

# 常用模板（包括网络赛）

## 一、基础算法

### 1. 二分

#### 1. 整数二分

```
1 // 1
2 while (l < r) {
3     int mid = (l + r) >> 1;
4     if (check(mid)) l = mid + 1;
5     else r = mid;
6 }
7 // 2
8 while (l < r) {
9     int mid = (l + r + 1) >> 1;
10    if (check(mid)) l = mid;
11    else r = mid - 1;
12 }
13 // 常用
14 while (l <= r) {
15     int mid = (l + r) >> 1;
16     if (check(mid)) ans = mid, l = mid + 1;
17     else r = mid - 1;
18 }
```

#### 2. 浮点数二分

```
1 // 通用版
2 while (r - l > 1e-5) {
3     double mid = (l + r) / 2;
4     if (check(mid)) l = mid;
5     else r = mid;
6 }
7 // 防卡精度
8 for (int i = 0; i < 100; ++i) {
9     double mid = (l + r) / 2;
10    if (check(mid)) l = mid;
11    else r = mid;
12 }
```

### 2. 离散化

```
1 vector<int> vt;
2 //二分找离散化值
3 inline int get_id(int x) { return lower_bound(vt.begin(), vt.end(), x) - vt.begin() + 1; }
4 inline void erase_vt() { // 离散化后去重
5     sort(vt.begin(), vt.end());
6     vt.erase(unique(vt.begin(), vt.end()), vt.end());
7 }
8 inline void id_table(int n, int *a, vector<int>& res) { // 打表, 注意, 原数组下标要从1开始, 返回离散化后的表
9     res.emplace_back(0);
10    for (int i = 1; i <= n; ++i) res.emplace_back(get_id(a[i]));
11 }
```

### 3. 高精

```
1 #include <cstdio>
2 #include <cctype>
3 #include <cstring>
4 #include <algorithm>
5 #include <string>
6 #include <iostream>
7
8 using namespace std;
9
10 typedef long long ll;
11 /*
12 用前必读:
13 本高精采用可调压位式运算对于不同需求记得更改下面的压位代码
14 注意笔者只封装了 bign * int 没有封装 int * bign , 用时注意顺序
15 若出现 bign * x, 中 x 不是常量变量或者不是变量 (例如具体数字 3 4 5.....)
16 记得把封装的与int相关的 运算符重载函数 的const和取地址符&去掉, 防止报错
```

```

17 若是爆栈请尝试开全局变量或者把数组改小即 M 改小
18 本代码暂时只支持
19 高精 + 高精
20 高精1 - 高精2 (高精1 > 高精2)
21 高精 * 低精
22 高精 / 低精
23 高精 * 高精
24 高精 % 低精
25 高精 与 高精 的大小对比
26 高精 += 高精
27 高精1 -= 高精2 (高精1 > 高精2)
28 高精 *= 低精
29 高精 /= 低精
30 高精 *= 高精
31 高精 %= 低精
32 尚未完成的功能
33 高精 / 高精
34 高精 % 高精
35 高精1 - 高精2 (高精1 < 高精2)
36 .....
37 其余请读者自己体会
38 */
39
40 const int w = 1e8, M = 1e4, wsize = 8; // 压位8个0
41 const char pout[] = "%08lld"; // 记得修改
42 struct bign{
43     ll num[M];
44     char str[M * wsize];
45     int len;
46     void clear() { memset(num, 0, sizeof num); len = 0; }
47     /* 初始化 */
48     bign() : len(0) { clear(); }
49     bign(int n);
50     bign(ll n);
51     bign(char str[]);
52     bign(string str);
53     void change();
54     void operator= (const int x) { *this = bign(x); }
55     void operator= (const ll x) { *this = bign(x); }
56     void operator= (char x[]) { *this = bign(x); }
57     void operator= (string x) { *this = bign(x); }
58     ll &operator[] (int x) { return num[x]; }
59     /* 输出 */
60     void print();
61     void print() const;
62     /* 比较(未验证) */
63     bool operator< (const bign &b) const;
64     bool operator> (const bign &b) const { return b < *this; }
65     bool operator<= (const bign &b) const { return !(b < *this); }
66     bool operator>= (const bign &b) const { return !(*this < b); }
67     bool operator!= (const bign &b) const { return b < *this || *this < b; }
68     bool operator== (const bign &b) const { return !(b < *this) && !(*this < b); }
69     /* 各种运算 */
70     bign operator+ (const bign &b) const; // 高精 + 高精
71     bign operator* (const int &b) const; // 高精 * 低精
72     bign operator* (const bign &b); // 高精 * 高精
73     bign operator- (const bign &b) const; // 高精 - 高精
74     bign operator/ (const int &b) const; // 高精 / 低精
75     bign operator% (const int &b); // 高精 % 低精
76     void operator+= (const bign &b); // 高精 += 高精
77     void operator*= (const int &b); // 高精 *= 低精
78     void operator*= (const bign &b); // 高精 *= 高精
79     void operator-= (const bign &b); // 高精 -= 高精
80     void operator/= (const int &b); // 高精 /= 低精
81     void operator%= (const int &b); // 高精 %= 低精
82     /* 输入输出重载 */
83     friend istream& operator>> (istream &in, bign &res);
84     friend ostream& operator<< (ostream &out, const bign &res);
85
86 };
87 bign::bign(int n) : len(0) {
88     clear();
89     while (1) {
90         num[++len] = n % w, n /= w;
91         if (!n) break;
92     }
93 }
94 bign::bign(ll n) : len(0) {
95     clear();
96     while (1) {
97         num[++len] = n % w, n /= w;
98         if (!n) break;
99     }
100 }
101 bign::bign(char str[]) : len(0) {
102     int l = strlen(str) - 1;
103     for (int i = l; i >= 0; i -= wsize) {

```

```

104         ll tmp = 0, k = 1;
105         for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
106             tmp += (str[i - j] - '0') * k;
107         }
108         num[++len] = tmp;
109     }
110 }
111 bign::bign(string str) : len(0) {
112     int l = str.size() - 1;
113     for (int i = l; i >= 0; i -= wsize) {
114         ll tmp = 0, k = 1;
115         for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
116             tmp += (str[i - j] - '0') * k;
117         }
118         num[++len] = tmp;
119     }
120 }
121 void bign::change() {
122     int l = strlen(str) - 1;
123     len = 0;
124     for (int i = l; i >= 0; i -= wsize) {
125         ll tmp = 0, k = 1;
126         for (int j = 0; j < wsize && i - j >= 0; j++, k *= 10) {
127             tmp += (str[i - j] - '0') * k;
128         }
129         num[++len] = tmp;
130     }
131 }
132 /* 输出 */
133 void bign::print() {
134     printf("%lld", num[len]);
135     for (int i = len - 1; i > 0; i--) printf(pout, num[i]);
136     puts("");
137 }
138
139 void bign::print() const {
140     printf("%lld", num[len]);
141     for (int i = len - 1; i > 0; i--) printf(pout, num[i]);
142     // puts("");
143 }
144
145 /* 比较(未验证) */
146 bool bign::operator< (const bign &b) const {
147     if (len != b.len) return len < b.len;
148     for (int i = len; i > 0; i--) {
149         if (num[i] != b.num[i]) return num[i] < b.num[i];
150     }
151     return false;
152 }
153
154 /******常用******/
155 /* 各种运算 */
156 bign bign::operator+ (const bign &b) const {
157     bign res = *this;
158     if (res.len < b.len) res.len = b.len;
159     for (int i = 1; i <= res.len; i++) {
160         res.num[i] += b.num[i];
161         res.num[i + 1] += res.num[i] / w;
162         res.num[i] %= w;
163     }
164     while (res.num[res.len + 1]) res.len++;
165     return res;
166 }
167
168 bign bign::operator* (const int &b) const {
169     bign res;
170     ll carry = 0;
171     for (int i = 1; i <= len; i++) {
172         ll tmp = num[i] * b + carry;
173         res.num[++res.len] = tmp % w;
174         carry = tmp / w;
175     }
176     while (carry) {
177         res.num[++res.len] = carry % w;
178         carry /= w;
179     }
180     return res;
181 }
182
183 bign bign::operator* (const bign &b) {
184     bign res;
185     for (int i = 1; i <= len; i++) {
186         ll up = 0;
187         for (int j = 1; j <= b.len; j++) {
188             ll tmp = num[i] * b.num[j] + res.num[i + j - 1] + up;
189             res.num[i + j - 1] = tmp % w;
190             up = tmp / w;

```

```

191     }
192     if (up) res.num[i + b.len] = up;
193 }
194 res.len = len + b.len;
195 while (res.len > 1 && res.num[res.len] == 0) res.len--;
196 return res;
197 }
198 /*****
199
200 bign bign::operator- (const bign &b) const {
201     bign res = *this;
202     for (int i = 1; i <= len; i++) {
203         if (res.num[i] < b.num[i]) res.num[i] += w, res.num[i + 1]--;
204         res.num[i] -= b.num[i];
205     }
206     while (res.len > 1 && res.num[res.len] == 0) res.len--;
207     return res;
208 }
209
210 bign bign::operator/ (const int &b) const {
211     bign res;
212     res.len = len;
213     ll r = 0; //余数
214     for (int i = len; i > 0; i--) {
215         r = r * w + num[i];
216         if (r < b) res.num[i] = 0;
217         else res.num[i] = r / b, r %= b;
218     }
219     while (res.len > 1 && res.num[res.len] == 0) res.len--;
220     return res;
221 }
222
223 bign bign::operator% (const int &b) {
224     ll r = 0; //余数
225     for (int i = len; i > 0; i--) {
226         r = r * w + num[i];
227         if (r >= b) r %= b;
228     }
229     return bign(r);
230 }
231 void bign::operator+= (const bign &b) {
232     if (len < b.len) len = b.len;
233     for (int i = 1; i <= len; i++) {
234         num[i] += b.num[i];
235         num[i + 1] += num[i] / w;
236         num[i] %= w;
237     }
238     while (num[len + 1]) len++;
239 }
240 void bign::operator*= (const int &b) {
241     for (int i = 1; i <= len; i++) num[i] *= b;
242     for (int i = 1; i <= len; i++) {
243         num[i + 1] += num[i] / w;
244         num[i] %= w;
245     }
246     while (num[len + 1]) {
247         len++;
248         num[len + 1] = num[len] / w;
249         num[len] %= w;
250     }
251 }
252
253 void bign::operator*= (const bign &b) {
254     *this = *this * b;
255 }
256
257 void bign::operator-= (const bign &b) {
258     for (int i = 1; i <= len; i++) {
259         if (num[i] < b.num[i]) num[i] += w, num[i + 1]--;
260         num[i] -= b.num[i];
261     }
262     while (len > 1 && num[len] == 0) len--;
263 }
264
265 void bign::operator/= (const int &b) {
266     ll r = 0;
267     for (int i = len; i > 0; i--) {
268         r = r * w + num[i];
269         if (r < b) num[i] = 0;
270         else num[i] = r / b, r %= b;
271     }
272     while (len > 1 && num[len] == 0) len--;
273 }
274
275 void bign::operator%= (const int &b) {
276     ll r = 0;
277     for (int i = len; i > 0; i--) {

```

```

278         r = r * w + num[i];
279         if (r >= b) r %= b;
280     }
281     *this = bign(r);
282 }
283
284 istream& operator>> (istream &in, bign &res) {
285     in >> res.str;
286     res.change();
287     return in;
288 }
289 ostream& operator<< (ostream &out, const bign &res) {
290     res.print();
291     return out;
292 }
293
294 int main() {
295     return 0;
296 }

```

## 4. 差分

```

1  const int M = 1e5 + 10;
2  int dif[M], arr[M], res[M];
3
4  void change(int l, int r, int v) { // [l, r] + v
5      dif[l] += v, dif[r + 1] -= v;
6  }
7  void init(int n) {
8      dif[1] = arr[1];
9      for (int i = 2; i <= n; ++i) dif[i] = arr[i] - arr[i - 1];
10 }
11 void get_res(int n) {
12     res[1] = dif[1];
13     for (int i = 2; i <= n; ++i) {
14         res[i] = dif[i] + res[i - 1];
15     }
16 }

```

## 5.三分

```

1  double ts(int l, int r) {
2      double ans1 = 0x3f3f3f3f, ans2 = 0x3f3f3f3f;
3      while (l < r) {
4          int lmid = l + (r - l) / 3;
5          int rmid = r - (r - l) / 3;
6          ans1 = f(lmid), ans2 = f(rmid);
7          if (ans1 <= ans2) r = rmid - 1; // 凹函数的最小值, 凸函数<=反过来
8          else l = lmid + 1;
9      }
10     return min(ans1, ans2); //凸函数这里记得改成max
11 }

```

## 二、数论

### 1. gcd与lcm

```

1  typedef long long ll;
2  //最大公因数, 公约数
3  ll gcd(ll a, ll b, ll m = 1) { while(b) m = a % b, a = b, b = m; return a; }
4  //最小公倍数
5  ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }

```

### 2. ex\_gcd

```

1  /*
2  通解为
3  x' = x * c / gcd + (b / gcd) * k
4  y' = y * c / gcd - (a / gcd) * k
5  */
6  typedef long long ll;
7  // 扩展欧几里得算法核心函数
8  void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
9      if (!b) {
10         g = a, x = 1, y = 0;

```

```

11         return;
12     }
13     exgcd(b, a % b, g, y, x);
14     y -= x * (a / b);
15 }
16 /**
17  * 此函数为 求解  $ax + by = c$ 
18  * 返回 x 的最小正整数解
19  * 返回 -1 说明无解
20  */
21 ll minx(ll a, ll b, ll c) {
22     ll x, y, g;
23     exgcd(a, b, g, x, y);
24     if (c % g != 0) return -1;
25     ll t = abs(b / g);
26     x = (x * c / g % t + t) % t;
27     return x == 0 ? x + t : x; // 返回最小正整数解
28 }

```

### 3. 素数筛

- 埃式筛

```

1  const int M = 101000;
2  int pri[M], cnt = 0;
3  bool isp[M];
4  // 复杂度  $O(n \log n)$ 
5  // true 为非素数, false 为素数
6  void table() {
7      isp[0] = isp[1] = true;
8      for (int i = 2; i < M; i++) {
9          if (isp[i]) continue;
10         pri[cnt++] = i;
11         for (int j = i + i; j < M; j += i) isp[j] = true;
12     }
13 }

```

- 线性筛

```

1  // 复杂度  $O(n)$ 
2  const int M = 1e5 + 10;
3  int pri[M], cnt = 0;
4  bool isp[M];
5  // true 为非素数, false 为素数
6  void table() {
7      isp[0] = isp[1] = 1;
8      for (int i = 2; i < M; i++) {
9          if (!isp[i]) pri[cnt++] = i;
10         for (int j = 0; j < cnt && i * pri[j] < M; j++) {
11             isp[i * pri[j]] = 1;
12             if (!(i % pri[j])) break;
13         }
14     }
15 }

```

### 4. 逆元

1. 线性法

```

1  typedef long long ll;
2  // 时间复杂度  $O(n)$ 
3  void fny(const int &n, ll *inv, const ll mod) {
4      inv[0] = inv[1] = 1;
5      for (ll i = 2; i <= n; i++) {
6          inv[i] = ((mod - mod / i) * inv[mod % i]) % mod;
7      }
8  }

```

2. 扩欧法

```

1  typedef long long ll;
2  void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
3      if (!b) {
4          g = a, x = 1, y = 0;
5          return;
6      }
7      exgcd(b, a % b, g, y, x);
8      y -= x * (a / b);

```

```

9   }
10
11  // ax=1(mod m)
12  ll inverse(ll a, ll m) { //扩展欧几里得法求逆元, 返回-1代表没有逆元
13      ll g, x, y;
14      exgcd(a, m, g, x, y);
15      return g == 1 ? (x % m + m) % m : -1;
16  }

```

## 2. 费马小定理法

```

1   //a ^ (p - 1) = 1 (mod p), p为素数
2   //a ^ (p - 2) = a ^ (-1) (mod p)
3   //a 的逆元为 a ^ (p - 2)
4   typedef long long ll;
5   ll pow_f(ll a, ll b, const ll mo) {
6       ll ans = 1;
7       a %= mo;
8       while (b) {
9           if (b & 1) ans = (ans * a) % mo;
10          a = (a * a) % mo;
11          b >>= 1;
12      }
13      return ans;
14  }
15
16  ll inverse(ll a, ll m) {
17      return pow_f(a, m - 2, m);
18  }

```

## 5. 快速幂

```

1   #define ll long long
2   ll powf(ll a, ll b, const ll mod) { // 返回a^b % mod
3       a %= mod;
4       ll res = 1;
5       while (b) {
6           if (b & 1) res = res * a % mod;
7           a = a * a % mod, b >>= 1;
8       }
9       return res;
10  }

```

## 6. 矩阵快速幂

```

1   const ll MOD = 1e9 + 7;
2   #define MO(x) ((x) % MOD)
3
4   struct Mat {
5       ll mat[10][10];
6       int n; // n * n 阶矩阵
7       Mat(int n = 2) : n(n) { memset(mat, 0, sizeof mat); } //记得修改
8       void to_one() {
9           for (int i = 0; i < n; i++)
10              mat[i][i] = 1;
11      }
12      Mat operator*(const Mat a) const {
13          Mat res;
14          for (int i = 0; i < n; i++) {
15              for (int j = 0; j < n; j++) {
16                  ll sum = 0;
17                  for (int k = 0; k < n; k++) {
18                      sum += MO(this->mat[i][k] * a.mat[k][j]);
19                  }
20                  res.mat[i][j] = MO(sum);
21              }
22          }
23          return res;
24      }
25      ll *operator[](int x) { return mat[x]; }
26  };
27
28
29  Mat pow_f(Mat a, ll b) { //a ^ b次幂
30      Mat ans;
31      ans.to_one();
32      while (b) {
33          if (b & 1)
34              ans = ans * a;
35          a = a * a;

```

```

36     b >>= 1;
37 }
38 return ans;
39 }

```

## 7. 高斯消元

### 1. 普通浮点数高斯消元，洛谷模板题

```

1 // 洛谷模板题
2 #include <algorithm>
3 #include <cstdio>
4 #include <vector>
5
6 using namespace std;
7 const int N = 110;
8 const double eps = 1e-6; // 用来控制精度
9 // 普通的高斯消元是将矩阵转化成上三角的形式，再回带求出答案
10 double ans[N]; // 用来记录答案
11 int gauss(int n, int m, vector<vector<double>> &a) { // n行m + 1列的增广矩阵
12     int r, c; // 当前行和当前列
13     for (r = c = 0; c < m && r < n; ++c) {
14         int maxr = r; // 记录最大
15         for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
16             maxr = i; // 寻找从当前行开始向下走的当前列中的绝对值最大值
17         if (r ^ maxr) swap(a[r], a[maxr]); // 如果不是当前行，则交换两行
18         if (abs(a[r][c]) < eps) continue; // 如果当前行当前列的最大值为0则不作消元
19         for (int i = m; i >= c; --i) a[r][i] /= a[r][c]; // 将当前行的从当前列开始到最后一列
20         for (int i = r + 1; i < n; ++i) {
21             if (abs(a[i][c]) < eps) continue; // 如果改行的当前列已经是0，则不作消元
22             for (int j = m; j >= c; --j) { // 逆向消元，可以少开一个变量
23                 a[i][j] -= a[r][j] * a[i][c];
24             }
25         }
26         ++r;
27     }
28     if (r < n) { // 无穷解 或者 无解
29         for (int i = r; i < n; ++i) if (abs(a[i][n]) < eps) return 0;
30         return -1; // 无穷解
31     }
32     for (int i = n - 1; ~i; --i) { // 回带
33         for (int j = i + 1; j < n; ++j) {
34             a[i][n] -= a[i][j] * ans[j];
35         }
36         ans[i] = a[i][n];
37     }
38     return 1; // 唯一解
39 }
40 vector<vector<double>> a;
41 int main() {
42     int n;
43     scanf("%d", &n);
44     for (int i = 0; i < n; ++i) {
45         a.push_back({});
46         for (int j = 0; j <= n; ++j) {
47             double x;
48             scanf("%lf", &x);
49             a[i].push_back(x);
50         }
51     }
52     if (gauss(n, n, a) == 1) {
53         for (int i = 0; i < n; ++i) printf("%.2f\n", ans[i]);
54     } else puts("No Solution");
55     return 0;
56 }

```

### 2. 浮点数高斯约旦消元法，洛谷模板题

```

1 // 洛谷模板题
2 #include <cstdio>
3 #include <cmath>
4 #include <vector>
5
6 using namespace std;
7 const double eps = 1e-6;
8 double ans[110]; // 记录答案
9 int gauss_j(int n, int m, vector<vector<double>> &a) { // n行m + 1列增广矩阵
10     int r, c; // 当前行当前列
11     for (r = c = 0; c < m && r < n; ++c) { //
12         int maxr = r; // 记录最大值

```



```

13     for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
14         maxr = i; // 寻找从当前行开始向下走的当前列中的绝对值最大值
15     if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
16     if (abs(a[r][c]) < eps) continue; // 如果为当前行中的当前列的值为0
17     for (int i = 0; i < n; ++i) { // 约旦消元
18         if (abs(a[i][c]) < eps || i == r) continue; // 如果是当前行或者改行的当前列已经是0
19         for (int j = m; j >= c; --j) a[i][j] -= a[i][c] / a[r][c] * a[r][j];
20     }
21     ++r;
22 }
23 if (r < n) return 0; // 无解或者无穷解
24 for (int i = 0; i < n; ++i) ans[i] = a[i][n] / a[i][i]; // 回带
25 return 1;
26 }
27
28 vector<vector<double>> a;
29 int main() {
30     int n;
31     scanf("%d", &n);
32     for (int i = 0; i < n; ++i) {
33         a.push_back({});
34         for (int j = 0; j <= n; ++j) {
35             double x;
36             scanf("%lf", &x);
37             a[i].push_back(x);
38         }
39     }
40     if (gauss_j(n, n, a) == 1) {
41         for (int i = 0; i < n; ++i) printf("%.2f\n", ans[i]);
42     } else puts("No Solution");
43     return 0;
44 }

```

### 3. 模意义下的高斯消元法, POJ - 2065 SETI

```

1 // POJ - 2065 SETI
2 #include <cassert>
3 #include <vector>
4 #include <cstdio>
5 #include <algorithm>
6 #include <cstring>
7
8 using namespace std;
9
10 int gcd(int a, int b) { int m; while(b) m = a % b, a = b, b = m; return a; }
11 int lcm(int a, int b) { return a / gcd(a, b) * b; }
12 int p, ans[310]; // 记录答案
13
14 int powf(int a, int b, const int mod, int ans = 1) {
15     a %= mod;
16     while (b) {
17         if (b & 1) ans = ans * a % mod;
18         b >>= 1, a = a * a % mod;
19     }
20     return ans;
21 }
22 int inv(int a, int m) {
23     return powf(a, m - 2, m);
24 }
25 // 模意义下的高斯消元不需要用约旦的方式, 因为整数不用考虑精度问题
26 // n 行 m + 1 列的增广矩阵, 从0开始
27 int gauss(int n, int m, vector<vector<int>> &a, const int &p) { // 传入模p
28     int r, c; // 当前行和当前列
29     for (r = c = 0; c < m && r < n; ++c) {
30         int maxr = r; // 记录最大值
31         for (int i = r + 1; i < n; ++i) if (abs(a[i][c]) > abs(a[maxr][c]))
32             maxr = i; // 寻找当前列中从当前行开始的绝对值的最大值
33         if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
34         if (!a[r][c]) continue; // 如果为0
35         for (int i = r + 1; i < n; ++i) {
36             if (!a[i][c]) continue; // 如果当前列中 改行已经为0
37             int LCM = lcm(abs(a[i][c]), abs(a[r][c]));
38             int x = LCM / abs(a[i][c]), y = LCM / abs(a[r][c]); // 使该行乘x, 当前行乘y使得他们在当前列的数都变成同一
39             个数
40             if (a[i][c] * a[r][c] < 0) y = -y; // 如果有一个是负数
41             for (int j = c; j <= m; ++j)
42                 a[i][j] = ((a[i][j] * x - a[r][j] * y) % p + p) % p;
43         }
44         ++r;
45     }
46     for (int i = r; i < n; ++i) if (a[i][c]) return 0; // 无解
47     if (r < m) return -1; // 无穷解
48     for (int i = m - 1; ~i; --i) {
49         int tmp = a[i][m];
50         for (int j = i + 1; j < m; ++j) {

```

```

50         if (!a[i][j]) continue;
51         tmp -= ans[j] * a[i][j];
52         tmp = ((tmp % p) + p) % p;
53     }
54     ans[i] = tmp * inv(a[i][i], p) % p; // 求逆元可以换成别的方式求
55 }
56 return 1;
57 }
58
59 vector<vector<int>> mat;
60 char str[110];
61 int main() {
62     int t;
63     scanf("%d", &t);
64     while (t--) {
65         mat.clear();
66         scanf("%d %s\n", &p, str + 1);
67         int n = strlen(str + 1);
68         for (int i = 1; i <= n; ++i) {
69             mat.push_back(vector<int>(n + 1, 0));
70             if (str[i] != '*') mat[i - 1][n] = str[i] - 'a' + 1;
71             for (int j = 0; j < n; ++j) mat[i - 1][j] = powf(i, j, p);
72         }
73         int res = gauss(n, n, mat, p);
74         for (int i = 0; i < n; ++i) printf("%d ", ans[i]);
75         puts("");
76     }
77     return 0;
78 }

```

#### 4. 异或的高斯消元法（带解决自由变元的），POJ1681 Painter's Problem

```

1 //POJ1681 Painter's Problem
2 #include <vector>
3 #include <cstdio>
4 #include <algorithm>
5
6 using namespace std;
7 const int N = 305;
8 int ans[N], freew[N]; // 记录答案和记录自由变元是哪些
9 int gauss(int n, int m, vector<vector<int>> &a) { // n行m + 1列的增广矩阵
10     int num = 0, r, c; // 自由变元的个数 当前行 当前列
11     for (int i = 0; i < n; ++i) ans[i] = freew[i] = 0; // 初始化答案和自由变元
12     for (r = c = 0; r < n && c < m; ++c) {
13         int maxr = r; // 记录最大值
14         for (int i = r + 1; i < n && !a[maxr][c]; ++i) maxr = i; // 查找1
15         if (maxr ^ r) swap(a[r], a[maxr]); // 交换两行
16         if (!a[r][c]) { // 如果这个数是0，则说明第c个变元是自由变元
17             freew[num++] = c; // 记录自由变元
18             continue;
19         }
20         for (int i = r + 1; i < n; ++i) { // 消元，消成一个上三角
21             if (!a[i][c]) continue; // 如果该行的当前列是0则该行不作消元
22             for (int j = m; j >= c; --j) a[i][j] ^= a[r][j]; // 直接异或，和加法不同
23         }
24         ++r;
25     }
26     for (int i = r; i < n; ++i) if (a[i][m]) return -1; // 无解
27     int cnt = 1 << (n - r), ret = 1 << 29; // 自由变元的取值方案数，记录最小操作数
28     for (int i = 0; i < cnt; ++i) { // 状态压缩求解
29         int res = 0, indx = i; // 当前方案的最小操作数，操作方案
30         for (int j = 0; j < num; ++j, indx >>= 1) {
31             ans[freew[j]] = indx & 1;
32             res += indx & 1; // 如果是1则说明这个地方需要操作，固操作数+1
33         }
34         for (int j = r - 1; j >= 0; --j) {
35             ans[j] = a[j][m]; // 记录答案
36             for (int k = j + 1; k < m; ++k) {
37                 if (a[j][k]) ans[j] ^= ans[k]; // 如果这个数的系数不为0
38             }
39             res += ans[j]; // 增加操作数
40         }
41         ret = min(res, ret); // 取最小值
42     }
43     return ret;
44 }
45
46 vector<vector<int>> mat;
47 int n;
48 const int dx[] = {0, 1, 0, -1, 0};
49 const int dy[] = {0, 0, 1, 0, -1};
50 char g[30][30];
51 bool check(int r, int c) { return r >= 0 && c >= 0 && r < n && c < n; }
52 int main() {
53     int _;

```

```

54     scanf("%d", &_);
55     for (int t = 1; t <= _; ++t) {
56         mat.clear();
57         scanf("%d", &n);
58         mat.resize(n * n, vector<int>(n * n + 1, 0));
59         for (int i = 0, r = 0; i < n; ++i) {
60             for (int j = 0; j < n; ++j, ++r) {
61                 for (int k = 0; k < 5; ++k) {
62                     int nr = i + dx[k], nc = j + dy[k];
63                     if (check(nr, nc)) mat[r][nr * n + nc] = 1;
64                 }
65             }
66         }
67         for (int i = 0, r = 0; i < n; ++i) {
68             scanf("%s", g[i]);
69             for (int j = 0; j < n; ++j, ++r) {
70                 if (g[i][j] == 'w') mat[r][n * n] = 1;
71             }
72         }
73         int res = gauss(n * n, n * n, mat);
74         printf("%d\n", res);
75     }
76     return 0;
77 }

```

## 8. lucas

```

1  const ll mod = 10007;
2  ll fac[mod + 10], inv[mod + 10];
3  void fny(const int &n, ll *inv, const ll mod) {
4      fac[0] = fac[1] = inv[0] = inv[1] = 1;
5      for (ll i = 2; i <= n; i++) {
6          inv[i] = ((mod - mod / i) * inv[mod % i]) % mod;
7          fac[i] = fac[i - 1] * i % mod;
8      }
9  }
10
11 ll comb(ll n, ll m) {
12     if (m > n) return 0;
13     return fac[n] * inv[fac[n - m] * fac[m] % mod] % mod;
14 }
15
16
17 ll lucas(ll n, ll m) {
18     if (m == 0) return 1;
19     if (n < mod) return comb(n, m);
20     return comb(n % mod, m % mod) * lucas(n / mod, m / mod) % mod;
21 }

```

## 9. 线性基

```

1  struct LB {
2      using ll = long long;
3      ll d[65], cnt, num; // cnt 原序列的个数, num 基的个数
4      bool re;
5      LB () : cnt(0), num(0), re(false) { memset(d, 0, sizeof d); }
6      // 添加一个数x
7      void add(ll x) {
8          ++cnt;
9          for (int i = 60; ~i && x; --i) {
10             if ((x >> i) & 1) {
11                 if (d[i] x ^ d[i];
12                 else d[i] = x, x = 0, ++num, re = 0;
13             }
14         }
15     }
16     // 询问是否能异或出x
17     bool check(ll x) {
18         for (int i = 60; ~i && x; --i) {
19             if ((x >> i) & 1) {
20                 if (d[i] x ^ d[i];
21                 else return true;
22             }
23         }
24         return false;
25     }
26     ll get_max() {
27         ll res = 0;
28         for (int i = 60; ~i; --i) {
29             if ((d[i] ^ res) > res) res ^= d[i];
30         }
31         return res;

```

```

32     }
33
34     // 求的是线性基的异或最小值，不是原序列，否则要特判是否为0
35     ll get_min() {
36         for (int i = 0; i <= 60; ++i) {
37             if (d[i]) return d[i];
38         }
39     }
40     void rebuild() {
41         for (int i = 0; i <= 60; ++i) {
42             for (int j = 0; j < i; ++j) {
43                 if ((d[i] >> j) & 1) {
44                     d[i] ^= d[j];
45                 }
46             }
47         }
48         re = true;
49     }
50     ll k_th(ll k) {
51         if (!re) rebuild();
52         if (k == 1 && num < cnt) return 0; // 如果异或得到0
53         if (num < cnt) --k;
54         ll res = 0;
55         for (int i = 0; i <= 60; ++i) {
56             if (d[i]) {
57                 if (k & 1) res ^= d[i];
58                 k >>= 1;
59             }
60         }
61         return res;
62     }
63 };

```

## 10. CRT

```

1  ll crt(ll *a, ll *b, int n) { // x % b[i] = a[i], 返回最小的x, b[i]中互质
2      ll mul = 1, ret = 0;
3      for (int i = 0; i < n; ++i) mul *= b[i];
4      for (int i = 0; i < n; ++i) {
5          ll minlcm = mul / b[i];
6          ll inv = inverse(minlcm, b[i]); // 求逆元
7          ret = (ret + minlcm * inv * a[i]) % mul;
8      }
9      return (ret + mul) % mul;
10 }

```

## 11. 欧拉函数

- 1. 线性 $O(n)$

```

1  void getphi() {
2      phi[1] = 1;
3      for (int i = 2; i < M; ++i) {
4          if (!isp[i]) pri[cnt++] = i, phi[i] = i - 1;
5          for (int j = 0; j < cnt && i * pri[j] < M; ++j) {
6              isp[i * pri[j]] = 1;
7              if (i % pri[j] == 0) {
8                  phi[i * pri[j]] = pri[j] * phi[i];
9                  break;
10             } else {
11                 phi[i * pri[j]] = (pri[j] - 1) * phi[i];
12             }
13         }
14     }
15 }

```

## 12. 求组合数

- 递推

```

1  ll comb(ll n, ll m) {
2      if (n < m) return 0;
3      ll ret = 1;
4      for (int i = 1; i <= m; ++i) ret *= (n - i + 1) / i;
5      return ret;
6  }

```

## 三、数据结构

## 1. 并查集

### 1. 简便的路径压缩版

```
1  const int Max = 1e5 + 10;
2  int fa[Max];
3  inline void init() { for (int i = 0; i < Max; i++) fa[i] = i; }
4
5  int findfa(int x) {
6      return x == fa[x] ? x : fa[x] = findfa(fa[x]);
7  }
8
9  void Un(int a, int b) {
10     int fa1 = findfa(a);
11     int fa2 = findfa(b);
12     if (fa1 != fa2) fa[fa1] = fa2;
13 }
```

### 2. 网络赛版

```
1  class UF {
2  public:
3      vector<int> parent;
4      vector<int> size;
5      int n;
6      // 当前连通分量数目
7      int cnt;
8
9  public:
10     UF(int _n): n(_n), cnt(_n), parent(_n), size(_n, 1) {
11         int i = 0;
12         for (auto &x : parent) x = i++;
13     }
14
15     int findset(int x) {
16         return parent[x] == x ? x : parent[x] = findset(parent[x]);
17     }
18
19     bool unite(int x, int y) {
20         x = findset(x);
21         y = findset(y);
22         if (x == y) {
23             return false;
24         }
25         if (size[x] < size[y]) {
26             swap(x, y);
27         }
28         parent[y] = x;
29         size[x] += size[y];
30         --cnt;
31         return true;
32     }
33
34     bool conn(int x, int y) {
35         x = findset(x);
36         y = findset(y);
37         return x == y;
38     }
39 };
```

## 2. 树状数组

### 1. 单点修改与区间查询

```
1  // 修改复杂度与查询复杂度O(logn)
2  #define lb(x) ((x) & (-x))
3  #define ll long long
4  const int N = 1e6 + 5, M = 2e5 + 5;
5  ll n, m, a[N], bit[N];
6  // 初始化
7  void build(int n) {
8      for (int i = 1; i <= n; ++i) {
9          bit[i] += a[i];
10         int j = lb(i) + i;
11         if (j <= n) bit[j] += bit[i];
12     }
13 }
14 // 单点修改
15 void update(int index, ll val) {
16     a[index] += val;
17     while (index <= n) {
```

```

18     bit[index] += val;
19     index += lb(index);
20 }
21 }
22 // 前缀查询
23 ll get(int index) {
24     ll res = 0;
25     while (index) {
26         res += bit[index];
27         index -= lb(index);
28     }
29     return res;
30 }
31 // 区间和查询
32 ll get(int l, int r) { return get(r) - get(l - 1); }

```

## 2. 区间修改，区间查询

1. 设  $d[i] = a[i] - a[i - 1]$
2. 则  $a[x] = \sum_{i=1}^x d_i$
3. 设  $sum[x] = \sum_{i=1}^x a_i$ , 即  $sum[x] = d[1] + d[1] + d[2] + d[1] + d[2] + d[3] + \dots + d[1] + \dots + d[n]$
4. 化简得  $sum[x] = \sum_{i=1}^x d_i \times (n - i + 1)$
5. 得  $sum[x] = (n + 1) \times \sum_{i=1}^x d_i - \sum_{i=1}^x i \times d_i$
6. 固开两个树状数组，一个维护差分数组  $d_i$ ，一个维护  $i \times d_i$

```

1  #define lb(x) ((x) & (-x))
2  #define int long long
3  const int N = 5e3 + 5, M = 1e6 + 5;
4  int n, m, d[M], id[M];
5  // 基础树状数组单点更新
6  void update(int i, int val, int *bit) {
7      while (i <= n) {
8          bit[i] += val;
9          i += lb(i);
10     }
11 }
12 // 单点修改
13 void update(int i, int val) {
14     update(i, val, d), update(i, val * i, id);
15 }
16 // 区间修改
17 void update(int l, int r, int val) {
18     update(l, val, d), update(r + 1, -val, d);
19     update(l, l * val, id), update(r + 1, (-val) * (r + 1), id);
20 }
21 // 前缀查询
22 int get(int i, int *bit) {
23     int res = 0;
24     while (i) res += bit[i], i -= lb(i);
25     return res;
26 }
27 // 区间和查询
28 int get(int l, int r) {
29     int res = get(r, d) * (r + 1) - get(r, id);
30     res -= get(l - 1, d) * l - get(l - 1, id);
31     return res;
32 }

```

- 网络赛类封装版

```

1  template<class T>
2  class BIT {
3      #define lb(x) ((x) & (-x))
4      vector<T> sum, maxv, arr;
5      const int inf = 0x3f3f3f3f;
6      T getSum(int i) {
7          T ret = 0;
8          while (i) {
9              ret += sum[i];
10             i -= lb(i);
11         }
12         return ret;
13     }
14     int n;
15 public:
16     BIT(int n, T *a) : sum(n + 1, 0), maxv(n + 1, -inf), arr(n + 1), n(n) {
17         for (int i = 1; i <= n; ++i) arr[i] = a[i];
18         for (int i = 1; i <= n; ++i) {
19             sum[i] += a[i];           // 求和
20             maxv[i] = max(maxv[i], a[i]); // 最大值
21             int j = i + lb(i);
22             if (j <= n) {

```

```

23         sum[j] += sum[i];
24         maxv[j] = max(maxv[i], maxv[j]);
25     }
26 }
27 }
28 BIT(int n) : sum(n + 1, 0), maxv(n + 1, -inf), arr(n + 1, 0), n(n) {}
29 void update(int indx, T val) { // 将indx下标的数加v
30     int i = indx;
31     T &tmp = arr[indx];
32     tmp += val;
33     while (i <= n) {
34         sum[i] += val;
35         maxv[i] = arr[i];
36         for (int j = 1; j < lb(i); j <= 1) {
37             maxv[i] = max(maxv[i], maxv[i - j]);
38         }
39         i += lb(i);
40     }
41 }
42
43 T getSum(int l, int r) {
44     return getSum(r) - getSum(l - 1);
45 }
46
47 T getMax(int l, int r) {
48     T ret = arr[r];
49     while(r >= 1) {
50         ret = max(arr[r], ret);
51         for (r--; r - lb(r) >= 1; r -= lb(r)) ret = max(maxv[r], ret);
52     }
53     return ret;
54 }
55
56 T getMax(int i) { return getMax(1, i); }
57 void clear() {
58     fill(sum.begin(), sum.end(), 0);
59     fill(arr.begin(), arr.end(), 0);
60     fill(maxv.begin(), maxv.end(), -inf);
61 }
62 };

```

### 3. 线段树

- 精简版
- 结构体版

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int M = 1e6;
5  typedef long long ll;
6  template<class T>
7  struct Tree{
8      #define ls(node) (node << 1)
9      #define rs(node) ((node << 1) | 1)
10     Tree(int len = 10): len(len) {}
11     T sum[M << 2], lazy[M << 2], arr[M];
12     int len;
13
14     private:
15     void pushup(const int node) { // 写题目要求维护的代码，如求和，最大最小.....
16         sum[node] = max(sum[ls(node)], sum[rs(node)]);
17     }
18
19     void pushdown(int l, int r, const int node) { // 同pushup
20         int mid = (l + r) >> 1;
21         lazy[ls(node)] += lazy[node], lazy[rs(node)] += lazy[node];
22         sum[ls(node)] += (mid - l + 1) * lazy[node];
23         sum[rs(node)] += (r - mid) * lazy[node];
24         lazy[node] = 0;
25     }
26
27     void build(int l, int r, int node) {
28         if (l == r) {
29             sum[node] = arr[l];
30             return;
31         }
32         int mid = (l + r) >> 1;
33         build(l, mid, ls(node)), build(mid + 1, r, rs(node));
34         pushup(node);
35     }
36
37     void update(int ql, int qr, T v, int l, int r, int node) {
38         if (ql <= l && r <= qr) {

```

```

39         sum[node] += (r - l + 1) * v; //更新操作根据题目要求更改
40         lazy[node] += v;
41         return;
42     }
43     if (lazy[node]) pushdown(l, r, node);
44     int mid = (l + r) >> 1;
45     if (ql <= mid) update(ql, qr, v, l, mid, ls(node));
46     if (qr > mid) update(ql, qr, v, mid + 1, r, rs(node));
47     pushup(node);
48 }
49
50 T getAsk(int ql, int qr, int l, int r, int node) {
51     if (ql <= l && r <= qr) return sum[node]; //上同
52     if (lazy[node]) pushdown(l, r, node);
53     T res = 0;
54     int mid = (l + r) >> 1;
55     if (ql <= mid) res = getAsk(ql, qr, l, mid, ls(node));
56     if (qr > mid) res = max(res, getAsk(ql, qr, mid + 1, r, rs(node)));
57     return res;
58 }
59
60 public:
61     //以下为可以，直接调用的函数
62     void build() { build(1, len, 1); }
63     void update(int ql, int qr, T v) { update(ql, qr, v, 1, len, 1); }
64     void update(int index, T v) { update(index, index, v, 1, len, 1); }
65     T getAsk(int ql, int qr) { return getAsk(ql, qr, 1, len, 1); }
66     void clear() {
67         memset(arr, 0, sizeof(T) * (len + 10));
68         memset(sum, 0, sizeof(T) * (len << 2));
69         memset(lazy, 0, sizeof(T) * (len << 2));
70     }
71     T& operator[] (int x) { return arr[x]; }
72 };
73
74 int main(){
75     return 0;
76 }

```

- 类版-网络赛

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5
6  /**
7   * 使用说明:
8   * 本线段树板子可用于多数 网络赛
9   * 需要注意的是此板子在性能方面由于三层封装，固较简洁版的线段树性能慢，但灵活性高（适合网络赛抢时间）
10  * 使用时只需要修改 “可修改区的上下界” 中间的代码部分
11  * lazy使用结构体封装是应付在某些题目在区间修改时有两种操作，下面给的例题有示例
12  * vals是最终题目需要维护的东西（sum, max, min.....），针对不同的操作只需修改结构体内的加法（+）操作即可
13  * 同时若题目需要区间修改，则直接修改区间修改部分的代码即可，单点修改同理
14  * 注意：区间修改针对不同的操作，lazy中的加等于（+=）也要具体根据题意修改
15  * 例题1：洛谷
16  */
17  /*****可修改区上界*****/
18  template<class T>
19  struct lazy{
20      T lazy1;
21      lazy(T lazy1 = 0){
22          this->lazy1 = lazy1;
23      }
24      void operator+= (const lazy<T> &b) {
25          this->lazy1 += b.lazy1; //常见操作
26      }
27      inline void del() {
28          this->lazy1 = 0; // 常见操作
29      }
30  };
31  template<class T>
32  struct vals{
33      T sum;
34      //初始化构造，用来方便做运算的初始化-
35      vals(int x = 0) {
36          sum = x; //常见初始化，方便后面做 + 的操作
37      }
38      // 操作区
39      constexpr vals operator+ (const vals& b){
40          vals res;
41          res.sum = this->sum + b.sum; //常见操作
42          return res;
43      }
44      //建树初始化区

```



```

45     inline void init(const T &v) {
46         // this->sum = v; // 常见操作, 即和本身一样
47     }
48     // 区间更新区
49     inline void update(const int &l, const int &r, const lazy<T> &v) {
50         // this->sum += (r - l + 1) * v.lazy1; // 常见操作
51         // printf("%d %d %d %d\n", l, r, v, (r - l + 1) * v); // 常见操作
52     }
53     // 单点更新区
54     inline void update(const lazy<T> &v) {
55         // this->sum += v; // 常见操作
56     }
57     // 初始化, 通常在多组输入题时才调用
58     inline void del() {
59         this->sum = 0;
60     }
61 };
62 /*****可修改区下界*****/
63
64 /*****以下区域建议不动 (除特殊题目) *****/
65 template<class T>
66 class segtree{
67 private:
68     const static int maxn = 1e5 + 10;
69     const int start = 1;
70     struct lazy_tabs{
71         lazy<T> v;
72         bool flag;
73         lazy_tabs (bool flag = false) : flag(flag) {}
74         // 懒标记更新
75         void update(const lazy<T> &v) { this->v += v, flag = true; }
76         // 懒标记删除
77         void del() { this->v.del(), flag = false; }
78     }lazy_tab[maxn << 2];
79     vals<T> val[maxn << 2];
80     // 计算左右孩子节点下标
81     inline static constexpr int ls(const int &indx) { return indx << 1; }
82     inline static constexpr int rs(const int &indx) { return indx << 1 | 1; }
83     // 上放
84     inline void push_up(const int &pos) { val[pos] = val[ls(pos)] + val[rs(pos)]; }
85     // 下放
86     void push_down(const int &l, const int &r, const int &pos) {
87         if (!lazy_tab[pos].flag) return; // 若没有标记则返回
88         const auto &v = lazy_tab[pos].v;
89         lazy_tab[ls(pos)].update(v), lazy_tab[rs(pos)].update(v);
90         const int mid = (l + r) >> 1;
91         val[ls(pos)].update(l, mid, v), val[rs(pos)].update(mid + 1, r, v); // 更新儿子节点
92         lazy_tab[pos].del();
93     }
94     // 建树
95     void build(const int &l, const int &r, const int &pos, const T *a) {
96         if (l == r) {
97             val[pos].init(a[l]);
98             return;
99         }
100         const int mid = (l + r) >> 1;
101         build(l, mid, ls(pos), a), build(mid + 1, r, rs(pos), a);
102         push_up(pos);
103     }
104     // 单点修改
105     void update(const int &indx, const int &l, const int &r,
106                const lazy<T> &v, const int &pos) {
107         if (l == r) {
108             val[pos].update(v);
109             return;
110         }
111         const int mid = (l + r) >> 1;
112         if (indx <= mid) update(indx, l, mid, ls(pos));
113         else update(indx, mid + 1, r, rs(pos));
114         push_up(pos);
115     }
116     // 区间修改
117     void update(const int &q1, const int &q2, const lazy<T> &v,
118                const int &l, const int &r, const int &pos) {
119         if (q1 <= l && r <= q2) { // 如果在查询区间内
120             lazy_tab[pos].update(v); // 懒更新
121             val[pos].update(l, r, v); // 区间值更新
122             return;
123         }
124         push_down(l, r, pos);
125         const int mid = (l + r) >> 1;
126         if (q1 <= mid) update(q1, q2, v, l, mid, ls(pos));
127         if (q2 > mid) update(q1, q2, v, mid + 1, r, rs(pos));
128         push_up(pos);
129     }
130     // 区间查询
131     vals<T> get(const int &q1, const int &q2, const int &l,

```

```

132         const int &r, const int &pos) {
133     if (ql <= 1 && r <= qr) return val[pos];
134     push_down(1, r, pos);
135     vals<T> res;
136     const int mid = (1 + r) >> 1;
137     if (ql <= mid) res = res + get(ql, qr, 1, mid, ls(pos));
138     if (qr > mid) res = res + get(ql, qr, mid + 1, r, rs(pos));
139     return res;
140 }
141 public:
142     /*****一下均可调用*****/
143     int n;
144     segtree (int n = -1) : n(n) {}
145     // 建树
146     void build(T *a) { build(start, n, start, a); }
147     // 查询
148     vals<T> get(int ql, int qr) { return get(ql, qr, start, n, start); }
149     // 单点修改
150     void update(int indx, lazy<T> val) { update(indx, start, n, val, start); }
151     // 区间修改
152     void update(int ql, int qr, lazy<T> val) { update(ql, qr, val, start, n, start); }
153     // 清空-
154     // 切记，先用再清空，因为通常是多组输入时太调用
155     void clear() {
156         assert(n != -1);
157         int len = n << 2;
158         assert(n < maxn);
159         for (int i = 1; i <= len; ++i) {
160             lazy_tab[i].del();
161             val[i].del();
162         }
163     }
164 };
165
166 int main() {
167     #ifndef ONLINE_JUDGE
168         freopen("D:/MYCODE/vsCode-c/test.in", "r", stdin);
169         freopen("D:/MYCODE/vsCode-c/test.out", "w", stdout);
170     #endif
171     return 0;
172 }

```

## 4. ST表

```

1  const int M = 1e5 + 5;
2  int st[M][30], lg[M];
3  // st表预处理，注意下标从1开始到n结束
4  void init(int *a, int n) {
5      lg[0] = -1;
6      for (int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1, st[i][0] = a[i];
7      for (int j = 1; j <= lg[n]; ++j) {
8          int k = 1 << (j - 1);
9          for (int i = 1; i + k - 1 <= n; ++i) {
10             st[i][j] = max(st[i][j - 1], st[i + k][j - 1]);
11         }
12     }
13 }
14 // 询问
15 // 尽可能让l + 2^(len) - 1接近r
16 int get(int l, int r) {
17     int x = lg[r - l + 1];
18     return max(st[l][x], st[r - (1 << x) + 1][x]);
19 }

```

## 5. 分块

```

1  const int N = 1e5 + 5, M = 500;
2  #define ll long long
3  ll a[N];
4  int belong[N];
5  struct blocks {
6      int l, r;
7      ll lazy;
8      blocks() : lazy(0){}
9  }b[M];
10 // 以下函数是基本不变的
11 void build(int n) {
12     int siz = sqrt(n), cnt = n / siz;
13     if (n % siz) ++cnt;
14     for (int i = 1; i <= cnt; ++i) {
15         b[i].l = (i - 1) * siz + 1;
16         b[i].r = i * siz;

```

```

17     }
18     b[cnt].r = n;
19     for (int i = 1; i <= n; ++i) belong[i] = (i - 1) / siz + 1;
20 }
21

```

## 6. 莫队

```

1  #include <cstdio>
2  #include <cstring>
3  #include <cmath>
4  #include <algorithm>
5
6  using namespace std;
7
8  const int M = 1e5 + 10;
9  int n, m, block, arr[M], pos[M], ans[M], res;
10 struct MO{
11     int l, r, k;
12     MO(int l = 0, int r = 0, int k = 0) : l(l), r(r), k(k) {}
13 }q[M];
14
15 bool cmp(MO a, MO b) {
16     if (pos[a.l] ^ pos[b.l]) { //不在同一个块
17         return pos[a.l] < pos[b.l];
18     }
19     if (pos[a.l] & 1) return a.r < b.r;
20     return b.r < a.r;
21 }
22 void add(int x) {
23
24 }
25 void del(int x) {
26
27 }
28
29 void solve() {
30     int l = 1, r = 0;
31     for (int i = 0; i < m; i++) {
32         while (l > q[i].l) add(--l);
33         while (l < q[i].l) del(l++);
34         while (r < q[i].r) add(++r);
35         while (r > q[i].r) del(r--);
36         ans[q[i].k] = res; //res根据题目意思来
37     }
38 }
39 void init() {
40     scanf("%d %d", &n, &m);
41     block = sqrt(n);
42     for (int i = 1; i <= n; i++) {
43         scanf("%d", arr + i);
44         pos[i] = i / block;
45     }
46     for (int i = 0; i < m; i++) {
47         int l, r;
48         scanf("%d %d", &l, &r);
49         q[i] = MO(l, r, i);
50     }
51     sort(q, q + m, cmp);
52 }
53
54 int main() {
55     init();
56     solve();
57     return 0;
58 }

```

## 7. 平衡树

### 1. fhq treap

```

1  // 洛谷板子题
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5  #include <random>
6  #include <cctype>
7  inline long long IO() {
8      long long x = 0;
9      bool f = false;
10     char c = getchar();
11     while (!isdigit(c)) {

```

```

12     if (c == '-') f = true;
13     c = getchar();
14 }
15 while (isdigit(c)) {
16     x = (x << 1) + (x << 3) + (c - '0');
17     c = getchar();
18 }
19 return f ? -x : x;
20 }
21 using namespace std;
22 const int N = 4e5 + 10;
23 mt19937 rnd(233);
24 struct treap{
25     int val, l, r, size, key;
26 }fhq[N];
27 int root, cnt;
28 inline void update(int now) {
29     fhq[now].size = fhq[fhq[now].l].size + fhq[fhq[now].r].size + 1;
30 }
31
32 int new_node(int val) {
33     fhq[++cnt] = {val, 0, 0, 1, rnd()};
34     return cnt;
35 }
36 void split(int now, int val, int &x, int &y) {
37     if (!now) { x = y = 0; return; }
38     if (fhq[now].val <= val) x = now, split(fhq[now].r, val, fhq[now].r, y);
39     else y = now, split(fhq[now].l, val, x, fhq[now].l);
40     update(now);
41 }
42
43 int merge(int x, int y) {
44     if (!x || !y) return x | y;
45     // 大根堆
46     if (fhq[x].key > fhq[y].key) { //右下角
47         fhq[x].r = merge(fhq[x].r, y), update(x);
48         return x;
49     }
50     // 左下角
51     fhq[y].l = merge(x, fhq[y].l), update(y);
52     return y;
53 }
54 // 插入
55 inline void insert(int val) {
56     int x, y;
57     split(root, val, x, y);
58     root = merge(merge(x, new_node(val)), y);
59 }
60 // 按值删除
61 inline void del(int val) {
62     int x, y, z;
63     split(root, val, x, z);
64     split(x, val - 1, x, y);
65     y = merge(fhq[y].l, fhq[y].r);
66     root = merge(merge(x, y), z);
67 }
68 // 按值获取排名
69 inline int getrank(int val) {
70     int x, y, ans;
71     split(root, val - 1, x, y);
72     ans = fhq[x].size + 1;
73     root = merge(x, y);
74     return ans;
75 }
76 // 按排名获取值
77 inline int getval(int rank) {
78     int now = root;
79     while (now) {
80         if (fhq[fhq[now].l].size + 1 == rank) break;
81         else if (fhq[fhq[now].l].size >= rank) now = fhq[now].l;
82         else rank -= fhq[fhq[now].l].size + 1, now = fhq[now].r;
83     }
84     return fhq[now].val;
85 }
86 // 求前驱, 即严格比val小的最大值
87 inline int pre(int val) {
88     int x, y;
89     split(root, val - 1, x, y);
90     int now = x;
91     while (fhq[now].r) now = fhq[now].r;
92     root = merge(x, y);
93     return fhq[now].val;
94 }
95 // 求后继, 即严格比val大的最小值
96 inline int nxt(int val) {
97     int x, y;
98     split(root, val, x, y);

```

```

99     int now = y;
100     while (fhq[now].l) now = fhq[now].l;
101     root = merge(x, y);
102     return fhq[now].val;
103 }
104
105 int main() {
106     int t = IO();
107     while (t--) {
108         int q = IO(), val = IO();
109         if (q == 1) insert(val);
110         else if (q == 2) del(val);
111         else if (q == 3) printf("%d\n", getrank(val));
112         else if (q == 4) printf("%d\n", getval(val));
113         else if (q == 5) printf("%d\n", pre(val));
114         else printf("%d\n", nxt(val));
115     }
116     return 0;
117 }

```

2. spaly
3. 替罪羊

## 8. 左偏树

## 9. 主席树

1. 主席树（静态）洛谷模板题

```

1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #include <vector>
5  #include <cctype>
6  inline long long IO() {
7      long long x = 0;
8      bool f = false;
9      char c = getchar();
10     while (!isdigit(c)) {
11         if (c == '-') f = true;
12         c = getchar();
13     }
14     while (isdigit(c)) {
15         x = (x << 1) + (x << 3) + (c - '0');
16         c = getchar();
17     }
18     return f ? -x : x;
19 }
20 using namespace std;
21
22 /*****离散化*****/
23 // vt存放可用于查询原本的数（用离散化值），打表后用于查询离散化表（用下标）
24 vector<int> vt;
25 inline int get_id(const int &x) { return lower_bound(vt.begin(), vt.end(), x) - vt.begin() + 1; }
26 inline void erase_vt() {
27     sort(vt.begin(), vt.end());
28     vt.erase(unique(vt.begin(), vt.end()), vt.end());
29 }
30 // 打表，注意，原数组下标要从1开始，返回离散化后的表大小
31 inline int id_table(int n, int *a, vector<int> &res) {
32     res.emplace_back(0);
33     for (int i = 1; i <= n; ++i) res.emplace_back(get_id(a[i]));
34     return vt.size();
35 }
36
37 /*****主席树*****/
38 const int N = 2e5 + 5;
39 struct nodes{
40     int l, r, sum;
41     nodes() : sum(0) {}
42 }hjt[N << 5];
43 int root[N], cnt; // 记录每个根结点的内存池编号，内存池
44 int build(int l, int r) {
45     int now = ++cnt; // 内存申请
46     if (l < r) {
47         int mid = (l + r) >> 1;
48         hjt[now].l = build(l, mid);
49         hjt[now].r = build(mid + 1, r);
50     }
51     return now;
52 }
53 // 插入新节点的操作
54 int update(int pre, int l, int r, int x) {
55     int now = ++cnt; // 内存申请

```

```

56     hjt[now] = hjt[pre], ++hjt[now].sum; // 继承
57     if (l < r) { // 寻找拼接点
58         int mid = (l + r) >> 1;
59         if (x <= mid) hjt[now].l = update(hjt[now].l, l, mid, x); // 如果x在左边，则让当前新节点的左孩子继承后的左孩
子
60         else hjt[now].r = update(hjt[now].r, mid + 1, r, x); // 否则同理
61     }
62     return now;
63 }
64 // 返回第qr版本的主席树 - 第ql版本的主席树，注意返回的是离散化后的值
65 int get(int ql, int qr, int l, int r, int k) {
66     if (l == r) return l;
67     int mid = (l + r) >> 1;
68     int dif = hjt[hjt[qr].l].sum - hjt[hjt[ql].l].sum;
69     if (k <= dif) return get(hjt[ql].l, hjt[qr].l, l, mid, k); // 左孩子上
70     return get(hjt[ql].r, hjt[qr].r, mid + 1, r, k - dif); // 右孩子上
71 }
72
73 /*****主函数*****/
74 int a[N];
75 int main() {
76     int n = IO(), m = IO();
77     for (int i = 1; i <= n; ++i) a[i] = IO(), vt.emplace_back(a[i]);
78     erase_vt();
79     vector<int> id;
80     int siz = id_table(n, a, id);
81     root[0] = build(1, siz);
82     for (int i = 1; i <= m; ++i) root[i] = update(root[i - 1], 1, siz, id[i]);
83     while (m--) {
84         int l = IO(), r = IO(), k = IO();
85         printf("%d\n", vt[get(root[l - 1], root[r], 1, siz, k) - 1]);
86     }
87     return 0;
88 }

```

## 10. LCA

```

1 // 洛谷板子题
2 #include <cstdio>
3 #include <algorithm>
4 #include <cstring>
5 #include <cctype>
6 #define ll long long
7 inline long long IO() {
8     long long x = 0;
9     bool f = false;
10    char c = getchar();
11    while (!isdigit(c)) {
12        if (c == '-') f = true;
13        c = getchar();
14    }
15    while (isdigit(c)) {
16        x = (x << 1) + (x << 3) + (c - '0');
17        c = getchar();
18    }
19    return f ? -x : x;
20 }
21 using namespace std;
22 const int maxn = 5e5 + 5, maxm = 5e5 + 5;
23 const int INF = 0x3f3f3f3f;
24
25 int head[maxn], cnt;
26
27 struct edges {
28     int to, next;
29     void add(int t, int n) {
30         to = t, next = n;
31     }
32 } edge[maxm << 1]; // 无向图则需要乘2
33
34 inline void add(int u, int v) {
35     edge[++cnt].add(v, head[u]);
36     head[u] = cnt;
37 }
38
39 int fa[maxn][35], dep[maxn], lg[maxn];
40
41 void dfs(int u, int f) {
42     fa[u][0] = f;
43     dep[u] = dep[f] + 1;
44     for (int i = 1; i <= lg[dep[u]]; ++i) fa[u][i] = fa[fa[u][i - 1]][i - 1];
45     for (int i = head[u]; ~i; i = edge[i].next) {
46         int v = edge[i].to;
47         if (v ^ f) dfs(v, u);
48     }
49 }

```

```

48     }
49 }
50 void init(int root, int n) {
51     dep[root] = lg[0] = -1;
52     memset(head, -1, sizeof head); cnt = 0;
53     for (int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1;
54 }
55 int lca(int a, int b) {
56     if (dep[a] < dep[b]) swap(a, b);
57     while (dep[a] > dep[b]) a = fa[a][lg[dep[a] - dep[b]]];
58     if (a == b) return a;
59     for (int i = lg[dep[a]]; ~i; --i) {
60         if (fa[a][i] != fa[b][i]) a = fa[a][i], b = fa[b][i];
61     }
62     return fa[a][0];
63 }
64 int main() {
65
66     int n = IO(), m = IO(), root = IO();
67     init(root, n);
68     for (int i = 1; i < n; ++i) {
69         int u = IO(), v = IO();
70         add(u, v), add(v, u);
71     }
72     dfs(root, 0);
73     while (m--) {
74         int a = IO(), b = IO();
75         printf("%d\n", lca(a, b));
76     }
77     return 0;
78 }

```

## 11. 树链剖分

```

1 // 洛谷板子题
2 #include <cstdio>
3 #include <cstring>
4 #include <algorithm>
5 #include <vector>
6 #include <iostream>
7 #include <cmath>
8 #include <bitset>
9 using namespace std;
10 #define ll long long
11
12 const int N = 1e5 + 5, M = 2e5 + 5;
13 const int maxn = 1e5 + 5, maxm = 2e5 + 5;
14 const int INF = 0x3f3f3f3f;
15
16 int head[maxn], cnt;
17
18 //初始化
19 void init() { memset(head, -1, sizeof head); cnt = -1; }
20
21 struct edges {
22     int to, next;
23     void add(int t, int n) {
24         to = t, next = n;
25     }
26 }edge[maxm << 1]; //无向图则需要乘2
27
28 inline void add(int u, int v) {
29     edge[++cnt].add(v, head[u]);
30     head[u] = cnt;
31 }
32
33 /*****树链剖分*****/
34 int fa[N], dep[N], siz[N], son[N];
35 void dfs1(int u, int f) {
36     fa[u] = f, siz[u] = 1;
37     dep[u] = dep[f] + 1;
38     for (int i = head[u]; ~i; i = edge[i].next) {
39         int v = edge[i].to;
40         if (v == f) continue;
41         dfs1(v, u);
42         siz[u] += siz[v];
43         if (siz[v] > siz[son[u]]) son[u] = v; // 找重儿子
44     }
45 }
46 int v[N]; // 点上的权值
47 int tim, dfn[N], top[N], w[N]; // w的下标是时间戳，对应的是相应时间戳上的点的点权
48 void dfs2(int u, int t) {
49     dfn[u] = ++tim, top[u] = t;
50     w[tim] = v[u];

```

```

51     if (!son[u]) return;
52     dfs2(son[u], t);
53     for (int i = head[u]; ~i; i = edge[i].next) {
54         int v = edge[i].to;
55         if (v == fa[u] || v == son[u]) continue;
56         dfs2(v, v);
57     }
58 }
59 /*****线段树*****/
60 inline int ls(const int& x) { return x << 1; }
61 inline int rs(const int& x) { return x << 1 | 1; }
62 ll seg[N << 2], lazy[N << 2], p;
63 int n, m;
64 inline ll op(const ll& a, const ll& b) {
65     // seg[x] = max(seg[ls(x)], seg[rs(x)]);
66     return (a + b) % p;
67 }
68 inline void push_down(const int& l, const int& r, const int& node) {
69     if (!lazy[node]) return;
70     lazy[ls(node)] += lazy[node], lazy[rs(node)] += lazy[node];
71     lazy[ls(node)] %= p, lazy[rs(node)] %= p;
72     int mid = (l + r) >> 1;
73     seg[ls(node)] = (lazy[node] * (mid - l + 1) + seg[ls(node)]) % p;
74     seg[rs(node)] = (lazy[node] * (r - mid + 1) + seg[rs(node)]) % p;
75     lazy[node] = 0;
76 }
77
78 void build(int l, int r, int node = 1) {
79     if (l == r) {
80         seg[node] = w[l];
81         return;
82     }
83     int mid = (l + r) >> 1;
84     build(l, mid, ls(node)), build(mid + 1, r, rs(node));
85     seg[node] = op(seg[ls(node)], seg[rs(node)]);
86 }
87
88 void update(int ql, int qr, ll x, int l = 1, int r = n, int node = 1) {
89     if (ql <= l && r <= qr) {
90         lazy[node] = (lazy[node] + x) % p;
91         seg[node] = (seg[node] + (r - l + 1) * x) % p;
92         return;
93     }
94     push_down(l, r, node);
95     int mid = (l + r) >> 1;
96     if (ql <= mid) update(ql, qr, x, l, mid, ls(node));
97     if (qr > mid) update(ql, qr, x, mid + 1, r, rs(node));
98     seg[node] = op(seg[ls(node)], seg[rs(node)]);
99 }
100
101 int get(int ql, int qr, int l = 1, int r = n, int node = 1) {
102     if (ql <= l && r <= qr) return seg[node];
103     push_down(l, r, node);
104     int mid = (l + r) >> 1, res = 0;
105     if (ql <= mid) res = get(ql, qr, l, mid, ls(node));
106     if (qr > mid) res = op(res, get(ql, qr, mid + 1, r, rs(node)));
107     return res;
108 }
109 /*****树上操作*****/
110 void update_chain(int x, int y, ll z) {
111     while (top[x] != top[y]) {
112         if (dep[top[x]] < dep[top[y]]) swap(x, y);
113         update(dfn[top[x]], dfn[x], z);
114         x = fa[top[x]];
115     }
116     if (dep[x] > dep[y]) swap(x, y);
117     update(dfn[x], dfn[y], z);
118 }
119
120 ll get_chain(int x, int y) {
121     int res = 0;
122     while (top[x] != top[y]) {
123         if (dep[top[x]] < dep[top[y]]) swap(x, y);
124         res = op(res, get(dfn[top[x]], dfn[x]));
125         x = fa[top[x]];
126     }
127     if (dep[x] > dep[y]) swap(x, y);
128     return op(res, get(dfn[x], dfn[y]));
129 }
130
131 void update_son(int x, ll z) {
132     update(dfn[x], dfn[x] + siz[x] - 1, z);
133 }
134 ll get_son(int x) {
135     return get(dfn[x], dfn[x] + siz[x] - 1);
136 }
137 /*****主函数*****/

```



```

138
139 int main() {
140     std::ios::sync_with_stdio(false);
141     cout.tie(0), cin.tie(0);
142     init();
143     int root;
144     cin >> n >> m >> root >> p;
145     for (int i = 1; i <= n; ++i) cin >> v[i];
146     for (int i = 1; i < n; ++i) {
147         int u, v;
148         cin >> u >> v;
149         add(u, v), add(v, u);
150     }
151     dfs1(root, root), dfs2(root, root);
152     build(1, n);
153     while (m--) {
154         int q, x, y, z;
155         cin >> q >> x;
156         if (q == 1) {
157             cin >> y >> z;
158             update_chain(x, y, z % p);
159         } else if (q == 2) {
160             cin >> y;
161             cout << get_chain(x, y) << endl;
162         } else if (q == 3) {
163             cin >> z;
164             update_son(x, z);
165         } else {
166             cout << get_son(x) << endl;
167         }
168     }
169     return 0;
170 }

```

## 四、图论

### 前置存图

```

1  const int maxn = 1e5, maxm = 2e5;
2  const int INF = 0x3f3f3f3f;
3
4  int head[maxn], cnt;
5
6  //初始化
7  void init() { memset(head, -1, sizeof head); cnt = -1; }
8
9  struct edges {
10     int to, next;
11     int w;
12     void add(int t, int n, int w) {
13         to = t, next = n, this->w = w;
14     }
15 }edge[maxm << 1]; //无向图则需要乘2
16
17 inline void add(int u, int v, int w) {
18     edge[++cnt].add(v, head[u], w);
19     head[u] = cnt;
20 }
21

```

### 1. 最短路

#### 1. dijkstra

```

1  //顶点数和边数
2  const int maxn = 1e5, maxm = 2e5;
3  const int INF = 0x3f3f3f3f;
4
5  int head[maxn], cnt, dis[maxn];
6  bool vis[maxn];
7  //初始化
8  void init() {
9     memset(head, -1, sizeof head);
10     memset(vis, false, sizeof vis);
11     cnt = 0;
12 }
13
14 struct edges {
15     int to, next;
16     int w;
17     edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
18 }edge[maxm << 1]; //无向图则需要乘2

```

```

19
20 inline void add_edges(int u, int v, int w) {
21     edge[++cnt] = edges(v, head[u], w);
22     head[u] = cnt;
23 }
24
25 struct qnode{
26     int v;
27     int w;
28     qnode(int v = 0, int w = 0) : v(v), w(w) {}
29     bool operator< (const qnode &t) const { return w > t.w; }
30 };
31
32 void dijkstra(int n, int s) { // n 为顶点数, m 为边数
33     for (int i = 0; i <= n; ++i) dis[i] = INF;
34     dis[s] = 0;
35     priority_queue<qnode> heap;
36     heap.push(qnode(s, dis[s]));
37     while (heap.size()) {
38         int u = heap.top().v;
39         heap.pop();
40         if (vis[u]) continue;
41         vis[u] = true;
42         for (int i = head[u]; ~i; i = edge[i].next) {
43             int v = edge[i].to;
44             int w = edge[i].w;
45             if (!vis[v] && dis[u] + w < dis[v]) { // 松弛
46                 dis[v] = dis[u] + w;
47                 heap.push(qnode(v, dis[v]));
48             }
49         }
50     }
51 }

```

## 2. bellman-ford

```

1  const int maxn = 1e5, maxm = 2e5;
2  int n, m, s; // n 为顶点数, m 为边数
3  int dis[maxn];
4
5  struct edges {
6      int u, v, w;
7      edges(int u = 0, int v = 0, int w = 0) : u(u), v(v), w(w) {}
8  } edge[maxm];
9
10 bool bf() {
11     for (int k = 1; k < n; k++) {
12         for (int i = 1; i < m; i++) {
13             if (dis[edge[i].v] > dis[edge[i].u] + edge[i].w) { // 松弛
14                 dis[edge[i].v] = dis[edge[i].u] + edge[i].w;
15             }
16         }
17     }
18     for (int i = 1; i < m; i++) {
19         if (dis[edge[i].v] > dis[edge[i].u] + edge[i].w) {
20             return false;
21         }
22     }
23     return true;
24 }
25

```

## 3. spfa

```

1  const int maxn = 1e5, maxm = 2e5;
2  int n, m, s, dis[maxn], num[maxn], head[maxn], cnt; // num 数组时判断是否有负环
3  bool inq[maxn];
4
5  void init() {
6      memset(inq, false, sizeof inq);
7      memset(dis, 0x3f, sizeof dis);
8      memset(num, 0, sizeof num);
9      memset(head, -1, sizeof head);
10     cnt = 0;
11 }
12
13 struct edges {
14     int to, w, next;
15     edges(int to = 0, int w = 0, int next = -1) : to(to), w(w), next(next) {}
16 } edge[maxm];
17
18 inline void add_edges(int u, int v, int w) {
19     edge[++cnt] = edges(v, w, head[u]);

```

```

20     head[u] = cnt;
21 }
22
23 bool spfa() {
24     queue<int> q;
25     q.push(s);
26     inq[s] = true;
27     num[s]++;
28     while (q.size()) {
29         int u = q.front();
30         q.pop();
31         inq[u] = false;
32         for (int i = head[u]; ~i; i = edge[i].next) {
33             int v = edge[i].to, w = edge[i].next;
34             if (dis[v] > dis[u] + w) {
35                 dis[v] = dis[u] + w;
36                 if (!inq[v]) {
37                     q.push(v);
38                     inq[v] = true;
39                     num[v]++;
40                     if (num[v] >= n) return false;
41                     //如果从1号点到x的最短路中包含至少n个点（不包括自己），则存在环
42                 }
43             }
44         }
45     }
46     return true;
47 }

```

#### 4. floyd

```

1  const int M = 2e2;
2  int n, m; //顶点数和边数
3  int dis[M][M];
4
5  void floyd() {
6      for (int k = 0; k < n; k++) {
7          for (int i = 0; i < n; i++) {
8              for (int j = 0; j < n; j++) {
9                  if (dis[i][j] > dis[i][k] + dis[k][j]) {
10                     dis[i][j] = dis[i][k] + dis[k][j];
11                 }
12             }
13         }
14     }
15 }
16
17 void init() {
18     memset(dis, 0x3f, sizeof dis);
19     for (int i = 0; i < M; ++i) dis[i][i] = 0;
20 }

```

## 2. 生成树

### 1. kruskal 适合稀疏图

```

1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #define ll long long
5  using namespace std;
6  #include <cctype>
7  inline long long IO() {
8      long long x = 0;
9      bool f = false;
10     char c = getchar();
11     while (!isdigit(c)) {
12         if (c == '-') f = true;
13         c = getchar();
14     }
15     while (isdigit(c)) {
16         x = (x << 1) + (x << 3) + (c - '0');
17         c = getchar();
18     }
19     return f ? -x : x;
20 }
21 const int M = 2e5 + 10, N = 5e5 + 5;
22 int fa[M];
23 struct edges { int u, v; ll w; } e[N];
24
25 bool cmp(edges& i, edges& j) { return i.w < j.w; }
26 int findset(int x) { return x == fa[x] ? x : fa[x] = findset(fa[x]); }

```

```

27
28 bool un(int a, int b) {
29     int fa1 = findset(a), fa2 = findset(b);
30     if (fa1 == fa2) return false;
31     fa[fa1] = fa2;
32     return true;
33 }
34
35 ll kruskal(int n, int m) {
36     sort(e, e + m, cmp);
37     for (int i = 0; i <= n; ++i) fa[i] = i;
38     int cnt = 0;
39     ll ans = 0;
40     for (int i = 0; i < m; ++i) {
41         if (un(e[i].u, e[i].v)) {
42             ans += e[i].w;
43             if (++cnt == n - 1) break;
44         }
45     }
46     return n - 1 == cnt ? ans : -1;
47 }
48
49 int main() {
50     int n = IO(), m = IO();
51     for (int i = 0; i < m; ++i) e[i].u = IO(), e[i].v = IO(), e[i].w = IO();
52     printf("%lld\n", kruskal(n, m));
53     return 0;
54 }

```

### 3. tarjan

### 4. 网络流

1. Edmonds-Karp算法, 速度较慢

```

1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #include <vector>
5
6  using namespace std;
7  #define ll long long
8  #include <cctype>
9  inline long long IO() {
10     long long x = 0;
11     bool f = false;
12     char c = getchar();
13     while (!isdigit(c)) {
14         if (c == '-') f = true;
15         c = getchar();
16     }
17     while (isdigit(c)) {
18         x = (x << 1) + (x << 3) + (c - '0');
19         c = getchar();
20     }
21     return f ? -x : x;
22 }
23
24 const int maxn = 1e5, maxm = 2e5;
25 const int INF = 0x3f3f3f3f;
26 const ll inf = 0xfffffffffff;
27
28 int head[maxn], cnt;
29
30 //初始化
31 void init() { memset(head, -1, sizeof head); cnt = -1; }
32
33 struct edges {
34     int to, next;
35     ll c;
36     edges(int to = 0, int next = -1, int c = 0) : to(to), next(next), c(c) {}
37 } edge[maxm << 1]; //无向图则需要乘2
38
39 inline void add(int u, int v, ll c, bool f = 1) {
40     edge[++cnt] = edges(v, head[u], c);
41     head[u] = cnt;
42     if (f) add(v, u, 0, 0); // 建立反向弧
43 }
44 #include <queue>
45 bool vis[maxn]; // 记录是否在队内
46 ll minc[maxn]; // 记录增广路的最小流
47 struct pairs { int u, i; } pre[maxn];
48 bool bfs(int s, int t, int n) {
49     queue<int> q;

```

```

50     for (int i = 0; i <= n; ++i) vis[i] = false;
51     q.push(s), vis[s] = true, minc[s] = inf;
52     while (q.size()) {
53         int u = q.front(); q.pop();
54         for (int i = head[u]; ~i; i = edge[i].next) {
55             int v = edge[i].to;
56             if (vis[v] || !edge[i].c) continue;
57             vis[v] = true, pre[v] = {.u = u, .i = i}; //记录当前点的前驱点和当前点的内存池编号
58             minc[v] = min(minc[u], edge[i].c);
59             if (v == t) return true;
60             q.push(v);
61         }
62     }
63     return false;
64 }
65
66
67 ll EK(int s, int t, int n) {
68     ll ans = 0, &dif = minc[t];
69     while (bfs(s, t, n)) {
70         ans += dif;
71         for (int i = t; i != s; i = pre[i].u) {
72             edge[pre[i].i].c -= dif;
73             edge[pre[i].i ^ 1].c += dif;
74         }
75     }
76     return ans;
77 }
78
79 int main() {
80     int n = IO(), m = IO(), s = IO(), t = IO();
81     init();
82     for (int i = 0; i < m; ++i) {
83         int u = IO(), v = IO();
84         ll c = IO();
85         add(u, v, c);
86     }
87     printf("%lld", EK(s, t, n));
88     return 0;
89 }
90

```

2. dinic, 当前弧优化+多路增广优化+炸点优化(模板题),复杂度 $O(n^2m)$

```

1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #include <vector>
5
6  using namespace std;
7  #include <cctype>
8  inline long long IO() {
9      long long x = 0;
10     bool f = false;
11     char c = getchar();
12     while (!isdigit(c)) {
13         if (c == '-') f = true;
14         c = getchar();
15     }
16     while (isdigit(c)) {
17         x = (x << 1) + (c << 3) + (c - '0');
18         c = getchar();
19     }
20     return f ? -x : x;
21 }
22 #include <iostream>
23 #include <string>
24 #include <queue>
25 #include <cmath>
26 #define ll long long
27 const int N = 1e5 + 5, M = 1e6 + 5;
28 const ll inf = 0xffffffff;
29
30 const int maxn = 1500, maxm = 1e5 + 2e4 + 5;
31
32 int head[maxn], cnt;
33
34 //初始化
35 inline void init() {
36     memset(head, -1, sizeof head);
37     cnt = -1;
38 }
39
40 struct edges {
41     int to, next;

```

```

42     ll c; //容量
43     edges(int to = 0, int next = -1, ll c = 0) : to(to), next(next), c(c) {}
44 }edge[maxm << 1]; //无向图则需要乘2
45
46 inline void add(int u, int v, ll c, bool f = 1) {
47     edge[++cnt] = edges(v, head[u], c), head[u] = cnt;
48     if (f) add(v, u, 0, 0); // 添加反向弧
49 }
50 #include <queue>
51 int deep[maxn], cur[maxn];
52 // bfs求增广路，一次求出多条增广路
53 bool bfs(int s, int t, int n) {
54     queue<int> q;
55     for (int i = 1; i <= n; ++i) deep[i] = 0;
56     deep[s] = 1;
57     q.push(s);
58     while (q.size()) {
59         int u = q.front();
60         q.pop();
61         for (int i = head[u]; ~i; i = edge[i].next) {
62             const int &v = edge[i].to;
63             const ll &c = edge[i].c;
64             if (deep[v] || !c) continue;
65             deep[v] = deep[u] + 1;
66             q.push(v);
67         }
68     }
69     return deep[t] != 0;
70 }
71
72 ll dfs(int u, int t, ll f) {
73     if (u == t) return f;
74     ll nowflow = 0;
75     for (int i = cur[u]; ~i; i = edge[i].next) {
76         cur[u] = i; // 当前弧优化
77         int &v = edge[i].to;
78         ll &c = edge[i].c;
79         if (deep[v] != deep[u] + 1 || !c) continue;
80         if (ll low = dfs(v, t, min(f - nowflow, c))) {
81             c -= low, edge[i ^ 1].c += low;
82             nowflow += low; // 多路增广优化
83             if (nowflow == f) break;
84         }
85     }
86     if (!nowflow) deep[u] = -2; // 炸点优化
87     return nowflow;
88 }
89
90 ll dinic(int s, int t, int n) {
91     ll ans = 0;
92     while (bfs(s, t, n)) {
93         for (int i = 1; i <= n; ++i) cur[i] = head[i]; // 预处理，方便当前弧优化
94         ans += dfs(s, t, inf); // 进过多路增广优化可不用循环
95     }
96     return ans;
97 }
98
99
100 int main() {
101     int n = IO(), m = IO(), s = IO(), t = IO();
102     init();
103     for (int i = 0; i < m; ++i) {
104         int u = IO(), v = IO();
105         ll c = IO();
106         add(u, v, c);
107     }
108     printf("%lld", dinic(s, t, n));
109     return 0;
110 }

```

3. 最小费用最大流，将ek算法中的bfs换成spfa

```

1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  #include <vector>
5
6  using namespace std;
7  #define ll long long
8  #include <cctype>
9  inline long long IO() {
10     long long x = 0;
11     bool f = false;
12     char c = getchar();
13     while (!isdigit(c)) {

```

```

14     if (c == '-') f = true;
15     c = getchar();
16 }
17 while (isdigit(c)) {
18     x = (x << 1) + (x << 3) + (c - '0');
19     c = getchar();
20 }
21 return f ? -x : x;
22 }
23
24 const int maxn = 1e5, maxm = 2e5;
25 const int INF = 0x3f3f3f3f;
26 const ll inf = 0xfffffffffff;
27
28 int head[maxn], cnt;
29
30 //初始化
31 void init() { memset(head, -1, sizeof head); cnt = -1; }
32
33 struct edges {
34     int to, next;
35     ll c, w;
36     edges(int to = 0, int next = -1, ll c = 0, ll w = 0) :
37         to(to), next(next), c(c), w(w) {}
38 } edge[maxm << 1]; //无向图则需要乘2
39
40 inline void add(int u, int v, ll c, ll w, bool f = 1) {
41     edge[++cnt] = edges(v, head[u], c, w);
42     head[u] = cnt;
43     if (f) add(v, u, 0, -w, 0); // 建立反向弧, 费用相反
44 }
45
46 #include <queue>
47 bool inq[maxn];
48 ll dist[maxn];
49 struct pairs { int u, i; } pre[maxn];
50 // 利用spfa找最小费用的路, 即最短路
51 bool spfa(int s, int t, int n) {
52     for (int i = 0; i <= n; ++i) inq[i] = false, dist[i] = inf;
53     queue<int> q;
54     dist[s] = 0, inq[s] = true, q.push(s), pre[t].i = -1;
55     while (q.size()) {
56         int u = q.front(); q.pop();
57         inq[u] = false;
58         for (int i = head[u]; ~i; i = edge[i].next) {
59             int v = edge[i].to;
60             if (dist[v] > dist[u] + edge[i].w && edge[i].c) {
61                 pre[v] = {u, i}; //记录当前点的前驱点和当前点的内存池编号
62                 dist[v] = dist[u] + edge[i].w;
63                 if (inq[v]) continue;
64                 inq[v] = true, q.push(v);
65             }
66         }
67     }
68     return pre[t].i != -1; // 如果说t没有前驱则说明找不到增广路了
69 }
70
71 void mcmf(int s, int t, int n, ll &maxf, ll &minv) {
72     maxf = minv = 0;
73     while (spfa(s, t, n)) {
74         ll low = inf;
75         for (int i = t; i != s; i = pre[i].u) low = min(low, edge[pre[i].i].c); //寻找最小流
76         for (int i = t; i != s; i = pre[i].u) {
77             edge[pre[i].i].c -= low;
78             edge[pre[i].i ^ 1].c += low;
79             minv += low * edge[pre[i].i].w;
80         }
81         maxf += low;
82     }
83 }
84
85 int main() {
86     int n = IO(), m = IO(), s = IO(), t = IO();
87     init();
88     for (int i = 0; i < m; ++i) {
89         int u = IO(), v = IO();
90         ll c = IO(), w = IO();
91         add(u, v, c, w);
92     }
93     ll maxflow, mincost;
94     mcmf(s, t, n, maxflow, mincost);
95     printf("%lld %lld", maxflow, mincost);
96     return 0;
97 }

```

## 5. 二分图

1. 匈牙利算法, 时间复杂度 $O(ev)$

```
1  int match[M];
2  bool vis[M];
3  bool dfs(int u) {
4      for (int &v : gp[u]) {
5          if (vis[v]) continue;
6          vis[v] = true;
7          if (!match[v] || dfs(match[v])) {
8              match[u] = v, match[v] = u;
9              return true;
10         }
11     }
12     return false;
13 }
14 // 主函数里
15 fill_n(match, n + 1, 0); // n是点的个数
16 for (int i = 1; i <= n; ++i) {
17     if (match[i]) continue;
18     fill_n(vis, n + 1, false);
19     dfs(i);
20 }
```

- 1.

## 五、字符串

### 1. KMP

```
1  const int M = 1e6 + 5;
2  //普通版本
3  void getNext(char *x, int len, int *nxt) {
4      int i = 0, j;
5      j = nxt[0] = -1;
6      while (i < len) {
7          while (j != -1 && x[i] != x[j]) j = nxt[j];
8          nxt[++i] = ++j;
9      }
10 }
11 //略微优化版本
12 void getNext(char *x, int len, int *nxt) {
13     int i = 0, j;
14     j = nxt[0] = -1;
15     while (i < len) {
16         while (j != -1 && x[i] != x[j]) j = nxt[j];
17         if (x[++i] == x[++j]) nxt[i] = nxt[j];
18         else nxt[i] = j;
19     }
20 }
21 // y是主串
22 int nxt[M];
23 int kmpCount(char *y, int n, char *x, int m) {
24     int i = 0, j = 0, ans = 0;
25     getNext(x, m, nxt);
26     while (i < n) {
27         while (j != -1 && y[i] != x[j]) j = nxt[j];
28         ++i, ++j;
29         if (j >= m) ++ans, j = nxt[j];
30     }
31     return ans;
32 }
```

### 2. 字符串Hash

```
1  unsigned int DJBHash(const char *str) {
2      unsigned int hash = 5381;
3      while (*str) hash += (hash << 5) + (*str++);
4      return (hash & 0xffffffff); //7个f
5  }
6
7  unsigned int BKDRHash(const char *str) {
8      unsigned int seed = 131; // 31 131 1313 13131 131313...
9      unsigned int hash = 0;
10     while (*str) hash = hash * seed + (*str++);
11     return (hash & 0xffffffff);
12 }
```



```

13 #define ull unsigned long long
14 ull strhash(const char *s) {
15     ull seed = 1313, res = 0; // 31 131 1313 13131
16     while (*s) res = res * seed + (*s++);
17     return res;
18 }
19

```

### 3. 马拉车

### 4. exkmp

```

1 void pre_exkmp(char x[], int m, int next[]) {
2     next[0] = m;
3     int j = 0;
4     while (j + 1 < m && x[j] == x[j + 1]) j++;
5     next[1] = j;
6     int k = 1;
7     for (int i = 2; i < m; i++) {
8         int p = next[k] + k - 1;
9         int L = next[i - k];
10        if (i + L < p + 1) next[i] = L;
11        else {
12            j = max(0, p - i + 1);
13            while (i + j < m && x[i + j] == x[j]) j++;
14            next[i] = j;
15            k = i;
16        }
17    }
18 }
19 void exkmp(char x[], int m, char y[], int n, int next[], int extend[]) {
20     pre_exkmp(x, m, next);
21     int j = 0;
22     while (j < n && j < m && x[j] == y[j]) j++;
23     extend[0] = j;
24     int k = 0;
25     for (int i = 1; i < n; i++) {
26         int p = extend[k] + k - 1;
27         int L = next[i - k];
28         if (i + L < p + 1) extend[i] = L;
29         else {
30             j = max(0, p - i + 1);
31             while (i + j < n && j < m && y[i + j] == x[j]) j++;
32             extend[i] = j;
33             k = i;
34         }
35     }
36 }

```

## 六、计算几何

- 未完善

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4 #include <cmath>
5 #include <vector>
6
7 /**
8  * 本板子属于半成品，有些功能并没有验证
9  * 函数说明：
10  * 关于点的函数
11  * 点的Point(double, double) 构造函数
12  * + 向量加法
13  * - 向量减法
14  * == 判断两个点是否相等
15  * *(Point) 向量点乘
16  * *(double) 向量伸长(没有除法，要用除法直接乘倒数)
17  * ^ 向量叉乘
18  * < 点对点的比较
19  * double len() 向量的长度，也可以用来求两个点的距离
20  * Point rotate(double angle) 向量逆时针旋转angle弧度
21  * Point rotate(Point, double) 点让点p逆时针旋转angle弧度
22  * void print() 将点输出
23  * int init() 输入点的坐标 返回值和scanf相同
24  * 其他非结构体函数
25  * angle(Point&, Point&) 计算两个向量的夹角
26  *
27  * 关于线的函数
28  * Line(Point, Point) 构造函数
29  * Line(Point, double) 根据一个点和一个倾斜角 0 <= angle < PI确定直线 (未验证)

```

```

30 * double len() 返回线段的长度
31 * double point(double t) 返回距离点p向前t倍向量的点
32 * double angle() 返回直线的倾斜角 范围[0, PI] (未验证)
33 * double disPointToLine(const Point&) 点到这条直线的距离
34 * double disPointToSeg(const Point&) 点到这条线段的距离 (未验证)
35 * Point getPro(const Point&) 点在这条线上的投影 (未验证)
36 * Point getSym(const Point&) 点关于这条线的对称点 (未验证)
37 * bool isOnLine(const Point&) 验证该点是否在这条直线上 (未验证)
38 * bool isOnSeg(const Point&) 验证该点是否在这条线段上 (未验证)
39 * Point cross(Line&) 直线和这条直线的交点, 前提是相交
40 * void print() 输出这条线段
41 *
42 * 其他非结构体
43 * int LineAndLine(Line&, Line&) 直线和直线的关系 0平行 1重合 2相交 (未验证)
44 * Point getLineInter(const Line&, const Line&) 求两直线的交点, 必须相交才能调用
45 * int SegAndSeg(const Line& l1, const Line& l2) 两个线段的关系 0不相交 1非规范相交 (其中一个线段的端点和另一个线段相交)
2 规范相交 (未验证)
46 * int LineAndSeg(const Line& line, const Line& seg) 直线和线段的关系, 0不相交 1非规范相交 2规范相交 (未验证)
47 *
48 * 关于圆的函数
49 * Circle(Point, double) 构造函数
50 * Circle(Point, Point, Point) 过三点的圆
51 * double area() const; 返回圆的面积
52 * double circum() const 返回圆的周长
53 * int PointAndCircle(Point&) 点和圆的关系 返回 0圆外 1圆上 2圆内
54 * int LineAndCircle(Point&) 点和圆的关系 返回 0不相交 1相交 2相交两个点 (未验证)
55 * int CircleAndCircle(Circle&) 圆和圆的关系 返回 0内含 1内切 2相交两点 3外切 4外离 (未验证)
56 *
57 * 关于三角形的函数
58 * Triangle(Point, Point, Point) 构造函数
59 * double area() const 返回三角形函数
60 * Circle outerCircle() 获取三角形的外接圆
61 *
62 * 关于多边形的函数
63 * Polygon(vector<Point>&) 构造函数
64 * double circum(); 求凸包的周长
65 * void graham(Polygon&) 求凸包 传入值为需要求出的凸包的点集
66 * int PointAndPolgon(Point&) 判断点与多边形的关系, 0外 1内 2边上 3点上 (未实现)
67 * double minRectCover() 点集的最小矩形覆盖, 自己必须是 (逆时针) 凸包才能调用 (未实现)
68 * Circle minCircleCover() 点集的最小圆覆盖
69 */
70
71 using namespace std;
72 const double eps = 1e-8, PI = acos(-1.0);
73 int dcmp(double x) {
74     if (fabs(x) < eps) return 0;
75     return x > 0 ? 1 : -1;
76 }
77 /*****点*****/
78 /*
79 除了结构体内部函数还有
80 angle(Point& a, Point& b) // 两个向量的夹角
81 */
82 struct Point {
83     double x, y;
84     Point(double x = 0, double y = 0) : x(x), y(y) {}
85     Point operator + (const Point&) const;
86     Point operator - (const Point&) const;
87     double operator * (const Point&) const; // 点乘
88     double operator ^ (const Point&) const; // 叉乘
89     bool operator == (const Point&) const;
90     bool operator < (const Point&) const; // 排序需要
91     Point operator * (double); // 向量伸长b倍
92     double len() const; // 向量的长度
93     Point rotate(double); // 向量逆时针旋转a弧度后
94     Point rotate(Point&, double); // 点绕p点顺时针旋转a弧度后
95     void print() { printf("%.2f %.2f", x, y); }
96     int init() { return scanf("%lf%lf", &x, &y); }
97 };
98 Point Point::operator + (const Point& b) const {
99     return Point(x + b.x, y + b.y);
100 }
101 Point Point::operator - (const Point& b) const {
102     return Point(x - b.x, y - b.y);
103 }
104 // 点乘
105 double Point::operator * (const Point& b) const {
106     return x * b.x + y * b.y;
107 }
108 // 叉乘
109 double Point::operator ^ (const Point& b) const {
110     return x * b.y - y * b.x;
111 }
112 bool Point::operator == (const Point& b) const {
113     return !dcmp(x - b.x) && !dcmp(y - b.y);
114 }
115 bool Point::operator < (const Point& b) const {

```

```

116     return (!dcmp(x - b.x)) ? dcmp(y - b.y) < 0 : x < b.x;
117 }
118 // 向量的长度
119 double Point::len() const { return sqrt(x * x + y * y); }
120 // 向量伸长b倍
121 Point Point::operator * (double b) {
122     return Point(x * b, y * b);
123 }
124 // 向量逆时针旋转a弧度后
125 // cosx -sinx
126 // sinx cosx
127 Point Point::rotate(double a) {
128     return Point(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
129 }
130 // 点绕p点顺时针旋转a弧度后
131 Point Point::rotate(Point &p, double a) {
132     Point vec = (*this) - p;
133     return vec.rotate(a) + p;
134 }
135 // 两个向量的夹角
136 double angle(Point& a, Point& b) {
137     return acos(a * b / a.len() / b.len());
138 }
139 /*****线*****/
140 /*
141 除了结构体内部函数还有
142 int LineAndLine(Line& l1, Line&) 直线和直线的关系 0平行 1重合 2相交
143 Point getLineInter(const Line& l1, const Line& l2) 求两直线的交点,必须相交才能调用
144 int SegAndSeg(const Line& l1, const Line& l2) 两个线段的关系 0不相交 1非规范相交(其中一个线段的端点和另一个线段相交) 2规范相交
145 int LineAndSeg(const Line& line, const Line& seg) 直线和线段的关系, 0不相交 1非规范相交 2规范相交
146 */
147 struct Line {
148     Point p1, p2, v;
149     Line() {}
150     Line(Point p1, Point p2) : p1(p1), p2(p2), v(p2 - p1) {}
151     Line(Point, double); // 根据一个点和一个倾斜角0<= angle < PI确定直线
152     double len(); // 线段的长度
153     Point point(double); // P = p1 + vt
154     double angle(); // 直线的倾斜角[0, PI)
155     double disPointToLine(const Point&); // 点到这条直线的距离
156     double disPointToSeg(const Point&); // 点到这条线段的距离
157     Point getPro(const Point&); // 点到这条线的投影
158     Point getSym(const Point&); // 点关于这条线的对称点
159     bool isOnLine(const Point&); // 点是否在这条直线上
160     bool isOnSeg(const Point&); // 点是否在这条线段上
161     Point cross(Line&); // 直线和这条直线的交点,前提是相交才能调用
162     void print(); // 输出线段
163 };
164 Line::Line(Point p, double angle) : p1(p) {
165     if (!dcmp(angle - PI / 2)) p2 = p1 + Point(0, 1);
166     else p2 = p1 + Point(1, tan(angle));
167 }
168 double Line::len() { return v.len(); }
169 Point Line::point(const double t) { return (p1 + (v * t)); }
170 double Line::angle() {
171     double ret = atan2(p2.y - p1.y, p2.x - p1.x);
172     if (dcmp(ret) < 0) ret += PI;
173     if (!dcmp(ret - PI)) ret -= PI;
174     return ret;
175 }
176 void Line::print() {
177     printf("(f,f)->(f,f)", p1.x, p1.y, p2.x, p2.y);
178 }
179 double Line::disPointToLine(const Point& p) {
180     Point vec = p - p1;
181     return fabs(v ^ vec) / v.len();
182 }
183 double Line::disPointToSeg(const Point& p) {
184     if (p1 == p2) return (p1 - p).len();
185     Point v1 = p - p1, v2 = p - p2;
186     if (dcmp(v1 * v) < 0) return v1.len();
187     if (dcmp(v2 * v) < 0) return v2.len();
188     return disPointToLine(p);
189 }
190 Point Line::getPro(const Point& p) {
191     return p1 + v * (v * (p - p1) / v.len());
192 }
193 Point Line::getSym(const Point& p) {
194     Point q = getPro(p);
195     return Point(2 * q.x - p.x, 2 * q.y - p.y);
196 }
197 bool Line::isOnLine(const Point& p) {
198     return !dcmp((p - p1) ^ (p - p2));
199 }
200 bool Line::isOnSeg(const Point& p) {
201     return isOnLine(p) && (dcmp((p - p1) * (p - p2)) <= 0);

```

```

202 }
203 Point Line::cross(Line& l) {
204     double a1 = l.v ^ (p1 - l.p1);
205     double a2 = l.v ^ (p2 - l.p1);
206     return Point((p1.x * a2 - p2.x * a1) / (a2 - a1),
207         (p1.y * a2 - p2.y * a1) / (a2 - a1));
208 }
209 // 直线和直线的关系 0平行 1重合 2相交
210 int LineAndLine(Line& l1, Line& l2) {
211     if (!dcmp(l1.v ^ l2.v)) return l2.isOnLine(l1.p1);
212     return 2;
213 }
214 // 求两个直线的交点, 必须相交才能调用
215 Point getLineInter(Line& l1, Line& l2) {
216     Point vec = l1.p1 - l2.p1;
217     double t = (l2.v ^ vec) / (l1.v ^ l2.v);
218     return l1.point(t);
219 }
220 // 判断两个线段的关系
221 int SegAndSeg(const Line& l1, const Line& l2) {
222     int d1 = dcmp(l1.v ^ (l2.p1 - l1.p1));
223     int d2 = dcmp(l1.v ^ (l2.p2 - l1.p1));
224     int d3 = dcmp(l2.v ^ (l1.p1 - l2.p1));
225     int d4 = dcmp(l2.v ^ (l1.p2 - l2.p1));
226     if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
227     return (!d1 && dcmp((l2.p1 - l1.p1) * (l2.p1 - l1.p2)) <= 0) ||
228         (d2 && dcmp((l2.p2 - l1.p1) * (l2.p2 - l1.p2)) <= 0) ||
229         (d3 && dcmp((l1.p1 - l2.p1) * (l1.p1 - l2.p2)) <= 0) ||
230         (d4 && dcmp((l1.p2 - l2.p1) * (l1.p2 - l2.p2)) <= 0);
231 }
232 // 直线和线段的关系, 0不相交 1非规范相交 2规范相交
233 int LineAndSeg(const Line& line, const Line& seg) {
234     int d1 = dcmp(line.v ^ (seg.p1 - line.p1));
235     int d2 = dcmp(line.v ^ (seg.p2 - line.p1));
236     if ((d1 ^ d2) == -2) return 2;
237     return d1 == 0 || d2 == 0;
238 }
239 /*****圆*****/
240 struct Circle {
241     Point p;
242     double r;
243     Circle() {}
244     Circle(Point p, double r) : p(p), r(r) {}
245     Circle(Point, Point, Point); // 过三点一个圆
246     double area() const; // 面积
247     double circum() const; // 周长
248     int PointAndCircle(Point&); // 点和圆的关系 0圆外 1圆上 2圆内
249     int LineAndCircle(Line&); // 直线和圆的关系 0不相交 1相切 2相交两点
250     int CircleAndCircle(Circle&); // 圆和圆的关系 0内含 1内切 2相交两点 3外切 4外离
251 };
252 double Circle::area() const { return PI * r * r; }
253 double Circle::circum() const { return 2 * PI * r; }
254 Circle::Circle(Point a, Point b, Point c) {
255     Point v1 = b - a, v2 = c - a;
256     Line l1((a + b) * 0.5, ((a + b) * 0.5) + Point(-v1.y, v1.x));
257     Line l2((b + c) * 0.5, ((b + c) * 0.5) + Point(-v2.y, v2.x));
258     p = getLineInter(l1, l2);
259     r = (p - a).len();
260 }
261 int Circle::PointAndCircle(Point& a) {
262     int d = dcmp((a - p).len() - r);
263     if (d > 0) return 0;
264     if (d < 0) return 2;
265     return 1;
266 }
267 int Circle::LineAndCircle(Line& l) {
268     int d = dcmp(l.disPointToLine(p) - r);
269     if (d > 0) return 0;
270     if (d < 0) return 2;
271     return 1;
272 }
273 int Circle::CircleAndCircle(Circle& c) {
274     double dist = (c.p - p).len();
275     if (dcmp(dist - r - c.r) > 0) return 4;
276     if (!dcmp(dist - r - c.r)) return 3;
277     double l = fabs(r - c.r);
278     if (dcmp(dist - r - c.r) < 0 && dcmp(dist - l) > 0) return 2;
279     return dcmp(dist - l) == 0;
280 }
281 /*****三角形*****/
282 struct Triangle {
283     Point p[3];
284     Triangle() {}
285     Triangle(Point A, Point B, Point C);
286     double area() const; // 三角形面积
287     Circle outerCircle(); // 外接圆
288 };

```

```

289 Triangle::Triangle(Point A, Point B, Point C) {
290     p[0] = A, p[1] = B, p[2] = C;
291 }
292 double Triangle::area() const {
293     double ret = 0;
294     for (int i = 0; i < 3; ++i) {
295         ret += p[i] ^ p[(i + 1) % 3];
296     }
297     return fabs(ret) / 2;
298 }
299 // r = a * b * c / 4S
300 // 两条边的中垂线的交点, 也可用kuangbin的方法, 但此方法不需要再次调用Line的函数
301 Circle Triangle::outerCircle() {
302     double A1 = 2.0 * (p[1].x - p[0].x), B1 = 2.0 * (p[1].y - p[0].y);
303     double A2 = 2.0 * (p[2].x - p[1].x), B2 = 2.0 * (p[2].y - p[1].y);
304     double C1 = 0, C2 = 0;
305     for (int i = 0, j = 1, k = -1; i < 2; ++i, ++j, k = 1) {
306         C1 += (p[i].x * p[i].x + p[i].y * p[i].y) * k;
307         C2 += (p[j].x * p[j].x + p[j].y * p[j].y) * k;
308     }
309     double x = ((C1 * B2) - C2 * B1) / ((A1 * B2) - A2 * B1);
310     double y = ((A1 * C2) - A2 * C1) / ((A1 * B2) - A2 * B1);
311     return Circle(Point(x, y), (Point(x, y) - p[0]).len());
312 }
313
314 /*****多边形/点集*****/
315
316 struct Polygon {
317     vector<Point> p;
318     // vector<Line> l;
319     Polygon(){}
320     Polygon(vector<Point>& p) : p(p) {};
321     double circum(); // 求凸包的周长
322     void graham(Polygon&); // 凸包
323     int PointAndPolygon(Point&); // 判断点与多边形的关系, 0外 1内 2边上 3点上
324     double minRectCover(); // 最小矩形覆盖, 自己必须是(逆时针)凸包才能调用
325     Circle minCircleCover(); // 点集的最小圆覆盖
326 };
327 double Polygon::circum() {
328     int n = p.size();
329     if (n < 2) return 0.0;
330     if (n == 2) return (p[0] - p[1]).len();
331     double ret = 0;
332     for (int i = 0; i < n; ++i) {
333         ret += (p[(i + 1) % n] - p[i]).len();
334     }
335     return ret;
336 }
337 void Polygon::graham(Polygon& res) {
338     int indx = 0, n = p.size();
339     for (int i = 1; i < n; ++i) if (p[i] < p[indx]) indx = i;
340     swap(p[0], p[indx]);
341     sort(p.begin() + 1, p.end(), [&](Point& i, Point& j) {
342         int d = dcmp(atan2(i.y - p[0].y, i.x - p[0].x) - atan2(j.y - p[0].y, j.x - p[0].x));
343         if (d) return d < 0;
344         return i.x < j.x;
345     });
346     res.p.emplace_back(p[0]);
347     if (n == 1) return;
348     res.p.emplace_back(p[1]);
349     if (n == 2) {
350         if (p[0] == p[1]) res.p.pop_back();
351         return;
352     }
353     int x = res.p.size();
354     for (int i = 2; i < n; ++i, ++x) {
355         while (x >= 2 && dcmp((res.p[x - 2] - res.p[x - 1]) ^ (res.p[x - 2] - p[i])) <= 0) {
356             --x;
357             res.p.pop_back();
358         }
359         res.p.emplace_back(p[i]);
360     }
361     if (res.p.size() == 2 && (res.p[0] == res.p[1])) res.p.pop_back();
362 }
363 Circle Polygon::minCircleCover() {
364     random_shuffle(p.begin(), p.end());
365     Circle res(p[0], 0);
366     int n = p.size();
367     for (int i = 1; i < n; ++i) {
368         if (res.PointAndCircle(p[i])) continue; // 在圆内
369         res = Circle(p[i], 0);
370         for (int j = 0; j < i; ++j) {
371             if (res.PointAndCircle(p[j])) continue; // 在圆内
372             res = Circle((p[i] + p[j]) * 0.5, (p[i] - p[j]).len() * 0.5);
373             for (int k = 0; k < j; ++k) {
374                 if (res.PointAndCircle(p[k])) continue; // 在圆内
375                 res = Circle(p[i], p[j], p[k]);

```

```

376     }
377     }
378     }
379     return res;
380 }
381 vector<Point> vt;
382 int main() {
383     int t;
384     while(scanf("%d", &t), t) {
385         Point p;
386         for (int i = 0; i < t; ++i) {
387             p.init();
388             vt.emplace_back(p);
389         }
390         Polygon pol(vt), ans;
391         pol.graham(ans);
392         printf("%.2f\n", ans.circum());
393         vt.clear();
394     }
395     return 0;
396 }

```

## 七、动态规划

### 1. 树形dp

#### 1. 树的最大独立集

```

1  /*
2  Loj 10160
3  每个点都有一个快乐值，子结点和父节点不能同时被选，问你最大的快乐值
4  dp[i][0]表示第i号结点不选时最大的快乐值
5  dp[i][1]表示第i号结点选时的最大的快乐值
6  */
7  #include <iostream>
8  #include <algorithm>
9  #include <cstdio>
10 #include <cctype>
11 #include <cstring>
12 using namespace std;
13 const int M = 6e3 + 5;
14 inline int read() {
15     int x = 0;
16     bool f = false;
17     char c = getchar();
18     while (!isdigit(c)) {
19         if (c == '-') f = true;
20         c = getchar();
21     }
22     while (isdigit(c)) {
23         x = (x << 1) + (x << 3) + (c - 48);
24         c = getchar();
25     }
26     return f ? -x : x;
27 }
28
29 struct es{
30     int to, nxt;
31 }e[M << 1];
32 int head[M], cnt;
33 inline void init() { memset(head, -1, sizeof head); cnt = 0; }
34 inline void add(int u, int v) {
35     e[++cnt] = {.to = v, .nxt = head[u]};
36     head[u] = cnt;
37 }
38
39 int n, h[M], vis[M], dp[M][2];
40
41 void dfs(int u) {
42     dp[u][1] = h[u];
43     for (int i = head[u]; ~i; i = e[i].nxt) {
44         int v = e[i].to;
45         dfs(v);
46         dp[u][0] += max(dp[v][0], dp[v][1]);
47         dp[u][1] += dp[v][0];
48     }
49 }
50
51
52 int main() {
53     int n = read();
54     for (int i = 1; i <= n; ++i) h[i] = read();

```

```

55     int u = read(), v = read();
56     init();
57     while (u | v) {
58         add(v, u);
59         vis[u] = true;
60         u = read(), v = read();
61     }
62     int root = 0;
63     for (int i = 1; i <= n; ++i) {
64         if (!vis[i]) root = i;
65     }
66     dfs(root);
67     printf("%d\n", max(dp[root][0], dp[root][1]));
68     return 0;
69 }

```

## 2. 树的最小支配集

```

1  /*
2  Loj 10157
3  每个点都有点权，一个点可以看守连着他的边上的点，选一些点出来，使他们能够看守整颗树上所有的点
4  问你最小选出来的权值
5  dp[0][i]表示i点被选中，则其 += min({dp[0][son], dp[2][son], dp[1][son]})
6  dp[1][i]表示i点没被选中，但是其父亲被选中了，则其 += min(dp[0][son], dp[2][son])
7  dp[2][i]表示i点没被选中，但是去其中某几个儿子被选中了，注意这个比较难转移，转移方式如下
8  先求出所有儿子min(dp[0][son], dp[1][son])的总和，然后在递归完后选出最小是那个儿子的dp[0][son]
9  即dp[2][u] = min(dp[2][u], sum - min(dp[2][v], dp[0][v]) + dp[0][v]);这行
10 */
11 #include <cstdio>
12 #include <cstring>
13 #include <algorithm>
14 #include <vector>
15
16 using namespace std;
17 #define ll long long
18 #include <cctype>
19 inline long long IO() {
20     long long x = 0;
21     bool f = false;
22     char c = getchar();
23     while (!isdigit(c)) {
24         if (c == '-') f = true;
25         c = getchar();
26     }
27     while (isdigit(c)) {
28         x = (x << 1) + (x << 3) + (c - '0');
29         c = getchar();
30     }
31     return f ? -x : x;
32 }
33
34 const int maxn = 1e5, maxm = 1e5;
35 const int INF = 0x3f3f3f3f;
36
37 int head[maxn], cnt, dis[maxn];
38
39 //初始化
40 void init() {
41     memset(head, -1, sizeof head);
42     // memset(vis, false, sizeof vis);
43     cnt = 0;
44 }
45
46 struct edges {
47     int to, next;
48     int w;
49     edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
50 } edge[maxm << 1]; //无向图则需要乘2
51
52 inline void add(int u, int v, int w = 0) {
53     if (cnt == 0) init();
54     edge[++cnt] = edges(v, head[u], w);
55     head[u] = cnt;
56 }
57
58 const int M = 1e4, inf = 0x3f3f3f3f;
59 int n, m, dp[3][M], vis[M], c[M];
60
61 void dfs(int u, int fa) {
62     dp[0][u] = c[u], dp[2][u] = inf;
63     int sum = 0;
64     for (int i = head[u]; ~i; i = edge[i].next) {
65         int v = edge[i].to;
66         if (v == fa) continue;
67         dfs(v, u);
68         dp[0][u] += min({dp[0][v], dp[2][v], dp[1][v]});

```

```

68     if (fa != -1) dp[1][u] += min(dp[2][v], dp[0][v]);
69     sum += min(dp[2][v], dp[0][v]);
70 }
71 for (int i = head[u]; ~i; i = edge[i].next) {
72     int v = edge[i].to;
73     if (v == fa) continue;
74     dp[2][u] = min(dp[2][u], sum - min(dp[2][v], dp[0][v]) + dp[0][v]);
75 }
76 }
77
78
79 int main() {
80     n = IO();
81     init();
82     for (int i = 0; i < n; ++i) {
83         int u = IO(), w = IO(), k = IO();
84         c[u] = w;
85         while (k--) {
86             int v = IO();
87             add(u, v), add(v, u);
88         }
89     }
90     dfs(1, -1);
91     printf("%d", min(dp[0][1], dp[2][1]));
92     return 0;
93 }

```

### 3. 树的最小点覆盖

```

1  /*
2  Loj10156
3  每个点都能看到他所连着的边，问你选出最少的点使树上所有的边都能被看到
4  dp[0][i]表示不选i点的最小选择数 则其 += dp[1][son]
5  dp[1][i]表示选i点的最小选择数，则其 += min(dp[0][son], dp[1][son])
6  */
7  #include <cstdio>
8  #include <cstring>
9  #include <algorithm>
10 #include <vector>
11
12 using namespace std;
13 #define ll long long
14 #include <cctype>
15 inline long long IO() {
16     long long x = 0;
17     bool f = false;
18     char c = getchar();
19     while (!isdigit(c)) {
20         if (c == '-') f = true;
21         c = getchar();
22     }
23     while (isdigit(c)) {
24         x = (x << 1) + (x << 3) + (c - '0');
25         c = getchar();
26     }
27     return f ? -x : x;
28 }
29
30 const int maxn = 1e5, maxm = 1e5;
31 const int INF = 0x3f3f3f3f;
32
33 int head[maxn], cnt, dis[maxn];
34
35 //初始化
36 void init() {
37     memset(head, -1, sizeof head);
38     // memset(vis, false, sizeof vis);
39     cnt = 0;
40 }
41
42 struct edges {
43     int to, next;
44     int w;
45     edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
46 } edge[maxm << 1]; //无向图则需要乘2
47
48 inline void add(int u, int v, int w = 0) {
49     if (cnt == 0) init();
50     edge[++cnt] = edges(v, head[u], w);
51     head[u] = cnt;
52 }
53 const int M = 1e4;
54 int n, m, dp[2][M], vis[M];
55
56 void dfs(int u, int fa) {

```



```

57     dp[1][u] = 1;
58     for (int i = head[u]; ~i; i = edge[i].next) {
59         int v = edge[i].to;
60         if (v == fa) continue;
61         dfs(v, u);
62         dp[1][u] += min(dp[0][v], dp[1][v]);
63         dp[0][u] += dp[1][v];
64     }
65 }
66
67 int main() {
68     n = IO();
69     for (int i = 0; i < n; ++i) {
70         int u = IO(), k = IO();
71         while (k--) {
72             int v = IO();
73             add(u, v), add(v, u);
74         }
75     }
76     dfs(0, -1);
77     // printf("%d\n", ans);
78     printf("%d", min(dp[0][0], dp[1][0]));
79     return 0;
80 }

```

#### 4. 树的直径

```

1  /*
2  Loj 10159
3  树的直径：树上最长路径
4  本题要求出所有直径（直径可能不唯一）上的所有点
5  解决方法：每次递归算出结点到其儿子中的最长路径和次长路径，然后相加
6  维护好全局变量ans，最终答案就是ans
7  */
8  #include <cstdio>
9  #include <cstring>
10 #include <algorithm>
11 #include <vector>
12
13 using namespace std;
14 #define ll long long
15 #include <cctype>
16 inline long long IO() {
17     long long x = 0;
18     bool f = false;
19     char c = getchar();
20     while (!isdigit(c)) {
21         if (c == '-') f = true;
22         c = getchar();
23     }
24     while (isdigit(c)) {
25         x = (x << 1) + (x << 3) + (c - '0');
26         c = getchar();
27     }
28     return f ? -x : x;
29 }
30
31 const int maxn = 2e5 + 5, maxm = 2e5 + 5;
32 const int INF = 0x3f3f3f3f;
33
34 int head[maxn], cnt, dis[maxn];
35
36 //初始化
37 void init() {
38     memset(head, -1, sizeof head);
39     // memset(vis, false, sizeof vis);
40     cnt = 0;
41 }
42
43 struct edges {
44     int to, next;
45     int w;
46     edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
47 }edge[maxm << 1]; //无向图则需要乘2
48
49 inline void add(int u, int v, int w = 0) {
50     if (cnt == 0) init();
51     edge[++cnt] = edges(v, head[u], w);
52     head[u] = cnt;
53 }
54
55 const int M = 2e5, inf = 0x3f3f3f3f;
56 int dp[M], t[M], ans = 0, d1[M], d2[M];
57 vector<int> res;
58
59 int dfs(int u, int fa) {

```

```

59     d1[u] = 0, d2[u] = 0;
60     for (int i = head[u]; ~i; i = edge[i].next) {
61         int v = edge[i].to;
62         if (v == fa) continue;
63         int len = dfs(v, u) + 1;
64         if (len >= d1[u]) {
65             d2[u] = d1[u], d1[u] = len;
66         } else if (len > d2[u]) {
67             d2[u] = len;
68         }
69     }
70     ans = max(ans, d1[u] + d2[u]);
71     return d1[u];
72 }
73
74 void dfs(int u, int fa, int d) {
75     for (int i = head[u]; ~i; i = edge[i].next) {
76         int v = edge[i].to;
77         if (v == fa) continue;
78         if (d1[v] == d) dfs(v, u, d - 1), res.push_back(v);
79     }
80 }
81
82 void solve(int u, int fa) {
83     if (d1[u] + d2[u] == ans) {
84         res.push_back(u);
85         if (d1[u] != d2[u]) dfs(u, fa, d2[u] - 1);
86         dfs(u, fa, d1[u] - 1);
87     }
88     for (int i = head[u]; ~i; i = edge[i].next) {
89         int v = edge[i].to;
90         if (v == fa) continue;
91         solve(v, u);
92     }
93 }
94
95 int main() {
96     int n = IO();
97     init();
98     for (int i = 1; i < n; ++i) {
99         int u = IO(), v = IO();
100         add(u, v), add(v, u);
101     }
102     dfs(0, -1);
103     solve(0, -1);
104     sort(res.begin(), res.end());
105     auto x = unique(res.begin(), res.end());
106     auto i = res.begin();
107     while (i != x) {
108         printf("%d\n", *i);
109         i++;
110     }
111     return 0;
112 }

```

## 5. 树的重心

- 树的重心的一些重要性质：
- 一棵树最少有一个重心，最多有两个重心，若有两个重心，则他们相邻（即连有直接边）
- 树上所有点到某个点的距离和里，到重心的距离和最小；若有两个重心，则其距离和相同
- 若以重心为根，则所有子树的大小都不超过整棵树的一半
- 在一棵树上添加或删除一个叶子节点，其重心最多平移一条边的距离
- 两棵树通过连一条边组合成新树，则新树重心在原来两棵树的重心的连线上

```

1  /*
2  Poj 1655
3  树的重心：重心是指树种的一个结点，如果将这个结点删除后剩余的各个连通块中结点个数的最大值最小，则称为树的重心
4  本题需要求出重心，如果有多个输出最小编号的结点，并输出重心被删除后连通块结点个数的最大值
5  */
6  #include <cstdio>
7  #include <cstring>
8  #include <algorithm>
9  #include <vector>
10 #include <cctype>
11 inline long long IO() {
12     long long x = 0;
13     bool f = false;
14     char c = getchar();
15     while (!isdigit(c)) {
16         if (c == '-') f = true;
17         c = getchar();
18     }
19     while (isdigit(c)) {
20         x = (x << 1) + (x << 3) + (c - '0');

```

```

21     c = getchar();
22 }
23 return f ? -x : x;
24 }
25 using namespace std;
26
27 const int maxn = 1e5 + 5, maxm = 2e5 + 5, inf = 0x3f3f3f3f;
28
29 int head[maxn], cnt;
30
31 //初始化
32 void init() { memset(head, -1, sizeof head); cnt = 0; }
33
34 struct edges {
35     int to, next;
36 } edge[maxm << 1]; //无向图则需要乘2
37
38 inline void add(int u, int v) {
39     edge[++cnt].to = v, edge[cnt].next = head[u];
40     head[u] = cnt;
41 }
42
43 int ans, siz;
44 int dfs(int u, int fa, const int &n) {
45     int tot = 1, num = 0;
46     for (int i = head[u]; ~i; i = edge[i].next) {
47         int v = edge[i].to;
48         if (v == fa) continue;
49         int tmp = dfs(v, u, n);
50         tot += tmp, num = max(num, tmp);
51     }
52     int res = max(n - tot, num);
53     if (siz >= res) {
54         if (siz == res) ans = min(u, ans);
55         else ans = u, siz = res;
56     }
57     return tot;
58 }
59
60 void solve() {
61     int n = IO();
62     init(), siz = inf;
63     for (int i = 1; i < n; ++i) {
64         int u = IO(), v = IO();
65         add(u, v), add(v, u);
66     }
67     dfs(1, -1, n);
68     printf("%d %d\n", ans, siz);
69 }
70
71 int main() {
72     int t = IO();
73     while (t--) solve();
74     return 0;
75 }

```

## 6. 树的中心

```

1  /*
2  例题：无
3  树的中心：找出一个点，使该点到其他点的最远距离最小，则这个点就是树的中心
4  解题思路：
5  从u点到其他点的最远距离分为两类
6  1. 从u点向下走的最远距离，用d1[u]表示
7  2. 从u点向上走的最远距离，用up[u]表示
8  则从u点到其他点的最远距离就是 max(d1[u], up[u]);
9  则中心到其他点的最远距离就是 ans = min{dp[i]}
10  其中d1[u]可用求树的直径的方法求出
11  记得同时维护d2[u]即次长距离，和维护最长的路是哪个儿子
12  关于up的计算方式要用父亲节点来更新儿子节点，与求d1是相反的
13  如果u的儿子结点son在最长的路径上则
14      up[son] = w[son] + max(up[u], d2[u])
15  否则
16      up[son] = w[son] + max(up[u], d1[u])
17  输入
18  5
19  2 1 1
20  3 2 1
21  4 3 1
22  5 1 1
23  输出中心到其他节点的最长长度
24  2
25  */
26 #include <cstdio>
27 #include <cstring>
28 #include <algorithm>
29 #include <vector>

```

```

30 using namespace std;
31
32 const int maxn = 1e5, maxm = 2e5, inf = 0x3f3f3f3f;
33
34 int head[maxn], cnt;
35
36 //初始化
37 void init() { memset(head, -1, sizeof head); cnt = 0; }
38
39 struct edges {
40     int to, next;
41     int w;
42 }edge[maxm << 1]; //无向图则需要乘2
43
44 inline void add(int u, int v, int w) {
45     edge[++cnt].to = v, edge[cnt].next = head[u];
46     edge[cnt].w = w, head[u] = cnt;
47 }
48
49 int d1[maxn], d2[maxn], maxv[maxn], up[maxn];
50 int dfs1(int u, int fa) {
51     d1[u] = d2[u] = 0;
52     for (int i = head[u]; ~i; i = edge[i].next) {
53         int v = edge[i].to;
54         if (v == fa) continue;
55         int d = dfs1(v, u) + edge[i].w;
56         if (d >= d1[u]) {
57             d2[u] = d1[u], d1[u] = d;
58             maxv[u] = v;
59         } else if (d > d2[u]) {
60             d2[u] = d;
61         }
62     }
63     return d1[u];
64 }
65 void dfs2(int u, int fa) {
66     for (int i = head[u]; ~i; i = edge[i].next) {
67         int v = edge[i].to;
68         if (v == fa) continue;
69         if (maxv[u] == v) {
70             up[v] = max(up[u], d2[u]) + edge[i].w;
71         } else {
72             up[v] = max(up[u], d1[u]) + edge[i].w;
73         }
74         dfs2(v, u);
75     }
76 }
77
78 int main() {
79     int n;
80     init();
81     scanf("%d\n", &n);
82     for (int i = 1; i < n; ++i) {
83         int u, v, w;
84         scanf("%d %d %d\n", &u, &v, &w);
85         add(u, v, w), add(v, u, w);
86     }
87     dfs1(1, -1);
88     dfs2(1, -1);
89     int res = 0x3f3f3f3f;
90     for (int i = 1; i <= n; ++i) {
91         res = min(res, max(up[i], d1[i]));
92     }
93     printf("%d", res);
94     return 0;
95 }
96

```

## 7. 依赖背包问题

```

1  /*
2  Loj 10154选课
3  学生不可能学完大学开设的所有课程，因此必须在入学时选定自己要学的课程。
4  每个学生可选课程的总数是给定的。请找出一种选课方案使得你能得到的学分最多，
5  并满足先修课优先的原则。假定课程间不存在时间上的冲突。
6
7  输入的第一行包括两个正整数 ， 分别表示待选课程数和可选课程数。
8  接下来 行每行描述一门课，课号依次为 。每行两个数，依次表示这门课先修课课号（若不存在，则该项值为 ）和该门课的学分。
9  输出一行，表示实际所选课程学分之和。
10
11  dp[i][j]代表第i门课程选j个课的最大学分和
12  题中的课程号是从1开始的，并不是一颗树，而是森林，我们假设有一个课程0，连接所有森林的根结点
13  */
14 #include <cstdio>
15 #include <cstring>

```

```

16 #include <algorithm>
17 #include <vector>
18
19 using namespace std;
20 #define ll long long
21 #include <cctype>
22 inline long long IO() {
23     long long x = 0;
24     bool f = false;
25     char c = getchar();
26     while (!isdigit(c)) {
27         if (c == '-') f = true;
28         c = getchar();
29     }
30     while (isdigit(c)) {
31         x = (x << 1) + (x << 3) + (c - '0');
32         c = getchar();
33     }
34     return f ? -x : x;
35 }
36
37 const int maxn = 1e4, maxm = 1e4;
38 const int INF = 0x3f3f3f3f;
39
40 int head[maxn], cnt, dis[maxn];
41
42 //初始化
43 void init() {
44     memset(head, -1, sizeof head);
45     // memset(vis, false, sizeof vis);
46     cnt = 0;
47 }
48
49 struct edges {
50     int to, next;
51     int w;
52     edges(int to = 0, int next = -1, int w = 0) : to(to), next(next), w(w) {}
53 } edge[maxm << 1]; //无向图则需要乘2
54
55 inline void add_edges(int u, int v, int w) {
56     edge[++cnt] = edges(v, head[u], w);
57     head[u] = cnt;
58 }
59 const int M = 310;
60 int n, m, dp[M][M];
61
62 int dfs(int u) {
63     int num = 1;
64     for (int i = head[u]; ~i; i = edge[i].next) {
65         int v = edge[i].to, w = edge[i].w;
66         int m = dfs(v);
67         for (int j = num + m; j; --j) { //01背包, 反向循环
68             for (int k = max(0, j - num); k < j && k <= m; ++k) {
69                 dp[u][j] = max(dp[u][j], dp[v][k] + dp[u][j - k - 1] + w);
70             }
71         }
72         num += m;
73     }
74     return num; //返回包括自己加上子树有多少个节点
75 }
76
77 int main() {
78     n = IO(), m = IO();
79     init();
80     for (int i = 1; i <= n; ++i) {
81         int u = IO(), w = IO();
82         add_edges(u, i, w);
83     }
84     dfs(0);
85     printf("%d", dp[0][m]);
86     return 0;
87 }

```

8. 基环树dp

9. 换根