ACM Templates

2023/09/17

@342cf07

动态规划

1. 背包DP

```
// 多重背包
for (int i = 1; i <= n; i++)
{
    int q = a[i].m;
    for (int j = 1; q; j *= 2)
    {
        if (j > q)
        {
            j = q;
        }
        q -= j;
        for (int k = w; k >= j * a[i].w; k--)
        {
            f[k] = max(f[k], f[k - j * a[i].w] + j * a[i].v);
        }
    }
}
```

二进制分组优化

```
index = 0;
for (int i = 1; i <= m; i++)
{
    int c = 1, p, h, k;
    cin >> p >> h >> k;
    while (k > c)
    {
        k -= c;
        list[++index].w = c * p;
        list[index].v = c * h;
        c *= 2;
    }
    list[++index].w = p * k;
    list[index].v = h * k;
}
```

2. 状态压缩DP

```
int cnt[1024];
int dp[40][1024][90];
int can[2000], num = 0;
int S = 1 << n;
for (int s = 0; s < S; s++)
{
    if ((s << 1) & s)
    {
        continue;
    }
    can[++num] = s;
    for (int j = 0; j < n; j++)
    {
        if ((s >> j) & 1)
        {
            cnt[num]++;
        }
    }
    dp[1][num][cnt[num]] = 1;
}
for (int i = 2; i <= n; i++)
{
    for (int j = 1; j <= num; j++)
    {
        int x = can[j];
        for (int p = 1; p <= num; p++)
        {
            int y = can[p];
            if ((y & x) || ((y << 1) & x) || ((y >> 1) & x))
                continue;
            for (int l = 0; l <= k; l++)
            {
                dp[i][j][cnt[j] + l] += dp[i - 1][p][l];
            }
        }
    }
}
```

3. 数位DP

现在问有多少的数比12345小

1.关于前导0：00999→999，09999→9999

2.关于limit前面的数是否紧贴上限

如果前面的数是紧贴上限的，当前这位枚举的上限便是当前数的上限

如果前面的数不是紧贴上限的，当前这位枚举的上限便是 9

3.关于DP维度

一般来说，DFS有几个状态，DP就几个维度

比如现在DP就是DP [pos] [limt] [zero]

4.关于记忆化DP

现在枚举到了 10××× 和 11×××

显然 这两种状态后面的×××状态数是一样的

重点：dp[pos][limit][zero]表示前面的数枚举状态确定，后面的数有多少种可能

5.关于DP细节

一般来说我们一开始都memset(dp,-1,sizeof(dp))

如果dp[pos][limt][zero]!=-1 return dp[pos][limit][zero];

6.关于初始化：

一开始 limit 是1，表示一开始的数只能选 1~a[1]

一开始zero 是1，假定表示前面的数全为0

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define mp make_pair
#define pb push_back  // vector函数
#define popb pop_back // vector函数
#define fi first
#define se second
const int N = 20;
// const int M=;
// const int inf=0x3f3f3f3f;     //一般为int赋最大值,不用于memset中
// const ll INF=0x3ffffffffffff; //一般为ll赋最大值,不用于memset中
int T, n, len, a[N], dp[N][2][2];
inline int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
    {
        x = (x << 1) + (x << 3) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}
int dfs(int pos, bool lim, bool zero)
{
    if (pos > len)
        return 1;
    if (dp[pos][lim][zero] != -1)
        return dp[pos][lim][zero];
    int res = 0, num = lim ? a[pos] : 9;
    for (int i = 0; i <= num; i++)
        res += dfs(pos + 1, lim && i == num, zero && i == 0);
    return dp[pos][lim][zero] = res;
```

```cpp
}
int solve(int x)
{
    len = 0;
    memset(dp, -1, sizeof(dp));
    for (; x; x /= 10)
        a[++len] = x % 10;
    reverse(a + 1, a + len + 1);
    return dfs(1, 1, 1);
}
int main()
{
    int l = read(), r = read();
    printf("%d\n", solve(r) - solve(l - 1));
    return 0;
}
```

快读

```cpp
inline int read()
{
    int x = 0, w = 1;
    char ch = 0;
    while (ch < '0' || ch > '9')
    {
        ch = getchar();
        if (ch == '-')
        {
            w = -1;
        }
    }
    while (ch >= '0' && ch <= '9')
    {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * w;
}
```

数据结构

1.树状数组

```
class BIT
{
    int n = 2e6;
    long long *a;

  public:
    BIT(int size) : n(size)
    {
        a = new long long[size + 10];
    }
    void update(int p, long long x)
    {
        while (p <= n)
            a[p] += x, p += (p & (-p));
    }

    long long query(int l, int r)
    {
        long long ret = 0;
        l--;
        while (r > 0)
            ret += a[r], r -= (r & (-r));
        while (l > 0)
            ret -= a[l], l -= (l & (-l));
        return ret;
    }
};
```

2. 并查集

```
#include <cassert>
#include <iostream>
#include <set>
/* Last modified: 23/08/01 */
class DSU
{
  private:
    int *f;
    int size;

  public:
    DSU(int size) : size(size)
    {
        assert(size > 1);
        f = new int[size + 10];
        for (int i = 1; i <= size; i++)
            f[i] = i;
    }
    int find(int x)
    {
        return f[x] == x ? x : (f[x] = find(f[x]));
    };
    bool same(int x, int y)
    {
        return find(x) == find(y);
    };
    bool merge(int x, int y)
    {
        int fx = find(x), fy = find(y);
        return ((fx != fy) ? f[fx] = fy : false);
    };
    int count()
    {
        std::set<int> s;
        for (int i = 1; i <= size; i++)
            s.insert(find(i));
        return s.size();
    }
};
```

3. ST表

```
// log2(x) 的预处理
// 1. 递推
lg[2] = 1;
for (int i = 3; i < N; i++)
    lg[i] = lg[i / 2] + 1;
// 2. 基于编译期计算
using std::array;
// WARNING: LOG_SIZE may cause CE if too big.
const int LOG_SIZE = 1e5 + 10;
constexpr array<int, LOG_SIZE> LOG = []() {
    array<int, LOG_SIZE> l{0, 0, 1};
    for (int i = 3; i < LOG_SIZE; i++)
        l[i] = l[i / 2] + 1;
    return l;
}();
// 3. 直接计算
int lg(int x)
{
    return 31 - __builtin_clz(x);
}
// STL 提供了 std::lg(), 底数是e.
```

```
class SparseTable
{
  private:
    // SIZE depends on range of f[i][0].
    // 22 is suitable for 1e5.
    static const int SIZE = 22;
    // f[i][j] maintains the result from i to i + 2 ^ j - 1;
    int (*f)[SIZE];
    using func = std::function<int(int, int)>;
    func op;
    // length of f from 1 to l;
    int l;

  public:
    SparseTable(int a[][SIZE], func foo, int len) : f(a), op(foo), l(len)
    {
        for (int j = 1; j < SIZE; j++)
            for (int i = 1; i + (1 << j) - 1 <= len; i++)
                // f[i][j] comes from f[i][j - 1].
                // f[i][j - 1], f[i + 2^(j - 1)] cover the range of f[i][j].
                f[i][j] = foo(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
    };
    int query(int x, int y)
    {
        int s = LOG[y - x + 1];
        return op(f[x][s], f[y - (1 << s) + 1][s]);
    }
};
```

4. 线段树

```
#include <bits/stdc++.h>

using namespace std;
#define IOS ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);
#define int long long
#define ull unsigned long long
#define lowbit(i) ((i) & (-i))
#define ls(p) (p << 1)
#define rs(p) (p << 1 | 1)
#define rep(i, a, b) for (int i = a; i <= b; i++)
#define per(i, a, b) for (int i = a, i >= b, i--)

typedef pair<int, int> PII;
const int mod = 1e9 + 7;
const int inf = 0x3f3f3f3f;
const int N = 1e5 + 200;
int qpow(int a, int n)
{
    int ans = 1;
    while (n)
    {
        if (n & 1)
        {
            ans = ans * a % mod;
        }
        a = a * a % mod;
        n >>= 1;
    }
    return ans;
}
int a[N];
int tag[4 * N];
int tree[4 * N];
int n;
void push_up(int p)
{
    tree[p] = tree[ls(p)] + tree[rs(p)];
}
void build(int p, int l, int r)
{
    if (l == r)
    {
        tree[p] = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(ls(p), l, mid);
    build(rs(p), mid + 1, r);
    push_up(p);
}
void push_down(int p, int l, int r)
{
    int mid = (l + r) >> 1;
    tag[ls(p)] += tag[p];
    tag[rs(p)] += tag[p];
    tree[ls(p)] += tag[p] * (mid - l + 1);
    tree[rs(p)] += tag[p] * (r - mid);
    tag[p] = 0;
}
void update(int nl, int nr, int k, int p = 1, int l = 1, int r = n)
{
    if (nl <= l && r <= nr)
    {
        tag[p] += k;
        tree[p] += k * (r - l + 1);
        return;
    }
    push_down(p, l, r);
    int mid = (l + r) >> 1;
    if (nl <= mid)
        update(nl, nr, k, ls(p), l, mid);
    if (nr > mid)
```

```
        update(nl, nr, k, rs(p), mid + 1, r);
    push_up(p);
}
int query(int x, int y, int l = 1, int r = n, int p = 1)
{
    int res = 0;
    if (x <= l && y >= r)
        return tree[p];
    int mid = (l + r) >> 1;
    push_down(p, l, r);
    if (x <= mid)
        res += query(x, y, l, mid, ls(p));
    if (y > mid)
        res += query(x, y, mid + 1, r, rs(p));
    return res;
}
signed main()
{
    IOS int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    build(1, 1, n);
    while (q--)
    {
        int op, x, y, k;
        cin >> op;
        if (op == 1)
        {
            cin >> x >> y >> k;
            update(x, y, k);
        }
        else
        {
            cin >> x >> y;
            cout << query(x, y) << endl;
        }
    }
    return 0;
}
```

数学

1. 快速幂

```
int qpow(int a,int n)
{
    int ans=1;
    while(n)
    {
        if(n&1)
        {
            ans =ans*a%mod;
        }
        a=a*a%mod;
        n>>=1;

    }
    return ans;
}
```

2. exgcd

```
int exgcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y),x0=x,y0=y;
    x=y0;
    y=x0-(a/b)*y0;
    return d;
}
```

3.线性inv

```
void getinv(int n)
{
    inv[1]=1;
    for(int i=2;i<=n;i++)
    {
        inv[i]=mod-((mod/i)*inv[mod%i])%mod;
    }
}
```

4.数论分块

```
int ans=0;
    for(int l=1,r;l<=n;l=r+1)
    {
        r=n/(n/l);
        ans+=(r-l+1)*(n/l);
    }
    cout<<ans<<endl;
```

5.欧拉筛

```
int Eular(int n) {
    int cnt = 0;
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for(int i=2;i<=n;i++)
    {
        if(is_prime[i])
        {
            prime[++cnt]=i;
        }
        for(int j=1;j<=cnt&&i*prime[j]<=n;j++)
        {
            is_prime[i*prime[j]]=0;
            if(i%prime[j]==0)break;
        }
    }
    return cnt;
}
```

6.欧拉函数

```
int Eular(int n) {
    int cnt = 0;
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for(int i=2;i<=n;i++)
    {
        if(is_prime[i])
        {
            prime[++cnt]=i;
        }
        for(int j=1;j<=cnt&&i*prime[j]<=n;j++)
        {
            is_prime[i*prime[j]]=0;
            if(i%prime[j]==0)break;
        }
    }
    return cnt;
}
```

7.组合数

```
int fac[N];
int  inv[N];
void init(int n)
{
    fac[0] = 1;
    inv[0] = 1;
    inv[1] = 1;
    fac[1] = 1;
    for(int i = 2;i<=2*n;i++)
    {
        fac[i] = fac[i-1]*i%mod;
        inv[i] = (mod-mod/i)*inv[mod%i]%mod;
    }
    for(int i = 1;i<=n;i++)
    {
        inv[i] = inv[i]*inv[i-1]%mod;
    }
}
int C(int n,int m)
{
    if(m>n||m<0||n<0)return 0;
    return fac[n]*inv[m]%mod*inv[n-m]%mod;
}
```

8.欧拉定理

a在mod m意义下，

a$^{(b}$c) 与 a^(b mod(eular(m))+m)同余

9.卡特兰数

```
int C(int n,int m)
{
    return fac[n]*qpow(fac[n-m],mod-2)%mod*qpow(fac[m],mod-2)%mod;
}
int cat(int n)
{
    return C(2*n,n)*qpow(n+1,mod-2)%mod;
}
```

10. 矩阵

```
    class matrix
{
    public:
        int x[105][105];
        int sz;
        matrix(int n)
        {sz=n;
```

```
            for(int i=1;i<=sz;i++)
            {
                for(int j=1;j<=sz;j++)
                {
                    x[i][j]=0;
                }
            }
        }
        matrix mul(matrix a,matrix b);
        matrix qpow(matrix a,int n);
        void tra(matrix a);
};

matrix matrix::mul(matrix a, matrix b) {
    matrix c(a.sz);
    for(int i=1;i<=a.sz;i++)
        for(int j=1;j<=a.sz;j++)
            for(int k=1;k<=a.sz;k++)
                c.x[i][j]=(c.x[i][j]%mod+(a.x[i][k]*b.x[k][j])%mod)%mod;
    return c;
}
matrix matrix::qpow(matrix a,int n)
{
    matrix res(a.sz);
    for(int i=1;i<=a.sz;i++)res.x[i][i]=1;
    while(n>0)
    {
        if(n&1)res= mul(res,a);
        a= mul(a,a);
        n>>=1;
    }
    return res;
}
void matrix::tra(matrix a) {
    for(int i=1;i<=a.sz;i++)
    {
        for(int j=1;j<=a.sz;j++)
        {
            cout<<a.x[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

图论

1. 最短路

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int inf = 0x3f3f3f3f;
typedef pair<int,int> PII;
vector<PII> mp[100100];
int n,m,s;
int dis[100100];
int vis[100100];
priority_queue<PII,vector<PII>,greater<PII> > q;

void dj(int s)
{
    for(int i=1;i<=n;i++)dis[i]=inf;
    dis[s]=0ll;
    q.push({dis[s],s});

    while(!q.empty())
    {
        int u=q.top().second;
        q.pop();
        if(vis[u])continue;
        vis[u]=1;
        for(auto [w,v]:mp[u])
        {
            if(dis[u]+w<dis[v])
            {
                dis[v]=dis[u]+w;
                q.push({dis[v],v});
            }
        }
    }
}
signed main() {
cin>>n>>m>>s;
for(int i=1;i<=m;i++)
{
    int u,v,w;
    cin>>u>>v>>w;
    mp[u].push_back({w,v});
}
dj(s);
for(int i=1;i<=n;i++)cout<<dis[i]<<" ";
```

```
    return 0;
}
```

2. 并查集

```
int fa[100100];
void init(int n)
{
    for(int i=1;i<=n;i++)fa[i]=i;
}
int find(int x)
{
    if(fa[x]==x)
        return x;
    fa[x]=find(fa[x]);
    return find(fa[x]);
}
void merge(int x,int y)
{
    fa[find(x)]=find(y);
}
```

3. 生成树
   prim

```
   #include <bits/stdc++.h>
using namespace std;
const int N = 5050;
const int inf=0x3f3f;
int g[N][N],dis[200200];
bool vis[200200];
int n,m,ans,u,v,w;
void init(){
    memset(g,inf,sizeof(g));
    memset(dis,inf,sizeof(dis));
}
void addedge(int u,int v,int w)
{
    if (u != v && g[u][v] > w) g[u][v] = g[v][u] = w;
}//存图
void prim(){
    dis[1]=0;
    for(int i=1;i<=n;i++){
        int t=0;
        for(int j=1;j<=n;j++){
            if(!vis[j]&&dis[j]<dis[t])t=j;
        }
        vis[t]=1;
        ans+=dis[t];
        for(int j=1;j<=n;j++){
            if(!vis[j]&&dis[j]>g[t][j]){
                    dis[j]=g[t][j];
                }
            }
        }
}

signed main() {
cin>>n>>m;
init();
for(int i=1;i<=m;i++){
    cin>>u>>v>>w;
    addedge(u,v,w);
    addedge(u,v,w);//无向图
}
prim();
for(int i=1;i<=n;i++){
    if(!vis[i]) {
        cout << "orz";
        return 0;
    }
}
cout<<ans;
    return 0;
}
```

kruskal

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int fa[200100];//并查集
struct edge{
    int u,v,w;
}e[200100],mst[200100];
int n,m,k;//图的节点数n,边数m,树的边数k
int ans;
bool cmp(edge a,edge b)
{
    return a.w<b.w;
}
void init(int n)
{
    for(int i=1;i<=n;i++)
    fa[i]=i;
}
int find(int x)
{
    if(fa[x]==x)
        return x;
    else{
    fa[x]=find(fa[x]);
    return find(fa[x]);
    }
}
void merge(int i,int j)
{
    fa[find(i)]=find(j);
}
void kruskal(){
    for(int i=1;i<=m;i++){
        if(find(e[i].u)!=find(e[i].v)){
            k++;
            mst[i].u=e[i].u;
            mst[i].v=e[i].v;
            mst[i].w=e[i].w;
            ans+=e[i].w;
            merge(e[i].u,e[i].v);
        }
    }
}
signed main()
{
 std::ios::sync_with_stdio(false);
    std::cin.tie(0);
cin>>n>>m;
init(n);
for(int i=1;i<=m;i++)cin>>e[i].u>>e[i].v>>e[i].w;
sort(e+1,e+m+1,cmp);
kruskal();
if(k==n-1)cout<<ans;
else cout<<"orz";
    return 0;
}
```

4. lca

```
   #include <bits/stdc++.h>

using namespace std;
#define IOS ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr);
#define int long long
#define ull unsigned long long
#define lowbit(i) ((i) & (-i))
#define ls(p) (p << 1)
#define rs(p) (p << 1 | 1)
#define rep(i, a, b) for (int i = a; i <= b; i++)
#define per(i, a, b) for (int i = a; i >= b; i--)

typedef pair<int, int> PII;
const int mod = 1e9 + 7;
const int inf = 0x3f3f3f3f;
const int N = 5e5 + 200;
```

```
int qpow(int a, int n)
{
    int ans = 1;
    while (n)
    {
        if (n & 1)
        {
            ans = ans * a % mod;
        }
        a = a * a % mod;
        n >>= 1;
    }
    return ans;
}
// lca模板题:
int n, q, root; // n个节点,q次查询，root为根节点
vector<int> mp[N];
int lg2[N];
int dep[N];
int f[N][20];
int vis[N];
void dfs(int u, int fa = 0)
{
    if (vis[u])
        return;
    vis[u] = 1;
    dep[u] = dep[fa] + 1;
    f[u][0] = fa;
    for (int i = 1; i <= lg2[dep[u]]; i++)
    {
        f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for (auto v : mp[u])
    {
        dfs(v, u);
    }
}
int lca(int a, int b)
{
    if (dep[a] > dep[b])
        swap(a, b);
    while (dep[a] != dep[b])
        b = f[b][lg2[dep[b] - dep[a]]];
    if (a == b)
        return a;
    for (int k = lg2[dep[a]]; k >= 0; k--)
    {
        if (f[a][k] != f[b][k])
        {
            a = f[a][k], b = f[b][k];
        }
    }
    return f[a][0];
}
signed main()
{
    IOS cin >> n >> q >> root;
    for (int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        mp[u].push_back(v);
        mp[v].push_back(u);
    }
    for (int i = 2; i <= n; i++)
    {
        lg2[i] = lg2[i / 2] + 1;
    } // 预处理log2
    dfs(root);
    while (q--)
    {
        int u, v;
        cin >> u >> v;
        cout << lca(u, v) << endl;
    }
    return 0;
}
```

字符串

1. KMP

```
int f[N];
void kmp(string s, string p)
{
    p += '@';
    p += s;
    for (int i = 1; i < p.size(); i++)
    {
        int j = f[i - 1];
        while (j && p[j] != p[i])
            j = f[j - 1];
        if (p[j] == p[i])
            f[i] = j + 1;
    }
}
```

2. Manacher

```
int p[N];
string s0, s = "@#";

int main()
{
    cin >> s0;
    for (auto i : s0)
        s += i, s += "#";#include <cstdio>
using namespace std;
using ll = long long;
// !!! N = n * 2, because you need to insert '#' !!!
const int N = 3e7;
#define min(A, B) ((A > B) ? B : A)
// p[i]: range of the palindrome i-centered.
int p[N];
// s: the string.
char s[N] = "@#";
// l: length of s.
int l = 2;

int main()
{
    char tmp = getchar();
    while (tmp > 'z' || tmp < 'a')
        tmp = getchar();
    while (tmp <= 'z' && tmp >= 'a')
        s[l++] = tmp, s[l++] = '#', tmp = getchar();
    /*<--- input & preparation --->*/
    int m = 0, r = 0;
    ll ans = 0;
    for (int i = 1; i < l; i++)
    {
        // evaluate p[i]
        if (i <= r)
            p[i] = min(p[m * 2 - i], r - i + 1);
        else
            p[i] = 1;
        // brute force!
        while (s[i - p[i]] == s[i + p[i]])
            ++p[i];
        // maintain m, r
        if (i + p[i] > r)
        {
            r = i + p[i] - 1;
            m = i;
        }
        // find the longest p[i]
        if (p[i] > ans)
            ans = p[i];
    }
    printf("%lld", ans - 1);

    return 0;
}
    int m = 0, r = 0;
    ll ans = 0;
    int l = s.size();
    for (int i = 1; i < l; i++)
    {
        if (i <= r)
            p[i] = min(p[m * 2 - i], r - i + 1);
        else
            p[i] = 1;
        while (s[i - p[i]] == s[i + p[i]])
            ++p[i];
        if (i + p[i] > r)
        {
            r = i + p[i] - 1;
            m = i;
        }
        if (p[i] > ans)
            ans = p[i];
    }
}
```

3. hash

随机素数表

42737, 46411, 50101, 52627, 54577, 191677, 194869, 210407, 221831, 241337, 578603, 625409, 713569, 788813, 862481, 2174729, 2326673, 2688877, 2779417, 3133583, 4489747, 6697841, 6791471, 6878533, 7883129, 9124553, 10415371, 11134633, 12214801, 15589333, 17148757, 17997457, 20278487, 27256133, 28678757, 38206199, 41337119, 47422547, 48543479, 52834961, 76993291, 85852231, 95217823, 108755593, 132972461, 171863609, 173629837, 176939899, 207808351, 227218703, 306112619, 311809637, 322711981, 330806107, 345593317, 345887293, 362838523, 373523729, 394207349, 409580177, 437359931, 483577261, 490845269, 512059357, 534387017, 698987533, 764016151, 906097321, 914067307, 954169327

1572869, 3145739, 6291469, 12582917, 25165843, 50331653 （适合哈希的素数）

```
#include <string>
using ull = unsigned long long;
using std::string;
const int mod = 998244353;

int n;
ull Hash[2000200]; // 自然溢出法用unsigned类型
ull RHash[2000200];
ull base[2000200];
void init()
{
    base[0] = 1;
    for (int i = 1; i <= 2000010; i++)
    {
        base[i] = base[i - 1] * 131 % mod;
    }
}
void get_hash(string s)
{
    for (int i = 1; i <= (int)s.size(); i++)
    {
        Hash[i] = Hash[i - 1] * base[1] % mod + s[i - 1];
        Hash[i] %= mod;
    }
}
void get_Rhash(string s)
{
    for (int i = (int)s.size(); i >= 1; i--)
    {
        RHash[s.size() - i + 1] = RHash[s.size() - i] * base[1] % mod + s[i - 1];
        RHash[i] %= mod;
    }
}
ull getR(int l, int r)
{
    if (l > r)
        return 0;
    return (RHash[r] - (RHash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
}
ull get(int l, int r)
{
    if (l > r)
        return 0;
    return (Hash[r] - (Hash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
}
```

4. 字典树

```
#include <string>
using std::string;
/* Last modified: 23/07/03 */
// Trie for string and prefix
class Trie
{

    static const int trie_tot_size = 1e5;
    // trie_node_size: modify if get() is modified.
    static const int trie_node_size = 64;
    int tot = 0;
    // end: reserved for count
    const int end = 63;
    int (*nxt)[trie_node_size];

  public:
    Trie()
    {
        nxt = new (int[trie_tot_size][trie_node_size]);
    }
    int get(char x)
    {
        // modify if x is in certain range, assuming 0-9 or a-z.
        if (x >= 'A' && x <= 'Z')
            return x - 'A';
        else if (x >= 'a' && x <= 'z')
            return x - 'a' + 26;
        else
            return x - '0' + 52;
    }
    int find(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            cnt = nxt[cnt][get(i)];
            if (!cnt)
                return 0;
        }
        return cnt;
    }
    void insert(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            auto j = get(i);
            // count how many strings went by
            nxt[cnt][end]++;
            if (nxt[cnt][j] > 0)
                // character i already exists.
                cnt = nxt[cnt][j];
            else
            {
                // doesn't exist, new node.
                nxt[cnt][j] = ++tot;
                cnt = tot;
            }
        }
        nxt[cnt][end]++;
    }
    int count(string s)
    {
        int cnt = find(s);
        if (!cnt)
            return 0;
        return nxt[cnt][end];
    }
    void clear()
    {
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j <= end; j++)
                nxt[i][j] = 0;
        tot = 0;
    }
};
```

5.AC自动机

```
#include <queue>
#include <string.h>
#include <string>
using std::string, std::queue;

struct AC_automaton
{
    static const int _N = 1e6;

    int (*trie)[27];
    int tot = 0;
    int *fail;
    int *e;
    AC_automaton()
    {
        trie = new int[_N][27];
        fail = new int[_N];
        e = new int[_N];
        memset(trie, 0, sizeof(trie));
        memset(fail, 0, sizeof(fail));
        memset(e, 0, sizeof(e));
    }

    void insert(string s)
    {
        int now = 0;
        for (auto i : s)
            if (trie[now][i - 'a'])
                now = trie[now][i - 'a'];
            else
                trie[now][i - 'a'] = ++tot, now = tot;
        e[now]++;
    }

    void build()
    {
        queue<int> q;
        for (int i = 0; i < 26; i++)
            if (trie[0][i])
                q.push(trie[0][i]);
        while (q.size())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (trie[u][i])
                    fail[trie[u][i]] = trie[fail[u]][i], q.push(trie[u][i]);
                else
                    trie[u][i] = trie[fail[u]][i];
        }
    }

    int query(string t)
    {
        int u = 0, res = 0;
        for (auto c : t)
        {
            u = trie[u][c - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j])
                res += e[j], e[j] = -1;
        }
        return res;
    }
};
```

心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小 （2倍，4倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序，初始化或者询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 不要把 int 开成 char
- 清零的时候全部用 0~n+1。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 `l + (r - l) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化
- 分块一定要特判在同一块中的情况