

1.图论

1.1图的存储

1.2拓扑序列

1.3最短路

a.dijkstra

b.floyd

1.4树上问题

a.拓扑序数

b.LCA

c.树上差分

1.5最小生成树

1.6强连通分量

2.字符串

2.1哈希

2.2KMP

2.3Manacher

2.4字典树

2.5AC自动机

3.数据结构

3.1树状数组

3.2ST表

3.3线段树

3.4链表

3.5并查集

3.6单调栈

3.7单调队列

4.数学

4.1高精度

4.2快速幂

4.3欧几里得和拓展欧几里得

4.4素数

4.5数论分块

4.6简单博弈

5.杂

5.1STL

a.set

b.priority_queue

c.string

d.algorithm

5.2二分模板

5.3对拍

5.4快读

5.5简单莫队

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。

```
1 //-----//邻接表可以比较灵活
2 #include<functional>
3 int n, m;
4 int main()
5 {
6     cin >> n >> m;
7     vector<vector<int>>>g(n + 1);
8     for (int i = 1; i <= m; i++)
9     {
10         int u, v; cin >> u >> v;
11         g[i].push_back(v);
12     }
13     function<void(int)>dfs = [&](int u)//仿函数
14     {
15         for (auto v : g[u])
16             dfs(v);
17     };
18     dfs(1);
19     return 0;
20 }
```

```

0 //-----//
1 const int N = 1e5 + 5;
2 vector<int>g[N];
3 int n, m;
4 void dfs(int u)
5 {
6     for (auto v : g[u])
7         dfs(v);
8 }
9 int main()
10 {
11     cin >> n >> m;
12     for (int i = 1; i <= m; i++)
13     {
14         int u, v; cin >> u >> v;
15         g[u].push_back(v);
16     }
17     dfs(1);
18     return 0;
19 }
20 //-----//
21 const int N = 1e5 + 5;
22 int h[N], tot, to[N], nxt[N];
23 void add(int u, int v) { to[tot] = v; nxt[tot] = h[u];
24 h[u] = tot++; }
25 int n, m;
26 void dfs(int u)
27 {
28     for (int i = h[u]; ~i; i = nxt[i])
29     {
30         int v = to[i];
31         dfs(v);
32     }
33 }
34 int main()
35 {
36     memset(h, -1, sizeof h);
37     cin >> n >> m;
38     for (int i = 1; i <= m; i++)

```

```

8      {
9          int u, v; cin >> u >> v;
10         add(u, v);
11     }
12     return 0;
13 }

```

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。

```

1  int n, m;
2  int main()
3  {
4      cin >> n >> m;
5      vector<vector<int>>>g(n+1);
6      vector<int>in(n); //入度
7      for (int i = 1; i <= m; i++)
8      {
9          int u, v; cin >> u >> v;
10         g[u].push_back(v);
11         in[v]++;
12     }
13     function<void(void)>topo = [&]()
14     {
15         //小/大根堆求字典序最小/大拓扑序
16         priority_queue<int, vector<int>, greater<int>>>Q;
17         vector<int>L;
18         for (int i = 1; i <= n; i++)
19             if (in[i] == 0)
20                 Q.push(i); //
21         while (!Q.empty())
22         {
23             int u = Q.top(); Q.pop();
24             L.push_back(u); //区别在于每次只在队列中加入一个数
25             for (int v : g[u])
26                 if (--in[v] == 0)
27                     Q.push(v);
28         }
29     }
30     topo();
31     return 0;
32 }

```

```

2         }
2         if (L.size() == n)
3         {
4             for (int u : L) cout << u << ' ';
3             cout << '\n';
2         }
3         else cout << "NO\n";
4     };
5     topo();
6     return 0;
3 }

```

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。

```

1 #include<iostream>
2 #include<vector>
3 #include<cstring>
4 #include<queue>
5 #include<stack>
6 #define pii pair<int,int>
7 using namespace std;
8 const int inf = 1e9;
9 const int mod = 1e5 + 3;
1 const int N = 1e6 + 5;
0
1 int sum[N]; //记录最短路径数
2 int pre[N]; //记录前驱
3
4 int n, m, s;
5 vector<pii>g[N << 1];
6
7 int dis[N], vis[N];
8 vector<int>L[N];
9

```

```

2 priority_queue<pii, vector<pii> , greater<pii>>Q; //fs是w,
0 se是u
2 void dij()
2 {
2     for (int i = 1; i <= n; i++) dis[i] = inf;
2     dis[s] = 0;
2
2     for (int i = 1; i <= n; i++) pre[i] = -1; //前驱标记为-1
2     sum[s] = 1;
2
2     Q.push({ 0, s });
2     while (!Q.empty())
2     {
2         pair<int, int>x = Q.top(); Q.pop();
2         int u = x.second;
2
2         if (vis[u]) continue;
2         vis[u] = 1;
2
2         for (auto to : g[u])
2         {
2             int v = to.first, w = to.second; //fs是u, se是w
2             if (dis[v] > dis[u] + w)
2             {
2                 dis[v] = dis[u] + w;
2                 Q.push({ dis[v], v });
2
2                 sum[v] = sum[u];
2                 pre[v] = u;
2             }
2             else if (dis[v] == dis[u] + w)
2                 sum[v] = (sum[v] + sum[u]) % mod;
2         }
2     }
2 }
2 void path(int u)
2 {
2     //用栈逆序输出, 或者递归输出
2     stack<int>s;

```

```

5     do
6     {
7         s.push(u);
8         u = pre[u];
9     } while (u != -1);
10    while (!s.empty())
11    {
12        cout << s.top() << ' ';
13        s.pop();
14    }
15    cout << '\n';
16 }
17
18 int main()
19 {
20     cin >> n >> m >> s;
21     for (int i = 1; i <= m; i++)
22     {
23         int u, v, w; cin >> u >> v >> w;
24         g[u].push_back({ v,w });
25         g[v].push_back({ u,w });
26     }
27     dij();
28     for (int i = 1; i <= n; i++) path(i);
29     return 0;
30 }

```

b.floyd

- 求多源最短路。
简单应用：图的传递闭包，记录最短路径。

```

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

```

```

8 int pre[N][N];
9 void path(int s, int t)
1 {
0     if (dis[s][t] == inf)
1     {
2         cout << "none";
3         return;
4     }
5     stack<int>sk;
6     do
7     {
8         sk.push(t);
9         t = pre[s][t];
0     } while (t != -1);
1     sk.push(s);
2     while (!sk.empty())
3     {
4         cout << sk.top() << ' ';
5         sk.pop();
6     }
7     cout << '\n';
8 }
9 int main()
0 {
1     cin >> n >> m >> s;
2     for (int i = 1; i <= n; i++)
3         for (int j = 1; j <= n; j++)
4             dis[i][j] = i == j ? 0 : inf,
5             pre[i][j] = -1; //初始化
6     for (int i = 1; i <= m; i++)
7     {
8         int u, v, w; cin >> u >> v >> w;
9         dis[u][v] = w;
0     }
1     for (int k = 1; k <= n; k++)
2         for (int i = 1; i <= n; i++)
3             for (int j = 1; j <= n; j++)
4                 if (dis[i][j] > dis[i][k] + dis[k][j])
5                     dis[i][j] = dis[i][k] + dis[k][j],

```



```

6         pre[i][j] = k; //记录前驱
7         //dis[i][j] |= dis[i][k] & dis[k][j]; //传递闭包
8         path(1, 4);
9         return 0;
10    }

```

1.4 树上问题

a. 拓扑序数

- 求树或森林的拓扑序数公式：

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

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

b. LCA

- 基于倍增

```

1  int n;
2  vector<int>g[N];
3  ///LCA
4  int dep[N], fa[N][22];
5  void dfs(int u, int pre)
6  { //预处理深度信息和树上st表
7      dep[u] = dep[pre] + 1;
8      fa[u][0] = pre; //当前节点的第2^0=1个节点是它的父亲节点
9      //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
1     for (int i = 1; (1 << i) <= dep[u]; i++)
11         fa[u][i] = fa[fa[u][i - 1]][i - 1];
12     for (auto v : g[u])
13         if (v != pre) //无向边
14             dfs(v, u);
15 }
16 int LCA(int x, int y)

```

```

6 {
7     if (dep[x] < dep[y]) swap(x, y); //保证x的深度大于y的深度
8     for (int i = 20; i >= 0; i--)
9     { //x, y跳到同一深度
10         if (dep[fa[x][i]] >= dep[y])
11             x = fa[x][i];
12         if (x == y) return x; //特判
13     }
14     for (int i = 20; i >= 0; i--) //同时跳
15         if (fa[x][i] != fa[y][i])
16             x = fa[x][i], y = fa[y][i];
17     return fa[x][0]; //最后LCA是它们的父亲节点
18 }
19 int main()
20 {
21     cin >> n;
22     for (int i = 1; i <= n - 1; i++)
23     {
24         int u, v; cin >> u >> v;
25         g[u].push_back(v); g[v].push_back(u);
26     }
27     dfs(1, 0);
28     return 0;
29 }

```

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。

```

1  #include<iostream>
2  using namespace std;
3  const int N = 1e7 + 5;
4  int n, m;
5  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
6  int ssum[N], rsum[N];
7  int ans[N];
8
9  void dfs(int k, int fa)
10 {
11     if (k > n) return;
12     ssum[k] = s[k] + ssum[fa];
13     dfs(k + k, k);
14     dfs(k + k + 1, k);
15     rsum[k] = r[k] + rsum[k + k] + rsum[k + k + 1];
16 }
17
18 int main()
19 {
20     cin >> n >> m;
21     for (int i = 1; i <= m; i++)
22     {
23         int op, x; cin >> op >> x;
24         if (op == 1)
25             s[x]++;
26         else if (op == 2)
27             s[x]--, g++;
28         else if (op == 3)
29             r[x]++;
30         else
31             r[x]--, g++;
32     }
33     dfs(1, -1);
34 }

```

```

3     for (int i = 1; i <= n; i++) ans[ssum[i] + rsum[i] +
3 g]++;
3     for (int i = 0; i <= m; i++) cout << ans[i] << ' ';
4     return 0;
5 }

```

◦ 边差分:

将路径 u 到 v 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。

```

1  #include<iostream>
2  #include<vector>
3  using namespace std;
4  const int N = 2e5 + 5;
5  int n, k;
6  int u[N], v[N];
7  vector<int>g[N];
8  //LCA
9  int dep[N], fa[N][22], son[N];
10 int diff[N];
11 void get(int u, int fa)
12 {
13     for (int v : g[u])
14     {
15         if (v == fa) continue;
16         get(v, u);
17         diff[u] += diff[v];
18     }
19 }
20 int main()
21 {
22     cin >> n;
23     for (int i = 1; i <= n - 1; i++)
24     {
25         cin >> u[i] >> v[i];
26     }
27 }

```

```

2         g[u[i]].push_back(v[i]),
6 g[v[i]].push_back(u[i]);
2     }
2     dfs(1, -1);
8     cin >> k;
9     for (int i = 1; i <= k; i++)
6     {
8         int u, v; cin >> u >> v;
2         int o = lca(u, v);
3         diff[u]++, diff[v]++;
4         diff[o] -= 2;
5     }
6     get(1, -1);
3     for (int i = 1; i <= n - 1; i++)
8     { //根据儿子大小判断
9         if (son[u[i]] < son[v[i]]) cout << diff[u[i]] <<
0 ' ';
4         else cout << diff[v[i]] << ' ';
4     }
2     return 0;
3 }

```

1.5最小生成树

- prim->dijkstral

```

1 int n, m;
2 vector<pii>g[N];
3
4 priority_queue<pii, vector<pii>, greater<pii>>Q;
5 //Q的fs为w, sec为now节点
6 int vis[N], dis[N], cnt;
7 int prim()
8 {
9     int ans = 0;
1
10     for (int i = 1; i <= n; i++) dis[i] = inf;
11     dis[1] = 0;
12

```

```

3   Q.push({ 0,1 });
4   while (!Q.empty() && cnt < n)
5   {
6       pii x = Q.top(); Q.pop();
7       int u = x.sec;
8
9       if (vis[u]) continue;
10      vis[u] = 1;
11
12      cnt++; ///最后用来判断是否连通, cnt==n说明连通
13      ans += x.fs;
14
15      for (pii it : g[u])
16      {
17          int v = it.fs, w = it.sec;
18          if (dis[v] > w)
19          {
20              dis[v] = w;
21              Q.push({ dis[v],v });
22          }
23      }
24  }
25  return ans;
26 }
27 int main()
28 {
29     cin >> n >> m;
30     for (int i = 1; i <= m; i++)
31     {
32         int u, v, w; cin >> u >> v >> w;
33         g[u].push_back({ v,w });
34         g[v].push_back({ u,w });
35     }
36     int ans = prim();
37     if (cnt == n) cout << ans;
38     else cout << "orz";
39     return 0;
40 }

```

- kuskal->并查集

```
1 int n, m;
2 struct edge///从定义出发存边
3 {
4     int a, b, w;
5     ///重载比较权值
6     bool operator<(const edge& x) const { return w <
x.w; }
7 }E[N];
8 int fa[N];///并查集
9 void init()
10 {
11     for (int i = 1; i <= n; i++)fa[i] = i;
12 }
13 int find(int x)
14 {
15     return fa[x] == x ? x : fa[x] = find(fa[x]);
16 }
17 void kuskal()
18 {
19     init();
20     int ans = 0, cnt = 0;
21     sort(E + 1, E + 1 + m);
22     for (int i = 1; i <= m; i++)
23     {
24         int a = E[i].a, b = E[i].b, w = E[i].w;
25         a = find(a), b = find(b);
26         if (a != b)///不连通就放进集合
27         {
28             fa[a] = b;
29             ans += w;
30             cnt++;
31         }
32     }
33     if (cnt == n - 1)cout << ans;
34     else cout << "orz";
```

```

3 }
5 int main()
6 {
7     cin >> n >> m;
8     for (int i = 1; i <= m; i++)
9     {
10         int a, b, c; cin >> a >> b >> c;
11         E[i] = { a,b,c };
12     }
13     kuskal();
14     return 0;
15 }

```

1.6强连通分量

- 缩点模板

```

1 #include<iostream>
2 #include<vector>
3 #include<queue>
4 using namespace std;
5 const int N = 1e5 + 5;
6 int n, m, w[N];
7 int u[N], v[N];
8 vector<int>g[N];///原图
9
10 int low[N], dfn[N], vis[N], cnt;///low是起始戳，dfn是时间戳
11 int st[N], top;///栈
12 int fa[N];///节点所在scc的编号
13 void tarjan(int u)
14 {
15     low[u] = dfn[u] = ++cnt;
16     st[++top] = u, vis[u] = 1;
17     for (int v : g[u])
18     {
19         if (!dfn[v])
20         {
21             tarjan(v);
22             low[u] = min(low[u], low[v]);

```



```

2         }
3         else if (vis[v])
4             low[u] = min(low[u], dfn[v]);
5     }
6     if (dfn[u] == low[u])
7     {
8         int v;
9         while (v = st[top--])
10        {
11            fa[v] = u; ///缩点, 或者说染色
12            vis[v] = 0;
13            if (u == v) break;
14            w[u] += w[v]; ///将一个强连通分量缩成一个点
15        }
16    }
17 }
18 vector<int> ng[N]; ///新图
19 int in[N], ans[N]; ///入度
20 int topo()
21 {
22     queue<int> Q;
23     for (int i = 1; i <= n; i++)
24         if (!in[i])
25         {
26             Q.push(i);
27             ans[i] = w[i];
28         }
29     while (!Q.empty())
30     {
31         int u = Q.front(); Q.pop();
32         for (int v : ng[u])
33         {
34             ans[v] = max(ans[v], ans[u] + w[v]); ///dp
35             if (--in[v] == 0) Q.push(v);
36         }
37     }
38     int sum = 0;
39     for (int i = 1; i <= n; i++) sum = max(sum, ans[i]);
40     return sum;

```

```

1 }
2 int main()
3 {
4     cin >> n >> m;
5     for (int i = 1; i <= n; i++) cin >> w[i];
6     for (int i = 1; i <= m; i++)
7     {
8         cin >> u[i] >> v[i]; ///储存边
9         g[u[i]].push_back(v[i]); ///原图
10    }
11    for (int i = 1; i <= n; i++) ///找强连通分量
12        if (!dfn[i]) tarjan(i);
13    for (int i = 1; i <= m; i++) ///建新图
14    {
15        int U = fa[u[i]], V = fa[v[i]];
16        if (U != V)
17        {
18            ng[U].push_back(V);
19            in[V]++;
20        }
21    }
22    cout << topo();
23    return 0;
24 }

```

• 最受欢迎的牛

```

1 const int N = 5e4 + 5;
2 int n, m;
3 vector<int>g[N];
4
5 int dfn[N], low[N], cnt; ///时间戳, 起始戳, 时间
6 stack<int>s; int vis[N]; ///是否在栈中
7
8 int scc_num, fa[N], scc[N], out[N];
9 ///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
10 void tarjan(int u)
11 {
12     low[u] = dfn[u] = ++cnt;

```

```

2   s.push(u); vis[u] = 1;
3   for (int v : g[u])
4   {
5       if (!dfn[v])
6       {
7           tarjan(v);
8           low[u] = min(low[u], low[v]);
9       }
10      else if (vis[v])
11          low[u] = min(low[u], dfn[v]);
12  }
13  if (low[u] == dfn[u])
14  {
15      scc_num++;
16      int v;
17      do {
18          v = s.top(); s.pop();
19          vis[v] = 0;
20          fa[v] = scc_num;
21          scc[scc_num]++;
22      } while (v != u);
23  }
24  }
25  int main()
26  {
27      cin >> n >> m;
28      for (int i = 1; i <= m; i++)
29      {
30          int u, v; cin >> u >> v;
31          g[u].push_back(v);
32      }
33      for (int u = 1; u <= n; u++) if (!dfn[u]) tarjan(u); ///
4  求强连通分量
4      for (int u = 1; u <= n; u++) ///求scc图的出度
5          for (int v : g[u])
6              if (fa[u] != fa[v]) out[fa[u]]++;
7      int scc_cnt = 0, ans = 0; ///特判
8      for (int i = 1; i <= scc_num; i++)
9      {

```

```

0         if (out[i] == 0)
1         {
2             scc_cnt++;
3             ans += scc[i];
4         }
5     }
6     if (scc_cnt == 1) cout << ans;
7     else cout << 0;
8     return 0;
9 }

```

2.字符串

2.1哈希

```

1 //基数取1e9+7，模数取2^64，通过自然溢出取模
2 using ull = unsigned long long;
3 const int N = 100;
4 const ull base = 1e9 + 7;
5 ///pow和init记得写到main函数里
6 //预处理base的n次方
7 ull powb[N];
8 void pow()
9 {
10     powb[0] = 1;
11     for (int i = 1; i <= N - 1; i++)
12         powb[i] = powb[i - 1] * base;
13 }
14 //初始化哈希表a
15 char s[N];
16 ull a[N];
17 void init()
18 {
19     for (int i = 1; i <= strlen(s + 1); i++)
20         a[i] = a[i - 1] * base + s[i];
21 }
22 //询问子串的哈希值

```

```

2 ull query(int l, int r)
3 {
4     return a[r] - (a[l - 1] * a[r - 1 + 1]);
5 }

```

2.2KMP

- 求一个串在另一个串中出现的位置

```

1 char s1[N], s2[N];
2 int len1, len2;
3 int nxt[N];
4 void getnext() ///s2的最大board
5 {
6     nxt[1] = nxt[0] = 0;
7     int j = 0;
8     for (int i = 2; i <= len2; i++)
9     {
10         while (s2[i] != s2[j + 1] && j != 0) j = nxt[j];
11         if (s2[i] == s2[j + 1]) nxt[i] = ++j;
12     }
13 }
14 void KMP()
15 {
16     int k = 0, num = 0;
17     for (int i = 1; i <= len1; i++)
18     {
19         while (k != 0 && s1[i] != s2[k + 1])
20             k = nxt[k];
21
22         if (s1[i] == s2[k + 1]) k++;
23         if (k == len2)
24         {
25             num++;
26             printf("%d\n", i - len2 + 1);
27         }
28     }
29 }
30 int main()

```

```

0 {
1     scanf("%s%s", s1 + 1, s2 + 1);
2     len1 = strlen(s1 + 1), len2 = strlen(s2 + 1);
3     getnext();
4     KMP();
5     for (int i = 1; i <= len2; i++)
6         printf("%d ", nxt[i]);
7     return 0;
8 }

```

2.3 Manacher

- 求最长回文子串

```

1 char s[N], a[N << 1];
2 int mana[N << 1];
3 int main()
4 {
5     scanf("%s", s + 1);
6     int len = strlen(s + 1);
7
8     a[0] = '@'; //构建新串
9     int j = 1, i = 1;
10    for (; i <= len * 2; j++, i += 2)
11    {
12        a[i] = '#';
13        a[i + 1] = s[j];
14    } a[i] = '#';
15    //printf("%s\n", a + 1);
16    len = i;
17    int mr = 0, mid = 0;
18    for (int i = 1; i <= len; i++)
19    {
20        if (i < mr)
21            mana[i] = min(mana[mid * 2 - i], mr - i);
22        while (a[i - mana[i] - 1] == a[i + mana[i] + 1])
23            mana[i]++;
24        if (i + mana[i] > mr) mr = i + mana[i], mid = i;
25    }

```

```

5 //for (int i = 1; i <= len; i++)cout << mana[i] << '
6 ';
2 int ans = 0;
2 for (int i = 1; i <= len; i++)
2     ans = max(ans, mana[i]);
9 printf("%d", ans);
0 return 0;
3 }

```

2.4字典树

```

1 int n, q;
2 int trie[N][65];
3 int getnum(char a) //映射字符
4 {
5     if ('a' <= a && a <= 'z') return a - 'a';
6     if ('A' <= a && a <= 'Z') return a - 'A' + 26;
7     if ('0' <= a && a <= '9') return a - '0' + 52;
8 }
9 int tot, cnt[N]; //ex是记录每个字符出现的次数
1 void insert(char* s)
0 {
1     int p = 0;
2     for (int i = 1; s[i]; i++)
3     {
4         int c = getnum(s[i]);
5         if (!trie[p][c])
6             trie[p][c] = ++tot;
7         p = trie[p][c];
8         cnt[p]++; //也可以记录其他信息
9     }
0 }
2 int find(char* s)
2 {
3     int p = 0;
4     for (int i = 1; s[i]; i++)
5     {
6         int c = getnum(s[i]);

```

```

2         if (!trie[p][c])
3             return 0;
4         p = trie[p][c];
5     }
6     return cnt[p];
7 }
8
9 char s[N];
10 int t;
11 int main()
12 {
13     scanf("%d", &t);
14     while (t--)
15     {
16         scanf("%d%d", &n, &q);
17         for (int i = 0; i <= tot; i++) ex[i] = 0;
18         for (int i = 0; i <= tot; i++)
19             for (int j = 0; j <= 64; j++)
20                 trie[i][j] = 0;
21         tot = 0; //清零
22         for (int i = 1; i <= n; i++)
23         {
24             scanf("%s", s + 1);
25             insert(s); //建树
26         }
27         for (int i = 1; i <= q; i++)
28         {
29             scanf("%s", s + 1);
30             printf("%d\n", find(s)); //查询
31         }
32     }
33     return 0;
34 }

```

2.5AC自动机

- 求有多少个模式串在文本串里出现过

```

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

```



```

3 {
4     int p = 0;
5     for (int i = 1; s[i]; i++)
6     {
7         int c = s[i] - 'a';
8         if (!trie[p][c])
9             trie[p][c] = ++tot;
1            p = trie[p][c];
12    }
13    ex[p]++;
14 }
15 queue<int>Q;
16 int fail[N];
17 void build()//求fail数组
18 {
19     for (int i = 0; i < 26; i++)
20         if (trie[0][i])Q.push(trie[0][i]);
21     while (!Q.empty())
22     {
23         int u = Q.front();
24         Q.pop();
25         for (int i = 0; i < 26; i++)
26         {
27             if (trie[u][i])
28             {
29                 fail[trie[u][i]] = trie[fail[u]][i];
30                 Q.push(trie[u][i]);
31             }
32             else trie[u][i] = trie[fail[u]][i];
33         }
34     }
35 }
36 int query(char* s)
37 {
38     int res = 0, p = 0;
39     for (int i = 1; s[i]; i++)
40     {
41         int c = s[i] - 'a';
42         p = trie[p][c];

```

```

1         for (int j = p; j != 0 && ex[j] != -1; j = fail[j])
2         {
3             res += ex[j];
4             ex[j] = -1;
5         }
6     }
7     return res;
8 }
9 char s[N];
10 int main()
11 {
12     int n; scanf("%d", &n);
13     for (int i = 1; i <= n; i++)
14     {
15         scanf("%s", s + 1);
16         insert(s);
17     }
18     scanf("%s", s + 1);
19     build();
20     printf("%d", query(s));
21     return 0;
22 }

```

3.数据结构

3.1树状数组

- 求逆序对

```

1 #define lowbit(x) (x&-x)
2 #define int long long
3 int n, c[N];
4 int a[N];
5 vector<int>b;
6 int get(int x)//离散化
7 {
8     return lower_bound(b.begin(), b.end(), x) - b.begin()
9     + 1;

```

```

9 }
1 void add(int x, int k)
0 {
1     for (int i = x; i <= N - 5; i += lowbit(i))
2         c[i] += k;
3 }
4 int getsum(int x)
5 {
6     int res = 0;
7     for (int i = x; i; i -= lowbit(i))
8         res += c[i];
9     return res;
0 }
2 signed main()
2 {
3     cin >> n;
4     for (int i = 1; i <= n; i++)
5     {
6         int x; cin >> x;
7         a[i] = x;
8         b.push_back(x);
9     }
0     sort(b.begin(), b.end());
1     b.erase(unique(b.begin(), b.end()), b.end());
2     int ans = 0;
3     for (int i = 1; i <= n; i++)
4     {
5         ans += i - 1 - getsum(get(a[i])) ;//总共有i-1个数，减
6         去比它小的数，剩下比它大的数
7         add(get(a[i]), 1);
8     }
9     cout << ans;
0     return 0;
0 }

```

- 求树上逆序数

```

1 void dfs(int u)
2 {
3     ans[u] -= getsum(n) - getsum(a[u]); //减去之前比u强的
4     for (int v : g[u]) dfs(v);
5     ans[u] += getsum(n) - getsum(a[u]); //加上遍历子树后比u强的
6     add(a[u], 1);
7 }

```

3.2ST表

- 反向在线求ST表可实现动态加点

```

1 int n, m; //现推画图可理解
2 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
3 int main()
4 {
5     cin >> n >> m;
6     //第i个元素区间长度为1的序列的最大值就是它本身
7     for (int i = 1; i <= n; i++) cin >> st[i][0];
8     //以一个元素为起点所能到达的最大区间长度为2^log2(n)
9     int log2_n = log2(n);
10    for (int j = 1; j <= log2_n; j++)
11        for (int i = 1; i + (1 << j) - 1 <= n; i++) //防止越
12    界
13        st[i][j] = max(st[i][j - 1], st[i + (1 << (j -
14    1))][j - 1]); //把一个区间分成两个子区间
15    for (int i = 1; i <= m; i++) ///查询
16    {
17        int l, r; cin >> l >> r;
18        int j = log2(r - l + 1); //从前往后和从后往前拆成两段
19        cout << max(st[l][j], st[r - (1 << j) + 1][j]) <<
20    '\n';
21    }
22    return 0;
23 }

```

3.3线段树

- 区间修改，区间查询

```
1 #define int long long
2 int n, m, sum[N << 2], add[N << 2], a[N];
3 void bt(int k, int l, int r)
4 {
5     add[k] = 0; //建树时初始化标记
6     if (l == r)
7     {
8         sum[k] = a[r];
9         return;
10    }
11    int m = (l + r) >> 1;
12    bt(k + k, l, m);
13    bt(k + k + 1, m + 1, r);
14    sum[k] = sum[k + k] + sum[k + k + 1];
15 }
16 void pushdown(int k, int l, int r)
17 {
18     int m = (l + r) >> 1;
19     add[k + k] += add[k];
20     sum[k + k] += (m - l + 1) * add[k];
21     add[k + k + 1] += add[k];
22     sum[k + k + 1] += (r - m) * add[k];
23     add[k] = 0;
24 }
25 void update(int k, int l, int r, int x, int y, int z)
26 {
27     if (l == x && r == y)
28     {
29         add[k] += z; //打上标记的同时更新区间，保证当前区间的值是正
30         sum[k] += (r - l + 1) * z;
31         return;
32     }
33     pushdown(k, l, r);
34     int m = (l + r) >> 1;
```

```

3     if (y <= m)
4         update(k + k, l, m, x, y, z);
5     else if (x > m)
6         update(k + k + 1, m + 1, r, x, y, z);
7     else
8         update(k + k, l, m, x, m, z),
9         update(k + k + 1, m + 1, r, m + 1, y, z);
10    sum[k] = sum[k + k] + sum[k + k + 1];
11}
12
13int query(int k, int l, int r, int x, int y)
14{
15    if (l == x && r == y) return sum[k]; //直接返回
16    pushdown(k, l, r);
17    int m = (l + r) >> 1;
18    int ans = 0;
19    if (y <= m)
20        ans = query(k + k, l, m, x, y);
21    else if (x > m)
22        ans = query(k + k + 1, m + 1, r, x, y);
23    else
24        ans = query(k + k, l, m, x, m) + query(k + k + 1, m
25+ 1, r, m + 1, y);
26    sum[k] = sum[k + k] + sum[k + k + 1]; //需要pushup才能返
27回
28    return ans;
29}
30
31signed main()
32{
33    cin >> n >> m;
34    for (int i = 1; i <= n; i++) cin >> a[i];
35    bt(1, 1, n);
36    while (m--)
37    {
38        int op, x, y, k; cin >> op;
39        if (op == 1)
40        {
41            cin >> x >> y >> k;
42            update(1, 1, n, x, y, k);
43        }

```

```

1         else
2         {
3             cin >> x >> y;
4             cout << query(1, 1, n, x, y) << '\n';
5         }
6     }
7     return 0;
8 }

```

3.4链表

- 双向链表插入和删除操作

```

1 int n, m, k, p; //自己模拟一下
2 int nxt[N], pre[N], vis[N];
3 int main()
4 {
5     cin >> n;
6     for (int i = 2; i <= n; i++)
7     {
8         cin >> k >> p;
9         if (p == 0) //左插
10        {
11            nxt[i] = k;
12            nxt[pre[k]] = i;
13            pre[i] = pre[k];
14            pre[k] = i;
15        }
16        else //右插
17        {
18            pre[i] = k;
19            pre[nxt[k]] = i;
20            nxt[i] = nxt[k];
21            nxt[k] = i;
22        }
23    }
24    cin >> m;
25    while (m--)
26    {

```

```

0      int x; cin >> x;
2      if (vis[x]) continue;
8      vis[x] = 1; //删除
9      nxt[pre[x]] = nxt[x];
0      pre[nxt[x]] = pre[x];
3  }
2  for (int i = 1; i <= n; i++)
3      if (nxt[i] != 0 && pre[i] == 0)
4      {
5          for (int j = i; j ; j = nxt[j]) cout << j << ' ';
6          break;
3      }
8  return 0;
9 }

```

3.5并查集

- 普通并查集

```

1 void init()
2 {
3     for (int i = 1; i <= n; i++) fa[i] = i;
4 }
5 int find(int x)
6 {
7     return fa[x] == x ? x : fa[x] = find(fa[x]);
8 }
9 void merge(int x, int y)
10 {
11     fa[x] = fa[find(x)] = find(y);
12 }
13 bool test(int x, int y)
14 {
15     return find(x) == find(y);
16 }

```

- 带权并查集，p1196


```

1  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
2  int find(int x)
3  {
4      if (fa[x] == x) return x;
5      int ast = find(fa[x]);
6      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
7      fa[x] = ast; //压缩路径
8      return ast;
9  }
1 int main()
0 {
1     for (int i = 1; i <= N; i++)
2     {
3         fa[i] = i;
4         front[i] = 0;
5         num[i] = 1;
6     }
7     int T; cin >> T;
8     while (T--)
9     {
0         string op; int x, y; cin >> op >> x >> y;
1         int fx = find(x), fy = find(y);
2         if (op == "M") //合并
3         {
4             front[fx] += num[fy]; //加上fy所在集合的大小
5
6             fa[fx] = fy;
7             num[fy] += num[fx];
8             num[fx] = 0;
9         }
0         else
1         {
2             if (fx != fy) cout << -1 << '\n';
3             else cout << abs(front[fx] - front[fy]) - 1 <<
4 '\n';
5         }
6     }
7 }

```

```

8     return 0;
3 }

```

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

```

1  int n, a[N];
2  int up[N], dn[N], t1, t2;
3  int ans;
4  int main()
5  {
6      cin >> n;
7      for (int i = 1; i <= n; i++) cin >> a[i];
8      for (int i = 1; i <= n; i++)
9      {
10         while (t1 && a[up[t1]] >= a[i]) //栈内维护的是下标
11             t1--;
12         while (t2 && a[dn[t2]] < a[i])
13             t2--;
14         int m = upper_bound(up + 1, up + 1 + t1, dn[t2]) -
15 up;
16         if (m != t1 + 1)
17             ans = max(ans, i - up[m] + 1);
18         dn[++t2] = up[++t1] = i;
19     }
20     cout << ans;
21     return 0;
22 }

```

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

```
1 int k, n;
2 int a[N], mxq[N], mnq[N];
3 int mxh, mnh, mxt, mnt, ans; //t->tail h->head
4 int main()
5 {
6     cin >> k >> n;
7     for (int i = 1; i <= n; i++) cin >> a[i];
8     mxq[1] = 1, mnq[1] = 1, pre = 1;
9     mxh = mxt = 1, mnh = mnt = 1;
10    for (int i = 2; i <= n; i++)
11    {
12        while (mxh <= mxt && a[mxq[mxt]] < a[i]) //维护最大值
13            mxt--;
14        while (mnh <= mnt && a[mnq[mnt]] > a[i])
15            mnt--;
16        mxq[++mxt] = i;
17        mnq[++mnt] = i;
18        int pre = 0;
19        while (a[mxq[mxh]] - a[mnq[mnh]] > k) //如果超过了k
20        {
21            if (mxq[mxh] < mnq[mnh]) //选择序号较小的
22            {
23                pre = mxq[mxh] + 1;
24                mxh++;
25            } else {
26                pre = mnq[mnh] + 1;
27                mnh++;
28            }
29        }
30        ans = max(ans, i - pre + 1);
31    }
```

```

0     }
3     cout << ans;
2     return 0;
3 }

```

4.数学

4.1高精度

```

1 void read(char str[], int arr[])//字符串转换为整型数组,倒着读
  入,倒着输出
2 {
3     int len = strlen(str);
4     for (int i = 0; i < len; i++)
5     {
6         int j = len - i;
7         arr[j] = str[i] - '0';
8     }
9     arr[0] = len;//0位储存数据的长度
10 }
11 void print(int arr[])//输出
12 {
13     int len = arr[0];
14     for (int i = len; i >= 1; i--)
15         printf("%d", arr[i]);
16 }
17 void add(int A[], int B[], int C[])//C=A+B
18 {
19     int len = A[0] > B[0] ? A[0] : B[0];
20     int tmp = 0;
21     for (int i = 1; i <= len; i++)
22     {
23         C[i] = (tmp + A[i] + B[i]) % 10;
24         tmp = (tmp + A[i] + B[i]) / 10;
25     }
26     C[0] = len + tmp;
27     C[++len] = tmp;
28 }

```

```

2 }
2 void sub(int A[], int B[], int C[]) //C=A-B (A>=B)
9 {
0     int len = A[0];
3     int tmp = 0;
2     for (int i = 1; i <= len; i++)
3     {
4         C[i] = (A[i] - B[i] + tmp + 10) % 10;
5         tmp = (A[i] - B[i] + tmp + 10) / 10 - 1;
6     }
7     while (len > 1 && C[len] == 0) len--; //位数减少, 用于检索
8 位数
3     C[0] = len;
9 }
4 void mul(int A[], int b, int C[]) //C=A*B (高精度×低精度)
4 {
2     int len = A[0];
3     int tmp = 0;
4     int i;
5     for (i = 1; i <= len || tmp != 0; i++)
6     {
7         tmp = A[i] * b + tmp;
8         C[i] = tmp % 10;
9         tmp /= 10;
6     }
5     C[0] = i - 1;
2 }
3 int div(int A[], int b, int C[]) //C=A/b (b为低精度)
4 {
5     int len = A[0];
6     int tmp = 0;
7     int last = 0;
8     for (int i = len; i >= 1; i--)
9     {
6         C[i] = (A[i] + tmp * 10) / b;
6         tmp = (A[i] + tmp * 10) % b;
8         if (i == 1) last = tmp;
3     }
4     while (len > 1 && C[len] == 0) len--;

```

```

5   C[0] = len;
6   return last; //返回余数,余数不能太大
7 }

```

4.2快速幂

```

1 long long qpow(long long a, long long b)
2 {
3     long long res = 1;
4     while (b)
5     {
6         if (b & 1) res = res * a % p;
7         a = a * a % p;
8         b >>= 1;
9     }
10    return res;
11 }

```

4.3欧几里得和拓展欧几里得

```

1 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)
2 void exgcd(int a, int b, int& x, int&
y) //ax+by=gcd(a,b), 求x,y
3 {
4     if (b == 0)
5     {
6         x = 1;
7         y = 0;
8     }else{
9         exgcd(b, a % b, y, x);
10        y -= a / b * x;
11    }
12 }

```

4.4素数

- 埃筛

```
1 bool not_prime[N];
2 int prime[N], cnt;
3 void getprime(int n)
4 {
5     not_prime[1] = true;
6     for (int i = 2; i <= n; i++)
7         if (!not_prime[i])
8         {
9             prime[++cnt] = i;
1            for (int j = i * 2; j <= n; j += i)
12                not_prime[j] = true;
13        }
14 }
```

- 线筛

```
1 bool not_prime[N];
2 int prime[N], cnt;
3 void getprime(int n)
4 {
5     not_prime[1] = true;
6     for (int i = 2; i <= n; i++)
7     {
8         if (!not_prime[i])
9             prime[++cnt] = i;
1            for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
12            {
13                not_prime[i * prime[j]] = true;
14                if(i % prime[j] == 0)break;//保证每个合数被最小的质
15                因数筛去
16            }
17        }
18 }
```

- 预处理每个数的最大质因数或最小质因数

```

1 bool not_prime[N];
2 int prime[N], cnt, mnp[N], mxp[N];
3 void getmxp(int n)
4 {
5     not_prime[1] = true, mnp[1] = 1;
6     for (int i = 2; i <= n; i++)
7     {
8         if (!not_prime[i])
9         {
10             mnp[i] = i; //质数的最大质因子即其本身
11             prime[++cnt] = i;
12         }
13         for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
14         {
15             not_prime[i * prime[j]] = true;
16             mnp[i * prime[j]] = prime[j];
17             if (i % prime[j] == 0) break;
18         }
19     }
20 }
21 void getmnp(int n)
22 {
23     not_prime[1] = true;
24     for (int i = 2; i <= n; i++)
25     {
26         if (!not_prime[i])
27         {
28             mnp[i] = i;
29             for (ll j = (ll) i * i; j <= n; j += i) //其实就改这里，注意开
30             long long
31             {
32                 not_prime[j] = true;
33                 mnp[j] = i;
34             }
35         }
36     }
37 }

```


4.5数论分块

- 求 $\sum_{i=1}^n \frac{n}{i}$

```
1 int main()
2 {
3     int n; cin >> n;
4     int ans = 0;
5     for (int l = 1, r; l <= n; l = r + 1)
6     {
7         r = n / (n / l);
8         ans += (r - l + 1) * (n / l);
9     }
10    cout << ans;
11    return 0;
12 }
```

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 $n \% (m+1) \neq 0$ 时必胜。

- 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_difference: 差集
- set_symmetric_difference: 对称差集

```
1 #define all(x) x.begin(),x.end()
2 #define ins(x) inserter(x,x.begin())
3
4 set_symmetric_difference(all(x), all(y), ins(z));
```

b.priority_queue

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

c.string

- 读取整行和转换大小写

```
1 string s;cin>>s;
2 transform(s.begin(),s.end(),s.begin(),::tolower);
3 cin.ignore();
4 string str;getline(cin,str);
5 transform(str.begin(),str.end(),str.begin(),::tolower);
```

d.algorithm

- 枚举全排列

```

1 for (int i = 1; i <= n; i++)p[i] = i;
2 do{
3     for (int i = 1; i <= n; i++)cout << p[i] << ' ';
4     cout << endl;
5 } while (next_permutation(p + 1, p + 1 + n));;

```

5.2二分模板

```

1 int l = 1 , r = n ;//必要时增大区间长度
2 while (l < r)
3 {
4     int mid = (l + r + 1) >> 1;
5     if (check(mid))
6         l = mid;
7     else
8         r = mid - 1;//必要时更新区间答案
9 }
1 while (l < r)
10 {
11     int mid = (l + r) >> 1;
12     if (check(mid))
13         r = mid;
14     else
15         l = mid + 1;
16 }

```

5.3对拍

- data.cpp

```

1 #include<iostream>
2 #include<windows.h>
3 #include<ctime>
4 using namespace std;
5 int n;
6 int main()
7 {
8     srand(time(0));

```

```

9      n=rand()%10+1;
1      cout<<n<<'\n';
0      for(int i=1;i<=n;i++)
1          cout<<rand()%10+1<<' ';
2      return 0;
3  }

```

- check.cpp

```

1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      int t;cin>>t;
6      while(t-->0)
7      {
8          system("data.exe>data.txt");
9          system("my.exe<data.txt>my.txt");
1         system("right.exe<data.txt>right.txt");
1         if(system("fc right.txt my.txt"))
2         {
3             cout<<"note!\n";
4             break;
5         }
6     }
7     return 0;
8 }

```

5.4快读

```

1 int rd()
2 {
3     int x = 0, f = 1; char ch = getchar();
4     while (ch < '0' || ch > '9') { if (ch == '-') f = -1; ch
= getchar(); }
5     while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0';
ch = getchar(); }
6     return x * f;
7 }

```

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。

```

1 const int N = 5e4 + 5;
2 int n, m, k;
3 long long ans[N];
4 int a[N], b[N]; ///a是原数组, b是桶, 统计每个数出现的次数
5 int sz; ///分块大小
6 struct node
7 {
8     int l, r, id;
9     bool operator<(const node& x)
10     {
11         if(l/sz != x.l/sz) return l<x.l; ///不在相同块按块从左往右
12         排序
13         return r < x.r; ///在相同块则按r降序排序
14     }
15 } mo[N];
16 long long c; ///随着区间移动而更新的答案
17 void add(int i)
18 {
19     c += 2 * b[i] + 1;
20     b[i]++;
21 }

```

```

0 void del(int i)
1 {
2     c -= 2 * b[i] - 1;
3     b[i]--;
4 }
5 int main()
6 {
7     cin >> n >> m >> k; ///根本与k无关
8     sz = sqrt(n);
9     for (int i = 1; i <= n; i++) cin >> a[i];
0     for (int i = 1; i <= m; i++)
1     {
2         cin >> mo[i].l >> mo[i].r;
3         mo[i].id = i;
4     }
5     sort(mo + 1, mo + 1 + m);
6     ///l:上一次查询的左端点, r:上一次查询的右端点
7     for (int i = 1, l = 1, r = 0; i <= m; i++)
8     {
9         ///挪动操作
0         while (l > mo[i].l) l--, add(a[l]);
1         while (r < mo[i].r) r++, add(a[r]);
2         while (l < mo[i].l) del(a[l]), l++;
3         while (r > mo[i].r) del(a[r]), r--;
4         ans[mo[i].id] = c;
5     }
6     for (int i = 1; i <= m; i++) cout << ans[i] << '\n';
7     return 0;
8 }

```

