1.图论

1.1图的存储

常用的有邻接表和链式前向星两种存图方式。

```
//-----------------//邻接表可以比较灵活
#include<functional>
int n, m;
int main()
{
    cin >> n >> m;
    vector<vector<int>>g(n + 1);
    for (int i = 1; i <= m; i++)
    {
        int u, v; cin >> u >> v;
        g[i].push_back(v);
    }
    function<void(int)>dfs = [&](int u)//仿函数
    {
        for (auto v : g[u])
            dfs(v);
    };
    dfs(1);
    return 0;
}
```

```cpp
//----------------//
const int N = 1e5 + 5;
vector<int>g[N];
int n, m;
void dfs(int u)
{
    for (auto v : g[u])
        dfs(v);
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
    }
    dfs(1);
    return 0;
}
//----------------//
const int N = 1e5 + 5;
int h[N], tot, to[N], nxt[N];
void add(int u, int v) { to[tot] = v; nxt[tot] = h[u];
h[u] = tot++; }
int n, m;
void dfs(int u)
{
    for (int i = h[u]; ~i; i = nxt[i])
    {
        int v = to[i];
        dfs(v);
    }
}
int main()
{
    memset(h, -1, sizeof h);
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
```

```
    {
        int u, v; cin >> u >> v;
        add(u, v);
    }
    return 0;
}
```

1.2拓扑序列

- 拓扑排序
  可用于检验是否有环，是否为链，或检验是否为确定拓扑序。

```
int n, m;
int main()
{
    cin >> n >> m;
    vector<vector<int>>g(n+1);
    vector<int>in(n);//入度
    for (int i = 1; i <= m; i++)
    {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        in[v]++;
    }
    function<void(void)>topo = [&]()
    {
        //小/大根堆求字典序最小/大拓扑序
        priority_queue<int,vector<int>,greater<int>>Q;
        vector<int>L;
        for (int i = 1; i <= n; i++)
            if (in[i] == 0)
                Q.push(i);//
        while (!Q.empty())
        {
            int u = Q.top(); Q.pop();
            L.push_back(u);//区别在于每次只在队列中加入一个数
            for (int v : g[u])
                if (--in[v] == 0)
                    Q.push(v);
```

```
        }
        if (L.size() == n)
        {
            for (int u : L)cout << u << ' ';
            cout << '\n';
        }
        else cout << "NO\n";
    };
    topo();
    return 0;
}
```

1.3最短路

a.dijkstra

- 求非负权图的单源最短路。
  简单应用：记录路径，求最短路径数。

```
#include<iostream>
#include<vector>
#include<cstring>
#include<queue>
#include<stack>
#define pii pair<int,int>
using namespace std;
const int inf = 1e9;
const int mod = 1e5 + 3;
const int N = 1e6 + 5;

int sum[N];//记录最短路径数
int pre[N];//记录前驱

int n, m, s;
vector<pii>g[N << 1];

int dis[N], vis[N];
vector<int>L[N];

```

```cpp
priority_queue<pii, vector<pii> , greater<pii>>Q;//fs是w，se是u
void dij()
{
    for (int i = 1; i <= n; i++)dis[i] = inf;
    dis[s] = 0;

    for (int i = 1; i <= n; i++)pre[i] = -1;//前驱标记为-1
    sum[s] = 1;

    Q.push({ 0,s });
    while (!Q.empty())
    {
        pair<int, int>x = Q.top(); Q.pop();
        int u = x.second;

        if (vis[u])continue;
        vis[u] = 1;

        for (auto to : g[u])
        {
            int v = to.first, w = to.second;//fs是u，se是w
            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                Q.push({ dis[v],v });

                sum[v] = sum[u];
                pre[v] = u;
            }
            else if (dis[v] == dis[u] + w)
                sum[v] = (sum[v] + sum[u]) % mod;
        }
    }
}
void path(int u)
{
    //用栈逆序输出，或者递归输出
    stack<int>s;
```

```
    do
    {
        s.push(u);
        u = pre[u];
    } while (u != -1);
    while (!s.empty())
    {
        cout << s.top() << ' ';
        s.pop();
    }
    cout << '\n';
}

int main()
{
    cin >> n >> m >> s;
    for (int i = 1; i <= m; i++)
    {
        int u, v, w; cin >> u >> v >> w;
        g[u].push_back({ v,w });
        g[v].push_back({ u,w });
    }
    dij();
    for (int i = 1; i <= n; i++)path(i);
    return 0;
}
```

b.floyd

- 求多源最短路。
  简单应用：图的传递闭包，记录最短路径。

```
#include<iostream>
#include<stack>
using namespace std;
const int N = 1e3 + 5;
const int inf = 1e9;
int n, m, s;
int dis[N][N];
```

```cpp
int pre[N][N];
void path(int s, int t)
{
    if (dis[s][t] == inf)
    {
        cout << "none";
        return;
    }
    stack<int>sk;
    do
    {
        sk.push(t);
        t = pre[s][t];
    } while (t != -1);
    sk.push(s);
    while (!sk.empty())
    {
        cout << sk.top() << ' ';
        sk.pop();
    }
    cout << '\n';
}
int main()
{
    cin >> n >> m >> s;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dis[i][j] = i == j ? 0 : inf,
            pre[i][j] = -1;//初始化
    for (int i = 1; i <= m; i++)
    {
        int u, v, w; cin >> u >> v >> w;
        dis[u][v] = w;
    }
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (dis[i][j] > dis[i][k] + dis[k][j])
                    dis[i][j] = dis[i][k] + dis[k][j],
```

```
                pre[i][j] = k;//记录前驱
    //dis[i][j] |= dis[i][k] & dis[k][j];//传递闭包
    path(1, 4);
    return 0;
}
```

1.4树上问题

a.拓扑序数

- 求树或森林的拓扑序数公式:

$$dp[u]=(\prod dp[v])*\frac{(sz[u]-1)!}{\prod(sz[v]!)}=(sz[u]-1)!*\prod\frac{dp[v]}{sz[v]!}$$

$$n!*\prod_{i=1}^{i=n}\frac{1}{sz[i]}$$

b.LCA

- 基于倍增

```
int n;
vector<int>g[N];
///LCA
int dep[N], fa[N][22];
void dfs(int u, int pre)
{//预处理深度信息和树上st表
    dep[u] = dep[pre] + 1;
    fa[u][0] = pre;//当前节点的第2^0=1个节点是它的父亲节点
    //当前节点的第2^i个节点是它的第2^i-1个节点的第2^i-1个节点
    for (int i = 1; (1 << i) <= dep[u]; i++)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (auto v : g[u])
        if (v != pre)//无向边
            dfs(v, u);
}
int LCA(int x, int y)
```

```
{
    if (dep[x] < dep[y])swap(x, y);//保证x的深度大于y的深度
    for (int i = 20; i >= 0; i--)
    {//x，y跳到同一深度
        if (dep[fa[x][i]] >= dep[y])
            x = fa[x][i];
        if (x == y)return x;//特判
    }
    for (int i = 20; i >= 0; i--)//同时跳
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];//最后LCA是它们的父亲节点
}
int main()
{
    cin >> n;
    for (int i = 1; i <= n - 1; i++)
    {
        int u, v; cin >> u >> v;
        g[u].push_back(v);g[v].push_back(u);
    }
    dfs(1, 0);
    return 0;
}
```

c.树上差分

- 疑惑：在差分数组上求和或在新数组上求和对答案的影响。
- 树上差分又分为**点差分**和**边差分**：
  - 点差分：
    - 路径差分：
      对于路径u到v的点权值加x，对差分数组的操作为：$diff[u]+=x, diff[v]+=x, diff[o]-=x, diff[p]-=x$，其中，o为$u$和$v$的LCA，$p$为$o$的父节点。对于LCA是根节点的特殊情况，相当于只修改u节点的差分数组。最后**从下往上**dfs求前缀和数组即可。

- 子树差分：
  对以u为根节点的子树权值全部加x，对差分数组的操作为：$sdiff[u]+=x$。最后从上往下dfs求前缀和数组即可。
- 题意：对一棵完全二叉树给定4个操作，操作1为将以x为根节点的子树所有点权值加1，操作3为将x到根节点路径上的点权值加1.

```cpp
#include<iostream>
using namespace std;
const int N = 1e7 + 5;
int n, m;
int s[N], r[N], g;//分别为子树差分和路径差分//g是对整棵树操作
int ssum[N], rsum[N];
int ans[N];

void dfs(int k, int fa)
{
    if (k > n)return;
    ssum[k] = s[k] + ssum[fa];
    dfs(k + k, k);
    dfs(k + k + 1, k);
    rsum[k] = r[k] + rsum[k + k] + rsum[k + k + 1];
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int op, x; cin >> op >> x;
        if (op == 1)
            s[x]++;
        else if (op == 2)
            s[x]--, g++;
        else if (op == 3)
            r[x]++;
        else
            r[x]--, g++;
    }
    dfs(1, -1);

```

```
    for (int i = 1; i <= n; i++)ans[ssum[i] + rsum[i] + g]++;
    for (int i = 0; i <= m; i++)cout << ans[i] << ' ';
    return 0;
}
```

- 边差分:
  将路径u到u的边的权值加x，对差分数组的操作为:
  $diff[u]+=x, diff[v]+=x, diff[o]-=2x$，其中$o$为$u$和$v$的LCA。最后从下往上dfs求差分数组的前缀和，$diff[v]$的值即为$u$到$v$所在边的权值变化量。
- 题意：求边被覆盖的次数，按边的输入顺序输出。

```
#include<iostream>
#include<vector>
using namespace std;
const int N = 2e5 + 5;
int n, k;
int u[N], v[N];
vector<int>g[N];
//LCA
int dep[N], fa[N][22], son[N];
int diff[N];
void get(int u, int fa)
{
    for (int v : g[u])
    {
        if (v == fa)continue;
        get(v, u);
        diff[u] += diff[v];
    }
}
int main()
{
    cin >> n;
    for (int i = 1; i <= n - 1; i++)
    {
        cin >> u[i] >> v[i];

```

```
        g[u[i]].push_back(v[i]),
g[v[i]].push_back(u[i]);
    }
    dfs(1, -1);
    cin >> k;
    for (int i = 1; i <= k; i++)
    {
        int u, v; cin >> u >> v;
        int o = lca(u, v);
        diff[u]++, diff[v]++;
        diff[o] -= 2;
    }
    get(1, -1);
    for (int i = 1; i <= n - 1; i++)
    {//根据儿子大小判断
        if (son[u[i]] < son[v[i]])cout << diff[u[i]] <<
' ';
        else cout << diff[v[i]] << ' ';
    }
    return 0;
}
```

1.5最小生成树

- prim->dijkstral

```
int n, m;
vector<pii>g[N];

priority_queue<pii, vector<pii>, greater<pii>>Q;
///Q的fs为w，sec为now节点
int vis[N], dis[N], cnt;
int prim()
{
    int ans = 0;

    for (int i = 1; i <= n; i++)dis[i] = inf;
    dis[1] = 0;

```

```
    Q.push({ 0,1 });
    while (!Q.empty() && cnt < n)
    {
        pii x = Q.top(); Q.pop();
        int u = x.sec;

        if (vis[u])continue;
        vis[u] = 1;

        cnt++;///最后用来判断是否连通，cnt==n说明连通
        ans += x.fs;

        for (pii it : g[u])
        {
            int v = it.fs, w = it.sec;
            if (dis[v] > w)
            {
                dis[v] = w;
                Q.push({ dis[v],v });
            }
        }
    }
    return ans;
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int u, v, w; cin >> u >> v >> w;
        g[u].push_back({ v,w });
        g[v].push_back({ u,w });
    }
    int ans = prim();
    if (cnt == n)cout << ans;
    else cout << "orz";
    return 0;
}
```

- kuskal->并查集

```
int n, m;
struct edge///从定义出发存边
{
    int a, b, w;
    ///重载比较权值
    bool operator<(const edge& x)const { return w < x.w; }
}E[N];
int fa[N];///并查集
void init()
{
    for (int i = 1; i <= n; i++)fa[i] = i;
}
int find(int x)
{
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}
void kuskal()
{
    init();
    int ans = 0, cnt = 0;
    sort(E + 1, E + 1 + m);
    for (int i = 1; i <= m; i++)
    {
        int a = E[i].a, b = E[i].b, w = E[i].w;
        a = find(a), b = find(b);
        if (a != b)///不连通就放进集合
        {
            fa[a] = b;
            ans += w;
            cnt++;
        }
    }
    if (cnt == n - 1)cout << ans;
    else cout << "orz";
```

```
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int a, b, c; cin >> a >> b >> c;
        E[i] = { a,b,c };
    }
    kuskal();
    return 0;
}
```

1.6强连通分量

- 缩点模板

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
const int N = 1e5 + 5;
int n, m, w[N];
int u[N], v[N];
vector<int>g[N];///原图

int low[N], dfn[N], vis[N], cnt;///low是起始戳，dfn是时间戳
int st[N], top;///栈
int fa[N];///节点所在scc的编号
void tarjan(int u)
{
    low[u] = dfn[u] = ++cnt;
    st[++top] = u, vis[u] = 1;
    for (int v : g[u])
    {
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
```

```
        }
        else if (vis[v])
            low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u])
    {
        int v;
        while (v = st[top--])
        {
            fa[v] = u;///缩点，或者说染色
            vis[v] = 0;
            if (u == v)break;
            w[u] += w[v];///将一个强连通分量缩成一个点
        }
    }
}
vector<int>ng[N];///新图
int in[N], ans[N];///入度
int topo()
{
    queue<int>Q;
    for (int i = 1; i <= n; i++)
        if (!in[i])
        {
            Q.push(i);
            ans[i] = w[i];
        }
    while (!Q.empty())
    {
        int u = Q.front(); Q.pop();
        for (int v : ng[u])
        {
            ans[v] = max(ans[v], ans[u] + w[v]);///dp
            if (--in[v] == 0)Q.push(v);
        }
    }
    int sum = 0;
    for (int i = 1; i <= n; i++)sum = max(sum, ans[i]);
    return sum;
```

```
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)cin >> w[i];
    for (int i = 1; i <= m; i++)
    {
        cin >> u[i] >> v[i];///储存边
        g[u[i]].push_back(v[i]);///原图
    }
    for (int i = 1; i <= n; i++)///找强连通分量
        if (!dfn[i])tarjan(i);
    for (int i = 1; i <= m; i++)///建新图
    {
        int U = fa[u[i]], V = fa[v[i]];
        if (U != V)
        {
            ng[U].push_back(V);
            in[V]++;
        }
    }
    cout << topo();
    return 0;
}
```

- 最受欢迎的牛

```
const int N = 5e4 + 5;
int n, m;
vector<int>g[N];

int dfn[N], low[N], cnt;///时间戳，起始戳，时间
stack<int>s; int vis[N];///是否在栈中

int scc_num, fa[N], scc[N], out[N];
///强连通分量个数，节点所在scc的编号，节点所在scc的个数，出度
void tarjan(int u)
{
    low[u] = dfn[u] = ++cnt;
```

```
    s.push(u); vis[u] = 1;
    for (int v : g[u])
    {
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (vis[u])
            low[u] = min(low[u], dfn[v]);
    }
    if (low[u] == dfn[u])
    {
        scc_num++;
        int v;
        do {
            v = s.top(); s.pop();
            vis[v] = 0;
            fa[v] = scc_num;
            scc[scc_num]++;
        } while (v != u);
    }
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
    }
    for (int u = 1; u <= n; u++)if (!dfn[u])tarjan(u);///
求强连通分量
    for (int u = 1; u <= n; u++)///求scc图的出度
        for (int v : g[u])
            if (fa[u] != fa[v])out[fa[u]]++;
    int scc_cnt = 0, ans = 0;///特判
    for (int i = 1; i <= scc_num; i++)
    {
```

```
        if (out[i] == 0)
        {
            scc_cnt++;
            ans += scc[i];
        }
    }
    if (scc_cnt == 1)cout << ans;
    else cout << 0;
    return 0;
}
```

2.字符串

2.1哈希

```
//基数取1e9+7，模数取2^64，通过自然溢出取模
using ull = unsigned long long;
const int N = 100;
const ull base = 1e9 + 7;
///pow和init记得写到main函数里
//预处理base的n次方
ull powb[N];
void pow()
{
    powb[0] = 1;
    for (int i = 1; i <= N - 1; i++)
        powb[i] = powb[i - 1] * base;
}
//初始化哈希表a
char s[N];
ull a[N];
void init()
{
    for (int i = 1; i <= strlen(s + 1); i++)
        a[i] = a[i - 1] * base + s[i];
}
//询问子串的哈希值
```

```
ull query(int l, int r)
{
    return a[r] - (a[l - 1] * a[r - l + 1]);
}
```

2.2KMP

- 求一个串在另一个串中出现的位置

```
char s1[N], s2[N];
int len1, len2;
int nxt[N];
void getnext()///s2的最大board
{
    nxt[1] = nxt[0] = 0;
    int j = 0;
    for (int i = 2; i <= len2; i++)
    {
        while (s2[i] != s2[j + 1] && j != 0)j = nxt[j];
        if (s2[i] == s2[j + 1])nxt[i] = ++j;
    }
}
void KMP()
{
    int k = 0, num = 0;
    for (int i = 1; i <= len1; i++)
    {
        while (k != 0 && s1[i] != s2[k + 1])
            k = nxt[k];

        if (s1[i] == s2[k + 1])k++;
        if (k == len2)
        {
            num++;
            printf("%d\n", i - len2 + 1);
        }
    }
}
int main()
```

```
{
    scanf("%s%s", s1 + 1, s2 + 1);
    len1 = strlen(s1 + 1), len2 = strlen(s2 + 1);
    getnext();
    KMP();
    for (int i = 1; i <= len2; i++)
        printf("%d ", nxt[i]);
    return 0;
}
```

2.3Manacher

- 求最长回文子串

```
char s[N], a[N << 1];
int mana[N << 1];
int main()
{
    scanf("%s", s + 1);
    int len = strlen(s + 1);

    a[0] = '@';//构建新串
    int j = 1, i = 1;
    for (; i <= len * 2; j++, i += 2)
    {
        a[i] = '#';
        a[i + 1] = s[j];
    }a[i] = '#';
    //printf("%s\n", a + 1);
    len = i;
    int mr = 0, mid = 0;
    for (int i = 1; i <= len; i++)
    {
        if (i < mr)
            mana[i] = min(mana[mid * 2 - i], mr - i);
        while (a[i - mana[i] - 1] == a[i + mana[i] + 1])
            mana[i]++;
        if (i + mana[i] > mr) mr = i + mana[i], mid = i;
    }
```

```
    //for (int i = 1; i <= len; i++)cout << mana[i] << '
';
    int ans = 0;
    for (int i = 1; i <= len; i++)
        ans = max(ans, mana[i]);
    printf("%d", ans);
    return 0;
}
```

2.4字典树

```
int n, q;
int trie[N][65];
int getnum(char a)//映射字符
{
    if ('a' <= a && a <= 'z')return a - 'a';
    if ('A' <= a && a <= 'Z')return a - 'A' + 26;
    if ('0' <= a && a <= '9')return a - '0' + 52;
}
int tot, cnt[N];//ex是记录每个字符出现的次数
void insert(char* s)
{
    int p = 0;
    for (int i = 1; s[i]; i++)
    {
        int c = getnum(s[i]);
        if (!trie[p][c])
            trie[p][c] = ++tot;
        p = trie[p][c];
        cnt[p]++;//也可以记录其他信息
    }
}
int find(char* s)
{
    int p = 0;
    for (int i = 1; s[i]; i++)
    {
        int c = getnum(s[i]);
```

```
        if (!trie[p][c])
            return 0;
        p = trie[p][c];
    }
    return cnt[p];
}
char s[N];
int t;
int main()
{
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &n, &q);
        for (int i = 0; i <= tot; i++)ex[i] = 0;
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j <= 64; j++)
                trie[i][j] = 0;
        tot = 0;//清零
        for (int i = 1; i <= n; i++)
        {
            scanf("%s", s + 1);
            insert(s);//建树
        }
        for (int i = 1; i <= q; i++)
        {
            scanf("%s", s + 1);
            printf("%d\n", find(s));//查询
        }
    }
    return 0;
}
```

2.5AC自动机

- 求有多少个模式串在文本串里出现过

```
int trie[N][30], ex[N], tot;//字典树
void insert(char* s)
```

```
{
    int p = 0;
    for (int i = 1; s[i]; i++)
    {
        int c = s[i] - 'a';
        if (!trie[p][c])
            trie[p][c] = ++tot;
        p = trie[p][c];
    }
    ex[p]++;
}
queue<int>Q;
int fail[N];
void build()//求fail数组
{
    for (int i = 0; i < 26; i++)
        if (trie[0][i])Q.push(trie[0][i]);
    while (!Q.empty())
    {
        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(char* s)
{
    int res = 0, p = 0;
    for (int i = 1; s[i]; i++)
    {
        int c = s[i] - 'a';
        p = trie[p][c];
```

```
        for (int j = p; j != 0 && ex[j] != -1; j = fail[j])
        {
            res += ex[j];
            ex[j] = -1;
        }
    }
    return res;
}
char s[N];
int main()
{
    int n;scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        scanf("%s", s + 1);
        insert(s);
    }
    scanf("%s", s + 1);
    build();
    printf("%d", query(s));
    return 0;
}
```

3.数据结构

3.1树状数组

- 求逆序对

```
#define lowbit(x) (x&-x)
#define int long long
int n, c[N];
int a[N];
vector<int>b;
int get(int x)//离散化
{
    return lower_bound(b.begin(), b.end(), x) - b.begin()
+ 1;
```

```
}
void add(int x, int k)
{
    for (int i = x; i <= N - 5; i += lowbit(i))
        c[i] += k;
}
int getsum(int x)
{
    int res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += c[i];
    return res;
}
signed main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x; cin >> x;
        a[i] = x;
        b.push_back(x);
    }
    sort(b.begin(), b.end());
    b.erase(unique(b.begin(), b.end()), b.end());
    int ans = 0;
    for (int i = 1; i <= n; i++)
    {
        ans += i - 1 - getsum(get(a[i])) ;//总共有i-1个数，减去比它小的数，剩下比它大的数
        add(get(a[i]), 1);
    }
    cout << ans;
    return 0;
}
```

- 求树上逆序数

```cpp
void dfs(int u)
{
    ans[u] -= getsum(n) - getsum(a[u]);//减去之前比u强的
    for (int v : g[u])dfs(v);
    ans[u] += getsum(n) - getsum(a[u]);//加上遍历子树后比u强的
    add(a[u], 1);
 }
```

3.2ST表

- 反向在线求ST表可实现动态加点

```cpp
int n, m;//现推画图可理解
int st[N][22];//st_i,j 表示第i个元素为起点，长度为2^j序列的最大值
int main()
{
    cin >> n >> m;
    //第i个元素区间长度为1的序列的最大值就是它本身
    for (int i = 1; i <= n; i++)cin >> st[i][0];
    //以一个元素为起点所能到达的最大区间长度为2^log2(n)
    int log2_n = log2(n);
    for (int j = 1; j <= log2_n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)//防止越界
            st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);//把一个区间分成两个子区间
    for (int i = 1; i <= m; i++)///查询
    {
        int l, r;cin >> l >> r;
        int j = log2(r - l + 1);//从前往后和从后往前拆成两段
        cout << max(st[l][j], st[r - (1 << j) + 1][j]) << '\n';
    }
    return 0;
}
```

3.3线段树

- 区间修改，区间查询

```
#define int long long
int n, m, sum[N << 2], add[N << 2], a[N];
void bt(int k, int l, int r)
{
    add[k] = 0;//建树时初始化标记
    if (l == r)
    {
        sum[k] = a[r];
        return;
    }
    int m = (l + r) >> 1;
    bt(k + k, l, m);
    bt(k + k + 1, m + 1, r);
    sum[k] = sum[k + k] + sum[k + k + 1];
}
void pushdown(int k, int l, int r)
{
    int m = (l + r) >> 1;
    add[k + k] += add[k];
    sum[k + k] += (m - l + 1) * add[k];
    add[k + k + 1] += add[k];
    sum[k + k + 1] += (r - m) * add[k];
    add[k] = 0;
}
void update(int k, int l, int r, int x, int y, int z)
{
    if (l == x && r == y)
    {
        add[k] += z;//打上标记的同时更新区间，保证当前区间的值是正确的
        sum[k] += (r - l + 1) * z;
        return;
    }
    pushdown(k, l, r);
    int m = (l + r) >> 1;
```

```
    if (y <= m)
        update(k + k, l, m, x, y, z);
    else if (x > m)
        update(k + k + 1, m + 1, r, x, y, z);
    else
        update(k + k, l, m, x, m, z),
        update(k + k + 1, m + 1, r, m + 1, y, z);
    sum[k] = sum[k + k] + sum[k + k + 1];
}
int query(int k, int l, int r, int x, int y)
{
    if (l == x && r == y)return sum[k];//直接返回
    pushdown(k, l, r);
    int m = (l + r) >> 1;
    int ans = 0;
    if (y <= m)
        ans = query(k + k, l, m, x, y);
    else if (x > m)
        ans = query(k + k + 1, m + 1, r, x, y);
    else
        ans = query(k + k, l, m, x, m) + query(k + k + 1, m
+ 1, r, m + 1, y);
    sum[k] = sum[k + k] + sum[k + k + 1];//需要pushup才能返
回
    return ans;
}
signed main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)cin >> a[i];
    bt(1, 1, n);
    while (m--)
    {
        int op, x, y, k; cin >> op;
        if (op == 1)
        {
            cin >> x >> y >> k;
            update(1, 1, n, x, y, k);
        }
```

```
        else
        {
            cin >> x >> y;
            cout << query(1, 1, n, x, y) << '\n';
        }
    }
    return 0;
}
```

3.4链表

- 双向链表插入和删除操作

```
int n, m, k, p;//自己模拟一下
int nxt[N], pre[N], vis[N];
int main()
{
    cin >> n;
    for (int i = 2; i <= n; i++)
    {
        cin >> k >> p;
        if (p == 0)//左插
        {
            nxt[i] = k;
            nxt[pre[k]] = i;
            pre[i] = pre[k];
            pre[k] = i;
        }
        else//右插
        {
            pre[i] = k;
            pre[nxt[k]] = i;
            nxt[i] = nxt[k];
            nxt[k] = i;
        }
    }
    cin >> m;
    while (m--)
    {
```

```
        int x; cin >> x;
        if (vis[x])continue;
        vis[x] = 1;//删除
        nxt[pre[x]] = nxt[x];
        pre[nxt[x]] = pre[x];
    }
    for (int i = 1; i <= n; i++)
        if (nxt[i] != 0 && pre[i] == 0)
        {
            for (int j = i; j ; j = nxt[j])cout << j << ' ';
            break;
        }
    return 0;
}
```

3.5并查集

- 普通并查集

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

- 带权并查集，p1196

```

```

```cpp
int fa[N + 5], num[N + 5], front[N + 5];//同时维护x所在集合的大小，x到队头的距离
int find(int x)
{
    if (fa[x] == x)return x;
    int ast = find(fa[x]);
    front[x] += front[fa[x]];//x到队头的距离等于x到父节点的距离加上父节点到父节点的距离
    fa[x] = ast;//压缩路径
    return ast;
}
int main()
{
    for (int i = 1; i <= N; i++)
    {
        fa[i] = i;
        front[i] = 0;
        num[i] = 1;
    }
    int T;cin >> T;
    while (T--)
    {
        string op; int x, y; cin >> op >> x >> y;
        int fx = find(x), fy = find(y);
        if (op == "M")//合并
        {
            front[fx] += num[fy];//加上fy所在集合的大小

            fa[fx] = fy;
            num[fy] += num[fx];
            num[fx] = 0;
        }
        else
        {
            if (fx != fy)cout << -1 << '\n';
            else cout << abs(front[fx] - front[fy]) - 1 << '\n';
        }
    }
```

```
    return 0;
}
```

3.6单调栈

- p6510
- 题目大意：求区间满足左端点是区间的严格最小值，右端点是严格最大值，求最长区间的长度
- 单调栈维护最大值和最小值，up单调递增且小于a[i],dn单调递减且大于等于a[i]，二分查找dn栈顶在up中第一次出现的位置，若找到，更新答案

```
int n, a[N];
int up[N], dn[N], t1, t2;
int ans;
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)cin >> a[i];
    for (int i = 1; i <= n; i++)
    {
        while (t1 && a[up[t1]] >= a[i])//栈内维护的是下标
            t1--;
        while (t2 && a[dn[t2]] < a[i])
            t2--;

        int m = upper_bound(up + 1, up + 1 + t1, dn[t2]) -
up;
        if (m != t1 + 1)
            ans = max(ans, i - up[m] + 1);
        dn[++t2] = up[++t1] = i;
    }
    cout << ans;
    return 0;
}
```

3.7单调队列

- p3512
- 题目大意：求最长的区间，满足最大值和最小值之差不超过k
- 单调队列维护最大值和最小值，若队头之差超过k，选择序号较小的出队，同时更新当前队列最前节点的位置，答案即是当前位置减去当前队列最前节点的位置加一

```
int k, n;
int a[N], mxq[N], mnq[N];
int mxh, mnh, mxt, mnt, ans;//t->tail h->head
int main()
{
    cin >> k >> n;
    for (int i = 1; i <= n; i++)cin >> a[i];
    mxq[1] = 1, mnq[1] = 1, pre = 1;
    mxh = mxt = 1, mnh = mnt = 1;
    for (int i = 2; i <= n; i++)
    {
        while (mxh <= mxt && a[mxq[mxt]] < a[i])//维护最大值和最小值
            mxt--;
        while (mnh <= mnt && a[mnq[mnt]] > a[i])
            mnt--;
        mxq[++mxt] = i;
        mnq[++mnt] = i;
        int pre = 0;
        while (a[mxq[mxh]] - a[mnq[mnh]] > k)//如果超过了k
        {
            if (mxq[mxh] < mnq[mnh])//选择序号较小的
            {
                pre = mxq[mxh] + 1;
                mxh++;
            }else{
                pre = mnq[mnh] + 1;
                mnh++;
            }
        }
        ans = max(ans, i - pre + 1);
```

```
    }
    cout << ans;
    return 0;
}
```

4.数学

4.1高精度

```
void read(char str[], int arr[])//字符串转换为整型数组,倒着读入，倒着输出
{
    int len = strlen(str);
    for (int i = 0; i < len; i++)
    {
        int j = len - i;
        arr[j] = str[i] - '0';
    }
    arr[0] = len;//0位储存数据的长度
}
void print(int arr[])//输出
{
    int len = arr[0];
    for (int i = len; i >= 1; i--)
        printf("%d", arr[i]);
}
void add(int A[], int B[], int C[])//C=A+B
{
    int len = A[0] > B[0] ? A[0] : B[0];
    int tmp = 0;
    for (int i = 1; i <= len; i++)
    {
        C[i] = (tmp + A[i] + B[i]) % 10;
        tmp = (tmp + A[i] + B[i]) / 10;
    }
    C[0] = len + tmp;
    C[++len] = tmp;
```

```
}
void sub(int A[], int B[], int C[])//C=A-B(A>=B)
{
    int len = A[0];
    int tmp = 0;
    for (int i = 1; i <= len; i++)
    {
        C[i] = (A[i] - B[i] + tmp + 10) % 10;
        tmp = (A[i] - B[i] + tmp + 10) / 10 - 1;
    }
    while (len > 1 && C[len] == 0) len--;//位数减少，用于检索
位数
    C[0] = len;
}
void mul(int A[], int b, int C[])//C=A*B(高精度×低精度)
{
    int len = A[0];
    int tmp = 0;
    int i;
    for (i = 1; i <= len || tmp != 0; i++)
    {
        tmp = A[i] * b + tmp;
        C[i] = tmp % 10;
        tmp /= 10;
    }
    C[0] = i - 1;
}
int div(int A[], int b, int C[])//C=A/b(b为低精度)
{
    int len = A[0];
    int tmp = 0;
    int last = 0;
    for (int i = len; i >= 1; i--)
    {
        C[i] = (A[i] + tmp * 10) / b;
        tmp = (A[i] + tmp * 10) % b;
        if (i == 1)last = tmp;
    }
    while (len > 1 && C[len] == 0)len--;
```

```
    C[0] = len;
    return last;//返回余数,余数不能太大
}
```

4.2快速幂

```
long long qpow(long long a, long long b)
{
    long long res = 1;
    while (b)
    {
        if (b & 1)res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}
```

4.3欧几里得和拓展欧几里得

```
int gcd(int a, int b){return b == 0 ? a : gcd(b, a %
b);}//求最小公倍数：a*b/gcd(a,b)
void exgcd(int a, int b, int& x, int&
y)//ax+by=gcd(a,b)，求x,y
{
    if (b == 0)
    {
        x = 1;
        y = 0;
    }else{
        exgcd(b, a % b, y, x);
        y -= a / b * x;
    }
}
```

4.4素数

- 埃筛

```
bool not_prime[N];
int prime[N], cnt;
void getprime(int n)
{
    not_prime[1] = true;
    for (int i = 2; i <= n; i++)
        if (!not_prime[i])
        {
            prime[++cnt] = i;
            for (int j = i * 2; j <= n; j += i)
                not_prime[j] = true;
        }
}
```

- 线筛

```
bool not_prime[N];
int prime[N], cnt;
void getprime(int n)
{
    not_prime[1] = true;
    for (int i = 2; i <= n; i++)
    {
        if (!not_prime[i])
            prime[++cnt] = i;
        for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
        {
            not_prime[i * pirme[j]] = true;
            if(i % prime[j] == 0)break;//保证每个合数被最小的质因数筛去
        }
    }
}
```

- 预处理每个数的最大质因数或最小质因数

```cpp
bool not_prime[N];
int prime[N], cnt, mnp[N], mxp[N];
void getmxp(int n)
{
    not_prime[1] = true, mnp[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (!not_prime[i])
        {
            mnp[i] = i;//质数的最大质因子即其本身
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
        {
            not_prime[i * prime[j]] = true;
            mnp[i * prime[j]] = prime[j];
            if (i % prime[j] == 0)break;
        }
    }
}
void getmnp(int n)
{
    not_prime[1]=true;
    for(int i=2;i<=n;i++)
    {
        if(!not_prime[i])
        {
            mnp[i]=i;
            for(ll j=(ll)i*i;j<=n;j+=i)//其实就改这里，注意开
long long
            {
                not_prime[j]=true;
                mnp[j]=i;
            }
        }
    }
}
```

4.5数论分块

- 求$\sum_{i=0}^{n} \frac{n}{i}$

```
int main()
{
    int n; cin >> n;
    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;
    return 0;
}
```

4.6简单博弈

- nim游戏：给定n堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。问如果两个人都采用最优策略，先手是否必胜。

  结论：当所有石子堆的异或和为0时先手必败。

- 巴什博弈：有一堆n个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取m个。最后取光者得胜。问如果两个人都采用最优策略，先手是否必胜。
  结论：当n%(m+1)==0时必败。

- 威佐夫博弈：有两堆各若干个物品，两个人轮流从任一堆取至少一个或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。问先手是否必胜。
  结论：规定第一堆数大于等于第二堆，若第一个数是两数差值乘黄金比下取整，那么先手必败。

- 对称策略。

5.杂

5.1STL

a.set

- set_union：并集
- set_intersection：交集
- set_defference：差集
- set_symmetric_differenec：对称差集

```
#define all(x) x.begin(),x.end()
#define ins(x) inserter(x,x.begin())

set_symmetric_difference(all(x), all(y), ins(z));
```

b.priority_queue

```
priority_queue<int> B;//大根堆
priority_queue<int, vector<int>, greater<int>> S;//小根堆
```

c.string

- 读取整行和转换大小写

```
string s;cin>>s;
transform(s.begin(),s.end(),s.begin(),::tolower);
cin.ignore();
string str;getline(cin,str);
transform(str.begin(),str.end(),str.begin(),::tolower);
```

d.algorithm

- 枚举全排列

```cpp
for (int i = 1; i <= n; i++)p[i] = i;
do{
   for (int i = 1; i <= n; i++)cout << p[i] << ' ';
   cout << endl;
} while (next_permutation(p + 1, p + 1 + n));;
```

5.2二分模板

```cpp
int l = 1 , r = n ;//必要时增大区间长度
while (l < r)
{
   int mid = (l + r + 1) >> 1;
   if (check(mid))
      l = mid;
   else
      r = mid - 1;//必要时更新区间答案
}
while (l < r)
{
   int mid = (l + r) >> 1;
   if (check(mid))
      r = mid;
   else
      l = mid + 1;
}
```

5.3对拍

- data.cpp

```cpp
#include<iostream>
#include<windows.h>
#include<ctime>
using namespace std;
int n;
int main()
{
   srand(time(0));
```

```
    n=rand()%10+1;
    cout<<n<<'\n';
    for(int i=1;i<=n;i++)
        cout<<rand()%10+1<<' ';
    return 0;
}
```

- check.cpp

```
#include<iostream>
using namespace std;
int main()
{
    int t;cin>>t;
    while(t--)
    {
        system("data.exe>data.txt");
        system("my.exe<data.txt>my.txt");
        system("right.exe<data.txt>right.txt");
        if(system("fc right.txt my.txt"))
        {
            cout<<"note!\n";
            break;
        }
    }
    return 0;
}
```

5.4快读

```cpp
int rd()
{
    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 * 10 + ch - '0'; ch = getchar(); }
    return x * f;
}
```

5.5简单莫队

- p2709
- 题目大意：有一个长为n的整数序列a，值域为$[1,k]$。一共有m个询问，每个询问给定一个区间$[l,r]$，求$\sum_{i=1}^{k} c_i$。其中c_i表示数字i在$[l,r]$中的出现次数。

```cpp
const int N = 5e4 + 5;
int n, m, k;
long long ans[N];
int a[N], b[N];///a是原数组，b是桶,统计每个数出现的次数
int sz;///分块大小
struct node
{
    int l, r, id;
    bool operator<(const node& x)
    {
        if(l/sz != x.l/sz)return l<x.l;//不在相同块按块从左往右排序
        return r < x.r;//在相同块则按r降序排序
    }
}mo[N];
long long c;///随着区间移动而更新的答案
void add(int i)
{
    c += 2 * b[i] + 1;
    b[i]++;
}
```

```
void del(int i)
{
    c -= 2 * b[i] - 1;
    b[i]--;
}
int  main()
{
    cin >> n >> m >> k;///根本与k无关
    sz = sqrt(n);
    for (int i = 1; i <= n; i++)cin >> a[i];
    for (int i = 1; i <= m; i++)
    {
        cin >> mo[i].l >> mo[i].r;
        mo[i].id = i;
    }
    sort(mo + 1, mo + 1 + m);
    ///l:上一次查询的左端点，r:上一次查询的右端点
    for (int i = 1, l = 1, r = 0; i <= m; i++)
    {
        ///挪动操作
        while (l > mo[i].l)l--, add(a[l]);
        while (r < mo[i].r)r++, add(a[r]);
        while (l < mo[i].l)del(a[l]), l++;
        while (r > mo[i].r)del(a[r]), r--;
        ans[mo[i].id] = c;
    }
    for (int i = 1; i <= m; i++)cout << ans[i] << '\n';
    return 0;
}
```