query(l,r,rs);
        return max(query(l,mid,ls),query(mid+1,r,rs));
    }
    int get(int pos,int id=1){
        if(q[id].l==q[id].r) return q[id].mx;
        int mid=q[id].l+q[id].r>>1;
        if(pos<=mid) return get(pos,ls);
        return get(pos,rs);
    }
};
#undef ls
#undef rs
```

树链剖分（重链剖分）（LCA模板）

直接把“P3384【模板】重链剖分/树链剖分”的代码复制了一遍

因为原题有取模操作，抄模板的时候记得**删去取模，删去取模，删去取模！！！**

也可以把树链剖分作为 LCA 的模板来使用，比倍增的写法优秀很多

```
void heavy_path_decomposition(){
    int n=read(),T=read(),root=read();// mod=read();
    vector<int>a(n+1);
    for(int i=1;i<=n;i++) a[i]=read()%mod;//origin value
    vector<vector<int>>e(n+1);
    for(int i=1,u,v;i<n;i++){
        u=read(),v=read();
        e[u].push_back(v);
        e[v].push_back(u);
    }
    vector<int>dep(n+1),fa(n+1),siz(n+1),son(n+1);
    auto dfs1=[&](auto self,int u,int pre)->void{
        dep[u]=dep[pre]+1,fa[u]=pre,siz[u]=1;
        for(int v:e[u]){
            if(v==pre) continue;
            self(self,v,u);
```

```
            siz[u]+=siz[v];
            if(siz[v]>siz[son[u]]) son[u]=v;
        }
    };
    dfs1(dfs1,root,0);
    vector<int>id(n+1),nw(n+1),top(n+1);
    int timStamp=0;
    auto dfs2=[&](auto self,int u,int t)->void{
        id[u]=++timStamp,nw[timStamp]=u,top[u]=t;
        if(!son[u]) return;
        self(self,son[u],t);
        for(int v:e[u]){
            if(v==fa[u] || v==son[u]) continue;
            self(self,v,v);
        }
    };
    dfs2(dfs2,root,root);
    auto LCA=[&](int u,int v)->int{
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            u=fa[top[u]];
        }
        return dep[u]<dep[v]?u:v;
    };
    vector<int>sgt_init(n+1);
    for(int i=1;i<=n;i++) sgt_init[i]=a[nw[i]];
    SGT sgt(sgt_init,n);//sgt needs to support seg add, seg query
    auto modify_path=[&](int u,int v,LL val)->void{
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            sgt.modify(id[top[u]],id[u],val);
            u=fa[top[u]];
        }
        if(dep[u]<dep[v]) swap(u,v);
        sgt.modify(id[v],id[u],val);
    };
    auto query_path=[&](int u,int v)->LL{
        LL cnt=0ll;
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            cnt+=sgt.query(id[top[u]],id[u]);
            u=fa[top[u]];
        }
        if(dep[u]<dep[v]) swap(u,v);
        cnt+=sgt.query(id[v],id[u]);
        return cnt%mod;
    };
    auto modify_tree=[&](int u,int val)->void{
        sgt.modify(id[u],id[u]+siz[u]-1,val);
    };
    auto query_tree=[&](int u)->LL{
        return sgt.query(id[u],id[u]+siz[u]-1);
    };
    for(int op,u,v,w;T--;){
        op=read(),u=read();
        if(op==1){
            v=read(),w=read();
            modify_path(u,v,w);
```

```
        }
        else if(op==2){
            v=read();
            printf("%lld\n",query_path(u,v));
        }
        else if(op==3){
            w=read();
            modify_tree(u,w);
        }
        else{//op==4
            printf("%lld\n",query_tree(u));
        }
    }
}
```

平衡树

非常朴素的 $fhqTreap$，用 $vector$ 实现了一下动态开点

```
template<typename T>
class fhqTreap{
private:
    struct Node{
        int l,r,siz; LL rnd;
        T val,sum;
        Node(){
            l=r=siz=0; rnd=0ll;
            val=sum=T(0);
        }
        Node(int l_,int r_,int siz_,LL rnd_,T val_,T sum_){
            l=l_,r=r_,siz=siz_; rnd=rnd_;
            val=val_,sum=sum_;
        }
    };
    vector<Node>q;
    int root,rootX,rootY,rootZ;
    int New(T val){
        Node new_node=Node(0,0,1,rng(),val,val);
        q.push_back(new_node);
        return q.size()-1;
    }
    void Update(int id){
        q[id].siz=q[q[id].l].siz+q[q[id].r].siz+1;
        q[id].sum=q[q[id].l].val+q[q[id].r].val+q[id].val;
    }
    void Split(int id,T key,int& idX,int& idY){
        if(id==0){
            idX=idY=0;
            return;
        }
        if(q[id].val<=key){
            idX=id;
            Split(q[id].r,key,q[id].r,idY);
        }
        else{
            idY=id;
            Split(q[id].l,key,idX,q[id].l);
```

```
        }
        Update(id);
    }
    int Merge(int l,int r){
        if(l==0 || r==0) return l+r;
        if(q[l].rnd<=q[r].rnd){
            q[r].l=Merge(l,q[r].l);
            Update(r);
            return r;
        }
        else{
            q[l].r=Merge(q[l].r,r);
            Update(l);
            return l;
        }
    }
public:
    fhqTreap(){
        init();
    }
    void init(){
        root=0; q.clear();
        Node empty_node=Node();
        q.push_back(empty_node);
    }
    void insert(T val){
        Split(root,val,rootX,rootY);
        root=Merge(Merge(rootX,New(val)),rootY);
    }
    void erase(T val){//actually, 'extract' may be more precise
        Split(root,val,rootX,rootZ);
        Split(rootX,val-1,rootX,rootY);
        rootY=Merge(q[rootY].l,q[rootY].r);
        root=Merge(Merge(rootX,rootY),rootZ);
    }
    T prev(T val){
        Split(root,val-1,rootX,rootY);
        int tmp=rootX;
        while(q[tmp].r) tmp=q[tmp].r;
        root=Merge(rootX,rootY);
        return q[tmp].val;
    }
    T next(T val){
        Split(root,val,rootX,rootY);
        int tmp=rootY;
        while(q[tmp].l) tmp=q[tmp].l;
        root=Merge(rootX,rootY);
        return q[tmp].val;
    }
    int rank(T val){//val's rank
        Split(root,val-1,rootX,rootY);
        int ans=q[rootX].siz+1;
        root=Merge(rootX,rootY);
        return ans;
    }
    T get(int rank){
        int id=root;
        while(1){
```

```
//          printf("id:%d val:%d l:%d lsiz:%d r:%d rsiz:%d|rank:%d\n",id,q[id].val,q[id].l,q[q[id].l].siz,q[id].r,q[q[id].r].siz,rank);
            if(q[q[id].l].siz>=rank) id=q[id].l;
            else if(q[q[id].l].siz+1==rank) return q[id].val;
            else{
                rank-=(q[q[id].l].siz+1);
                id=q[id].r;
            }
        }
    }
//  void output(){
//      auto dfs=[&](auto self,int u)->void{
//          if(!u) return;
//          self(self,q[u].l);
//          debug("%d|val=%d ls=%d rs=%d\n",u,q[u].val,q[u].l,q[u].r);
//          self(self,q[u].r);
//      };
//      debug("output fhqTreap\n");
//      dfs(dfs,root);
//      debug("\n");
//  }
};
```

字符串

字符串哈希（多重哈希）

修改 HL（$hash\_layer$） 以决定使用几重哈希

$init$ 时，传入字符数组首地址，字符串长度（以及是否倒着求哈希，0和缺省为正着求，1为倒着求）

注意字符数组内数据的下标是 $1\sim n$，$1\sim n$，$1\sim n$！！！

```
const int HL=2;//hash layer
namespace HC{//Hash Const
    const int P[4]={13331,233,131,19260817};
    const int MOD[4]={(int)(1e9+7),998244353,1004535809,754974721};
    LL ksm[N][4];
    void init(int use=HL){
        for(int j=0;j<use;j++){
            ksm[0][j]=1;
            for(int i=1;i<N;i++){
                ksm[i][j]=(ksm[i-1][j]*P[j])%MOD[j];
            }
        }
    }
}
template<int T>//T must be a constant
class Hash{
private:
    vector<array<LL,T>>h;
    bool sign;//0:normal 1:reverse
public:
    Hash(char *s,int n,bool sign_=0){
        init(s,n,sign_);
    }
```

```
    void init(char *s,int n,bool sign_=0){//s stores at pos [1,n], and just give para s (NOT s+1)
        h.resize(n+2);
        sign=sign_;
        if(!sign){
            for(int j=0;j<T;j++){
                h[0][j]=0;
                for(int i=1;i<=n;i++){
                    h[i][j]=(h[i-1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
                }
            }
        }
        else{
            for(int j=0;j<T;j++){
                h[n+1][j]=0;
                for(int i=n;i>0;i--){
                    h[i][j]=(h[i+1][j]*HC::P[j]+s[i]-'a'+1)%HC::MOD[j];
                }
            }
        }
    }
    array<LL,T>calc(const int& l,const int& r){
        array<LL,T>ret;
        if(!sign){
            for(int j=0;j<T;j++){
                ret[j]=h[r][j]-h[l-1][j]*HC::ksm[r-l+1][j];
            }
        }
        else{
            for(int j=0;j<T;j++){
                ret[j]=h[l][j]-h[r+1][j]*HC::ksm[r-l+1][j];
            }
        }
        for(int j=0;j<T;j++){
            ret[j]=(ret[j]%HC::MOD[j]+HC::MOD[j])%HC::MOD[j];
        }
        return ret;
    }
    static bool check(const array<LL,T>& a,const array<LL,T>& b){
        for(int i=0;i<T;i++){
            if(a[i]!=b[i]) return 0;
        }
        return 1;
    }
};
```

Z函数（扩展KMP）

约定：字符串下标以 0 为起点

定义函数 $z[i]$ 表示 s 和 $s[i,n-1]$（即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被成为 s 的 **Z函数**。特别的，$z[0]=0$

```
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
```

```
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

数学

高精度

BigIntTiny Template Link，顺便从大佬的 README 里放个 Features preview，不支持的功能就现写吧。

operators	BigIntHex	BigIntDec	BigIntMini	BigIntTiny
constructor Bigint	✓	✓	✓	✓
constructor int	✓	✓	✓	✓
constructor char*	✓	✓	✕	✕
constructor string	✓	✓	✓	✓
=Bigint	✓	✓	✓	✓
=int	✓	✓	✓	✓
=string	✓	✓	✓	✓
=char*	✓	✓	✓	✕
<, ==, >, <=, >=, != Bigint	✓	✓	✓	✓
+, -, *, /, % int	✕	✕	✕	✓
+=, -=, *=, /=, %= int	✕	✕	✕	✕
+, -, *, /, % Bigint	✓	✓	✓	✓
+=, -=, *=, /=, %= Bigint	✓	✓	✕	✕
Base conversion	✓	✓	✕	✕
Efficiency	★★★★	★★★	★★	★

```
struct BigIntTiny {
    int sign;
    std::vector<int> v;

    BigIntTiny() : sign(1) {}
    BigIntTiny(const std::string &s) { *this = s; }
    BigIntTiny(int v) {
        char buf[21];
```

```
        sprintf(buf, "%d", v);
        *this = buf;
    }
    void zip(int unzip) {
        if (unzip == 0) {
            for (int i = 0; i < (int)v.size(); i++)
                v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i * 4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
        } else
            for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >= 0; i--)
                a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] = (i & 1) ? a / 10 : a % 10;
        setsign(1, 1);
    }
    int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos]; }
    BigIntTiny &setsign(int newsign, int rev) {
        for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
            v.erase(v.begin() + i);
        sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ? newsign * sign : newsign);
        return *this;
    }
    std::string to_str() const {
        BigIntTiny b = *this;
        std::string s;
        for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
            s += char(*(b.v.rbegin() + i) + '0');
        return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
    }
    bool absless(const BigIntTiny &b) const {
        if (v.size() != b.v.size()) return v.size() < b.v.size();
        for (int i = (int)v.size() - 1; i >= 0; i--)
            if (v[i] != b.v[i]) return v[i] < b.v[i];
        return false;
    }
    BigIntTiny operator-() const {
        BigIntTiny c = *this;
        c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
        return c;
    }
    BigIntTiny &operator=(const std::string &s) {
        if (s[0] == '-')
            *this = s.substr(1);
        else {
            for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
                v.push_back(*(s.rbegin() + i) - '0');
            zip(0);
        }
        return setsign(s[0] == '-' ? -1 : 1, sign = 1);
    }
    bool operator<(const BigIntTiny &b) const {
        return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) : b.absless(*this));
    }
    bool operator==(const BigIntTiny &b) const { return v == b.v && sign == b.sign; }
```

```
    BigIntTiny &operator+=(const BigIntTiny &b) {
        if (sign != b.sign) return *this = (*this) - -b;
        v.resize(std::max(v.size(), b.v.size()) + 1);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i);
            v[i] = carry % 10000, carry /= 10000;
        }
        return setsign(sign, 0);
    }
    BigIntTiny operator+(const BigIntTiny &b) const {
        BigIntTiny c = *this;
        return c += b;
    }
    void add_mul(const BigIntTiny &b, int mul) {
        v.resize(std::max(v.size(), b.v.size()) + 2);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
            carry += v[i] + b.get_pos(i) * mul;
            v[i] = carry % 10000, carry /= 10000;
        }
    }
    BigIntTiny operator-(const BigIntTiny &b) const {
        if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
        if (sign != b.sign) return (*this) + -b;
        if (absless(b)) return -(b - *this);
        BigIntTiny c;
        for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
            borrow += v[i] - b.get_pos(i);
            c.v.push_back(borrow);
            c.v.back() -= 10000 * (borrow >>= 31);
        }
        return c.setsign(sign, 0);
    }
    BigIntTiny operator*(const BigIntTiny &b) const {
        if (b < *this) return b * *this;
        BigIntTiny c, d = b;
        for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
            c.add_mul(d, v[i]);
        return c.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator/(const BigIntTiny &b) const {
        BigIntTiny c, d;
        BigIntTiny e=b;
        e.sign=1;

        d.v.resize(v.size());
        double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
                           (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
        for (int i = (int)v.size() - 1; i >= 0; i--) {
            c.v.insert(c.v.begin(), v[i]);
            int m = (int)((c.get_pos((int)e.v.size()) * 10000 + c.get_pos((int)e.v.size() - 1)) * db);
            c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
            while (!(c < e))
                c = c - e, d.v[i] += 1;
        }
        return d.setsign(sign * b.sign, 0);
```

```
    }
    BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b * b; }
    bool operator>(const BigIntTiny &b) const { return b < *this; }
    bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
    bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
    bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
};
```

快速幂&矩阵乘法

矩阵乘法中，f 为答案矩阵，是一维长为 n 的 $vector$；a 为转移矩阵，是二维 $n\cdot n$ 的 $vector$

传参数时，直接传入当前的 f 与 a 即可

```
LL ksm(LL a,LL b){
    a%=mod; LL ret=1;
    while(b){
        if(b&1) ret=ret*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ret;
}

void mul(const vector<vector<LL>>& a,vector<LL>& f,int n){//f=a*f
    vector<LL>b(n,0);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            b[i]=(b[i]+a[i][j]*f[j]%mod)%mod;
        }
    }
    swap(f,b);
}
void self_mul(vector<vector<LL>>& a,int n){//a=a*a
    vector b(n,vector<LL>(n,0));
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                b[i][j]=(b[i][j]+a[i][k]*a[k][j]%mod)%mod;
            }
        }
    }
    swap(a,b);
}
```

扩展欧几里得

最终会给出一组满足 $a\cdot x+b\cdot y=g$ 的解

- 迭代法实现

```
int exgcd(int a, int b, int &x, int &y) {
  if (!b) {
    x = 1;
    y = 0;
    return a;
  }
  int d = Exgcd(b, a % b, x, y);
  int t = x;
  x = y;
  y = t - (a / b) * y;
  return d;
}
```

- 非迭代法实现

```
int exgcd(int a, int b, int& x, int& y) {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
```

线性筛欧拉函数

P.S. 以下 N 均为据题意设定好的一个常数

```
bool notP[N]={}; int phi[N],mn_p[N]={1};
vector<int>prime;
void init_prime(const int& n=N){//init [1,n-1]
    phi[1]=1;
    for(int i=2;i<n;i++){
        if(!notP[i]){
            prime.push_back(mn_p[i]=i);
            phi[i]=i-1;
        }
        for(const int& p:prime){
            if(i>(n-1)/p) break;
            notP[i*p]=1,mn_p[i*p]=p;
            if(i%p==0){
                phi[i*p]=phi[i]*p;
                break;
            }
            phi[i*p]=phi[i]*(p-1);
        }
    }
}
```

组合数学

阶乘相关的初始化

这个给模板只用来充数的qwq

```
namespace Fac{//factorial
    LL fc[N]={1},fc_inv[N]={1};
    void init(){
        for(int i=1;i<N;i++){
            fc[i]=fc[i-1]*i%mod;
            fc_inv[i]=fc_inv[i-1]*ksm(i,mod-2)%mod;
        }
    }
    LL F(const int& x){
        return fc[x];
    }
    LL P(const int& x,const int& y){
        return fc[x]*fc_inv[x-y]%mod;
    }
    LL C(const int& x,const int& y){
        return fc[x]*fc_inv[y]%mod*fc_inv[x-y]%mod;
    }
}
```

错位排列

- 递推公式（两种任选即可）

$$D_n = (n-1)\cdot(D_{n-1}+D_{n-2})$$
$$D_n = n\cdot D_{n-1}+(-1)^n$$

其中D 的前几项是 $D_1=0, D_2=1, D_3=9, D_4=44, D_5=265$（OEIS A000166）

- 其他关系

错位排列数有一个向下取整的简单表达式，增长速度与阶乘仅相差常数：

$$D_n = \left\lfloor \frac{n!}{e} \right\rfloor$$

随着元素数量的增加，形成错位排列的概率 P 接近：

$$P = \lim_{n\to\infty}\frac{D_n}{n!} = \frac{1}{e}$$

卡特兰数

H_n = 进栈序列为 $1,2,3,\cdots,n$ 的栈的出栈序列个数，以下是一些常见公式

$$H_n = \frac{\binom{2n}{n}}{n+1}(n\geq 2, n\in\mathbb{N}_+)$$
$$H_n = \begin{cases}\sum_{i=1}^{n} H_{i-1}H_{n-i} & n\geq 2, n\in\mathbb{N}_+\\ 1 & n=0,1\end{cases}$$
$$H_n = \frac{H_{n-1}(4n-2)}{n+1}$$
$$H_n = \binom{2n}{n}-\binom{2n}{n-1}$$

斯特林数

第二类斯特林数

$\begin{Bmatrix} n \\ k \end{Bmatrix}$，$S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数

- 递推公式

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = [n=0]$

- 通项公式

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{i=0}^{m} \frac{(-1)^{m-i} \cdot i^n}{i! \cdot (m-i)!}$$

待补充：同一行/列的第二类斯特林数的计算

第一类斯特林数

$\begin{bmatrix} n \\ k \end{bmatrix}$，$s(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空轮换的方案数

一个轮换即一个首尾相接的环形排列，如 $[A,B,C,D]=[B,C,D,A]=[C,D,A,B]=[D,A,B,C]$，即两个可以通过旋转而相互得到的轮换是等价的。但是，翻转不算旋转，如 $[A,B,C,D] \neq [D,C,B,A]$

- 递推公式

$$\begin{bmatrix} n \\ m \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$$

边界是 $\begin{bmatrix} n \\ 0 \end{bmatrix} = [n=0]$

- 通项公式

没有

待补充：同一行/列的第一类斯特林数的计算

杂项

火车头

娱乐用，用来平时做题凹卡常题

```
#pragma GCC target("avx")
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
```

```
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

写在最后（来自Benq大神的几句话）

```
stuff you should look for
int overflow, array bounds
special cases (n=1?)
do smth instead of nothing and stay organized
WRITE STUFF DOWN
DON'T GET STUCK ON ONE APPROACH
```