

ACM-ICPC 模板

—— E.S.Promise

2014/10/17

目录

1. 数据结构.....	4
1.1 Splay	4
1.2 Treap.....	10
1.3 矩形面积并	12
1.4 Heap	13
1.5 RMQ.....	15
2 字符串	16
2.1 Kmp.....	16
2.2 exKMP.....	17
2.3 后缀数组 （DA）	17
后缀数组（DC3）	20
2.4 AC 自动机（array）	23
2.5 AC 自动机（pointer）	25
2.6 字符串 Hash.....	27
3 图论	28
3.1 SPFA.....	28
3.2 Floyd(改进版：可求最小环权值)	30
3.3 Tarjan (无向图求割点).....	32
3.4 BFS 判断二分图	34

4 其他.....	35
4.1 大数类.....	35
4.2 输入挂.....	41

1. 数据结构

1.1 Splay

```
1.  #include <cstdio>
2.  #include <cstring>
3.  #include <iostream>
4.  #include <vector>
5.  #include <algorithm>
6.
7.  #define MAXN 400010
8.
9.  using namespace std;
10.
11. struct Node{
12.     int key, sz, cnt, lazy;
13.     Node *chd[2], *pa;    //左右儿子和父亲
14.     Node(){lazy = 0;}
15.     Node(int x, int y, int z){
16.         key = x, sz = y, cnt = z;
17.         lazy = 0;
18.     }
19.     void push_up(){
20.         sz = chd[0]->sz + chd[1]->sz + cnt;
21.     }
22.     // 要用 lazy 的时候再写,下例为转置。
23.     void push_down()
24.     {
25.         if(lazy)
26.         {
27.             swap(chd[0], chd[1]);
28.             chd[0]->lazy ^= 1;
29.             chd[1]->lazy ^= 1;
30.             lazy = 0;
31.         }
32.     }
33. }nil(0, 0, 0), *NIL = &nil;
34.
35. struct Splay{           //伸展树结构体类型
36.     vector<int> vec;
37.     Node *root;
38.     int ncnt;           //计算 key 值不同的结点数, 注意已经去重了
39.     Node nod[MAXN];
40.     // 首先要初始化
41.     void init(){
42.         NIL->key = NIL->cnt = NIL->sz = NIL->lazy = 0;
43.         root = NIL;
44.         ncnt = 0;
45.     }
46.     //旋转操作, d 为 true 表示右旋
```

```

47. void rotate(Node *x, bool d){
48.     Node *y = x->pa;
49.     y->push_down();
50.     x->push_down();
51.     y->chd[!d] = x->chd[d];
52.     if (x->chd[d] != NIL)
53.         x->chd[d]->pa = y;
54.     x->pa = y->pa;
55.     if (y->pa != NIL){
56.         if (y == y->pa->chd[d])
57.             y->pa->chd[d] = x;
58.         else
59.             y->pa->chd[!d] = x;
60.     }
61.     x->chd[d] = y;
62.     y->pa = x;
63.     y->push_up();
64.     x->push_up();
65. }
66. // 当 target 为 NIL 时，虽然结点 x 在 NIL 下，但是 root 不是 NIL
67. void splay(Node *x, Node *target){    //将 x 伸展到 target 的儿
    子位置处
68.     Node *y;
69.     while (x->pa != target){
70.         y = x->pa;
71.         if (x == y->chd[0]){

```

```

72.             if (y->pa != target && y == y->pa->chd[0])
73.                 rotate(y, true);
74.                 rotate(x, true);
75.             }
76.         else{
77.             if (y->pa != target && y == y->pa->chd[1])
78.                 rotate(y, false);
79.                 rotate(x, false);
80.             }
81.         }
82.         if (target == NIL)
83.             root = x;
84.     }
85.     /*****以上一般不用修改*****/
86.
87. // 根据维护信息不同，相应修改. 插入一个值
88. void insert(int key){
89.     if (root == NIL){
90.         ncnt = 0;
91.         root = &nod[++ncnt];    // 新结点都指向 NIL
92.         root->chd[0] = root->chd[1] = root->pa = NIL;
93.         root->key = key;
94.         root->sz = root->cnt = 1;
95.         return;
96.     }
97.     Node *x = root, *y;

```

```

98.     while (1)
99.     {
100.         x->sz++;
101.         if (key == x->key){
102.             x->cnt++;
103.             x->push_up();
104.             y = x;
105.             break;
106.         }
107.         else if (key < x->key){
108.             if (x->chd[0] != NIL)
109.                 x = x->chd[0];
110.             else{
111.                 x->chd[0] = &nod[++ncnt];
112.                 y = x->chd[0];
113.                 y->key = key;
114.                 y->sz = y->cnt = 1;
115.                 y->chd[0] = y->chd[1] = NIL;
116.                 y->pa = x;
117.                 break;
118.             }
119.         }
120.         else{
121.             if (x->chd[1] != NIL)
122.                 x = x->chd[1];
123.             else{

```

```

124.                 x->chd[1] = &nod[++ncnt];
125.                 y = x->chd[1];
126.                 y->key = key;
127.                 y->sz = y->cnt = 1;
128.                 y->chd[0] = y->chd[1] = NIL;
129.                 y->pa = x;
130.                 break;
131.             }
132.         }
133.     }
134.     splay(y, NIL);
135. }
136. // 通过 键值 去寻找，以下还有通过第几个元素去寻找。
137. Node* search(int key){           //查找一个值，返回指针
138.     if (root == NIL)
139.         return NIL;
140.     Node *x = root, *y = NIL;
141.     while (1){
142.         if (key == x->key){
143.             y = x;
144.             break;
145.         }
146.         else if (key > x->key){
147.             if (x->chd[1] != NIL)
148.                 x = x->chd[1];
149.             else

```

```

150.             break;
151.         }
152.         else{
153.             if (x->chd[0] != NIL)
154.                 x = x->chd[0];
155.             else
156.                 break;
157.         }
158.     }
159.     splay(x, NIL);
160.     return y;
161. }
162. //查找最小值，返回指针
163. Node* searchmin(Node *x){
164.     Node *y = x->pa;
165.     while (x->chd[0] != NIL){    //遍历到最左的儿子就是最小值
166.         x = x->chd[0];
167.     }
168.     splay(x, y);
169.     return x;
170. }
171. //删除一个值
172. void del(int key){
173.     if (root == NIL)
174.         return;
175.     Node *x = search(key), *y;

```

```

176.     if (x == NIL)
177.         return;
178.     if (x->cnt > 1){
179.         x->cnt--;
180.         x->push_up();
181.         return;
182.     }
183.     else if (x->chd[0] == NIL && x->chd[1] == NIL){
184.         init();
185.         return;
186.     }
187.     else if (x->chd[0] == NIL){
188.         root = x->chd[1];
189.         x->chd[1]->pa = NIL;
190.         return;
191.     }
192.     else if (x->chd[1] == NIL){
193.         root = x->chd[0];
194.         x->chd[0]->pa = NIL;
195.         return;
196.     }
197.     y = searchmin(x->chd[1]);
198.     y->pa = NIL;
199.     y->chd[0] = x->chd[0];
200.     x->chd[0]->pa = y;
201.     y->push_up();

```

```

202.     root = y;
203. }
204. int rank(int key){           //求结点高度
205.     Node *x = search(key);
206.     if (x == NIL)
207.         return 0;
208.     return x->chd[0]->sz + 1    /* or x->cnt */;
209. }
210. Node* findkth(int kth){       //查找第 k 小的值
211.     if (root == NIL || kth > root->sz)
212.         return NIL;
213.     Node *x = root;
214.     while (1){
215.         if (x->chd[0]->sz + 1 <= kth && kth <= x->chd[0]->sz + x->cnt)
216.             break;
217.         else if (kth <= x->chd[0]->sz)
218.             x = x->chd[0];
219.         else{
220.             kth -= x->chd[0]->sz + x->cnt;
221.             x = x->chd[1];
222.         }
223.     }
224.     splay(x, NIL);
225.     return x;
226. }
227. //找第 x 个元素

```

```

228. Node *getNth(Node *rt, int x)
229. {
230.     if(rt == NIL) return NIL;
231.     rt->push_down();
232.     int l = rt->chd[0]->sz;
233.     if(x <= l)
234.         return getNth(rt->chd[0], x);
235.     else if(x == l+1)
236.         return rt;
237.     else
238.         return getNth(rt->chd[1], x-l-1);
239. }
240. //得到以 rt 为根的子树的序列
241. void get_seq(Node *rt)
242. {
243.     if(rt == NIL) return ;
244.     rt->push_down();
245.     get_seq(rt->chd[0]);
246.     vec.push_back(rt->key);
247.     get_seq(rt->chd[1]);
248. }
249.
250. }sp;
251.
252. // HDU 3487 CUT 和 FLIP
253.

```



```

254. int n, m;
255. int main(){
256.     while(~scanf("%d%d", &n, &m))
257.     {
258.         if(n == -1 && m == -1) break;
259.         sp.init();
260.         for(int i = 0; i <= n+1; i++)
261.             sp.insert(i);
262.         char cmd[10];
263.         while(m--) {
264.             scanf("%s", cmd);
265.             if(!strcmp(cmd, "CUT"))
266.             {
267.                 int l, r, z; scanf("%d%d%d", &l, &r, &z);
268.                 l++, r++, z++;
269.                 Node *x, *y, *p;
270.                 x = sp.getNth(sp.root, l-1), y = sp.getNth(sp.root, r+1);
271.                 sp.splay(x, NIL);
272.                 sp.splay(y, x);
273.                 x = y->chd[0];
274.                 y->chd[0] = NIL;
275.                 y->push_up();
276.                 y = sp.getNth(sp.root, z);
277.                 sp.splay(y, NIL);
278.                 p = sp.getNth(sp.root, z+1);
279.                 sp.splay(p, y);

```

```

280.                 p->chd[0] = x;
281.                 x->pa = p;
282.                 p->push_up();
283.             }
284.         else
285.         {
286.             int l, r; scanf("%d%d", &l, &r);
287.             l++; r++;
288.             Node *x, *y;
289.             x = sp.getNth(sp.root, l-1), y = sp.getNth(sp.root, r+1);
290.             sp.splay(x, NIL);
291.             sp.splay(y, x);
292.             x = y->chd[0];
293.             x->lazy ^= 1;
294.         }
295.     }
296.     sp.vec.clear();
297.     Node *x, *y;
298.     x = sp.getNth(sp.root, 1);
299.     y = sp.getNth(sp.root, n+2);
300.     sp.splay(x, NIL);
301.     sp.splay(y, x);
302.     sp.get_seq(y->chd[0]);
303.     printf("%d", sp.vec[0]);
304.     for(int i = 1; i < sp.vec.size(); i++)
305.         printf(" %d", sp.vec[i]);

```

```

306.         printf("\n");
307.     }

```

```

308.     return 0;
309. }

```

1.2 Treap

```

1.  #include <cstdio>
2.  #include <cstdlib>
3.  #include <iostream>
4.  #include <cstring>
5.
6.  #define MAXN 20010
7.  #define MAXM 60010
8.  #define MAXQ 500010
9.  using namespace std;
10.
11. class TreapNode
12. {
13. public:
14.     int pri, val, sz;
15.     TreapNode *chd[2];
16.     TreapNode(int _val): val(_val)
17.     {
18.         chd[0] = chd[1] = NULL;
19.         pri = rand();

```

```

20.         sz = 1;
21.     }
22.     bool operator < (const TreapNode &para) const
23.     {
24.         return pri < para.pri;
25.     }
26.
27.     int cmp(int x)
28.     {
29.         if(x == val)
30.             return -1;
31.         return x >= val;
32.     }
33.     void maintain()
34.     {
35.         sz = 1;
36.         if(chd[0]) sz += chd[0]->sz;
37.         if(chd[1]) sz += chd[1]->sz;
38.     }

```

```

39. };
40.
41. void rotate(TreapNode *&rt, int d)
42. {
43.     TreapNode *k = rt->chd[d^1];
44.     rt->chd[d^1] = k->chd[d];
45.     k->chd[d] = rt;
46.     rt->maintain(); //先子树 maintain, 此时 k 相对于 rt 已变成 root
47.     k->maintain();
48.     rt = k;
49. }
50.
51. void insert(TreapNode *&rt, int x)
52. {
53.     if(rt == NULL)
54.         rt = new TreapNode(x);
55.     else
56.     {
57.         int d = (x >= rt->val);
58.         insert(rt->chd[d], x);
59.         if(rt->chd[d]->pri > rt->pri)
60.             rotate(rt, 1^d);
61.     }
62.     rt->maintain();
63. }
64.
65. void remove(TreapNode *&rt, int x)
66. {
67.     int d = rt->cmp(x);
68.     if(d == -1)

```

```

69.     {
70.         TreapNode *u = rt;
71.         if(rt->chd[0] && rt->chd[1])
72.         {
73.             int d2 = (rt->chd[0]->pri > rt->chd[1]->pri);
74.             rotate(rt, d2);
75.             remove(rt->chd[d2], x);
76.         }
77.         else
78.         {
79.             if(rt->chd[0] == NULL)
80.                 rt = rt->chd[1];
81.             else
82.                 rt = rt->chd[0];
83.             delete u;
84.         }
85.     }
86.     else
87.         remove(rt->chd[d], x);
88.     if(rt != NULL)
89.         rt->maintain();
90. }
91.
92. int kth(TreapNode *rt, int k)
93. {
94.     if(rt == NULL || k < 0 || k > rt->sz)
95.         return 0;
96.     int s = rt->chd[1] == NULL ? 0 : rt->chd[1]->sz;
97.     if(k == s + 1)
98.         return rt->val;

```

```

99.     else if(k <= s)
100.         return kth(rt->chd[1], k);
101.     else
102.         return kth(rt->chd[0], k-s-1);
103. }

```

1.3 矩形面积并

```

1.  #include <cstdio>
2.  #include <iostream>
3.  #include <cstring>
4.  #include <cmath>
5.  #include <string>
6.  #include <queue>
7.  #include <map>
8.  #include <vector>
9.  #include <algorithm>
10. #include <set>
11.
12. #define DEBUG 0
13. #define MP make_pair
14. #define lson id<<1, l, mid
15. #define rson id<<1|1, mid+1, r
16. #define A first
17. #define B second
18. #define INF 0x3fffffff
19. #define OUTSTARS printf("*****\n");
20. #define MAXN 1005
21.

```

```

22. using namespace std;
23.
24. typedef long long LL;
25. typedef pair<pair<double, double>, pair<double, int>> > PDDDI;
26.
27. using namespace std;
28.
29. vector<PDDDI> seg;
30. set<double> DX;
31. vector<double> D;
32. int cnt[MAXN];
33. double S[MAXN];
34.
35. void push_up(int id, int l, int r)
36. {
37.     if(cnt[id])
38.         S[id] = D[r] - D[l - 1];
39.     else if(l == r)
40.         S[id] = 0;
41.     else
42.         S[id] = S[id<<1] + S[id<<1|1];
43. }
44.
45. void update(int id, int l, int r, int x, int y, int c)
46. {
47.     if(x <= l && y >= r) {
48.         cnt[id] += c;
49.         push_up(id, l, r);
50.         return ;
51.     }

```

```

52.     int mid = (l + r) >> 1;
53.     if(x <= mid) update(lson, x, y, c);
54.     if(y > mid) update(rson, x, y, c);
55.     push_up(id, l, r);
56. }
57.
58. int main()
59. {
60.     int n, t = 1;
61.     double x1, y1, x2, y2;
62.     while(~scanf("%d", &n))
63.     {
64.         if(!n) break;
65.         DX.clear();
66.         seg.clear();
67.         memset(cnt, 0, sizeof(cnt));
68.         memset(S, 0, sizeof(S));
69.         for(int i = 0; i < n; i++) {
70.             scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
71.             DX.insert(x1);
72.             DX.insert(x2);
73.             seg.push_back(MP(MP(y1, x1), MP(x2, 1)));
74.             seg.push_back(MP(MP(y2, x1), MP(x2, -1)));
75.         }
76.         D = *(new vector<double>(DX.begin(), DX.end()));
77.         sort(seg.begin(), seg.end());
78.         double ans = 0, last;
79.         for(int i = 0; i < 2 * n; i++) {
80.             if(i) ans += S[1] * (seg[i].A.A - last);
81.             int l = lower_bound(D.begin(), D.end(), seg[i].A.B) - D.begin();

```

```

82.             int r = lower_bound(D.begin(), D.end(), seg[i].B.A) - D.begin();
83.             update(1, 1, D.size(), l+1, r, seg[i].B.B);
84.             last = seg[i].A.A;
85.         }
86.         printf("Test case #%d\nTotal explored area: %.2lf\n\n", t++, ans);
87.     }
88.     return 0;
89. }

```

1.4 Heap

```

1.  #include <iostream>
2.  using namespace std;
3.  #define MAX_LONG 2147483647
4.
5.  class binary_heap
6.  {
7.      protected:
8.          long cnt;
9.          long h[100000];
10.         binary_heap() {cnt=0;h[0]=0;}//其实 h[0]没用
11.         long pre(long n) {return n>>1;}
12.         long lch(long n) {return n<<1;}
13.         long rch(long n) {return (n<<1)+1;}
14.
15.     public:
16.         long size(){return cnt;}
17. };

```

```

18. class maxheap:public binary_heap
19. {
20.     public:
21.     void ins(long n)
22.     {
23.         int p=++cnt;//元素个数+1, p 指向堆底
24.         while(p>1 && n>h[pre(p)])//若没到堆顶 且 待插入元素大于 p 的
            父亲
25.         {
26.             h[p]=h[pre(p)];//则把 p 的父亲移到他儿子的地方(就是 p
            的地方)
27.             p=pre(p);//p 指向它的父亲
28.         }
29.         h[p]=n;//最后把待插入元素放入 p 的地方
30.     }
31.     long pop()
32.     {
33.         if(cnt==0) return -MAX_LONG;//返回-MAX_LONG 视为错误
34.         long p=1;
35.         long ans=h[1];
36.         long tmp=h[cnt--];//取出堆底元素然
37.         long to;
38.         while(p<=cnt)//若未到栈底
39.         {
40.             to=-MAX_LONG;//init
41.             if(lch(p)<=cnt && tmp<h[lch(p)])
42.                 //若有左孩子且比取出的栈底元素大
43.                 to=lch(p);
44.             if(rch(p)<=cnt && tmp<h[rch(p)] && h[to]<h[rch(p)])
45.                 to=rch(p);

```

```

46.             if(to!=-MAX_LONG) {h[p]=h[to];p=to;}
47.             else{h[p]=tmp;break;}
48.         }
49.
50.         return ans;
51.     }
52. };
53. class minheap:public binary_heap
54. {
55.     public:
56.     void ins(long n)
57.     {
58.         int p=++cnt;//元素个数+1, p 指向堆底
59.         while(p>1 && n<h[pre(p)])//若没到堆顶 且 待插入元素小于 p 的
            父亲
60.         {
61.             h[p]=h[pre(p)];//则把 p 的父亲移到他儿子的地方(就是 p
            的地方)
62.             p=pre(p);//p 指向它的父亲
63.         }
64.         h[p]=n;//最后把待插入元素放入 p 的地方
65.     }
66.     long pop()
67.     {
68.         if(cnt==0) return -MAX_LONG;//返回-MAX_LONG 视为错误
69.         long p=1;
70.         long ans=h[1];
71.         long tmp=h[cnt--];//取出堆底元素然
72.         long to;
73.         while(p<=cnt)//若未到栈底

```

```

74.         {
75.             to=-MAX_LONG;//init
76.             if(lch(p)<=cnt && tmp>h[lch(p)])//若有左孩子且比取出的栈
           底元素小
77.                 to=lch(p);
78.             if(rch(p)<=cnt && tmp>h[rch(p)] && h[to]>h[rch(p)])//若有右
           孩子 且 比取出的栈底元素小 且 小于左孩子
79.                 to=rch(p);
80.             if(to!=-MAX_LONG) {h[p]=h[to];p=to;}//如果比左右孩子都
           小, 就说明找对位置了, 把取出的栈底放上去; 否则交换
81.             else{h[p]=tmp;break;}
82.         }
83.
84.     return ans;
85. }
86. };
87.
88. minheap h1;
89.
90. int main()
91. {
92.     long tmp;
93.     while(cin >>tmp)
94.         h1.ins(tmp);
95.     cout <<endl<<"SIZE:"<<h1.size()<<endl<<endl;
96.     for(long i=h1.size();i>0;i--)
97.         cout <<h1.pop()<<endl;
98.     //system("pause");
99. }

```

1.5 RMQ

```

1.  #include <cstdio>
2.  #include <algorithm>
3.  using namespace std;
4.  const int MAXN = 50010;
5.  int dp[MAXN][20];
6.  int lg2[MAXN];      //求 log2
7.
8.  //初始化 RMQ, b 数组下标从 1 开始, 从 0 开始简单修改
9.  void initRMQ(int n, int b[])
10. {
11.     lg2[0] = -1;
12.     for(int i = 1; i <= n;i++)
13.     {
14.         lg2[i] = ((i&(i-1)) == 0)?lg2[i-1]+1:lg2[i-1];
15.         dp[i][0] = b[i];
16.     }
17.     for(int j = 1; j <= lg2[n];j++)
18.         for(int i = 1;i + (1<<j) -1 <= n;i++)
19.             dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
20. }
21. //查询最大值
22. int rmq(int x,int y)
23. {
24.     int k = lg2[y-x+1];
25.     return max(dp[x][k], dp[y-(1<<k)+1][k]);
26. }
27.
28. // 测试

```

```

29. int num[MAXN];
30. int main()
31. {
32.     for(int i = 1; i <= 10; i++)
33.         num[i] = i;
34.     initRMQ(10, num);
35.     printf("%d\n", rmq(1, 1));
36.     return 0;
37. }

```

2 字符串

2.1 Kmp

```

38. #include <cstring>
39. #include <cstdio>
40. #include <iostream>
41. #include <cmath>
42.
43. #define MAXN 1000005
44.
45. using namespace std;
46.
47. char str[MAXN], T[MAXN];
48. int next[MAXN];
49. int lenT, lenS;
50.
51. void get_next(char *T, int lenT)

```

```

52. {
53.     next[0] = -1;           //next[i]=j, 表示 str[0..j]=str[i-j..i]
54.     int j = next[0];
55.     for(int i = 1; i < lenT; i++)
56.     {
57.         while(j >= 0 && T[i] != T[j+1]) j = next[j];
58.         if(T[i] == T[j+1]) j++;
59.         next[i] = j;
60.     }
61. }
62.
63. /**
64. *   cnt  用来计算出现次数
65. *   或者只返回第一次出现的下标
66. */
67. int kmp(char *S, char *T)
68. {
69.     int j = next[0];
70.     int cnt = 0;
71.     for(int i = 0; i < lenS; i++)
72.     {
73.         while(j >= 0 && S[i] != T[j+1]) j = next[j];
74.         if(S[i] == T[j+1]) j++;
75.         if(j == lenT - 1)
76.         {
77.             return i - lenT + 1;
78.             // 如果要返回出现次数，或是记录多次出现的位置，不用 return
79.             j = next[j];
80.             cnt++;
81.         }

```



```

82.     }
83. }
84. //    求循环节
85. int main()
86. {
87.     while(scanf("%s", T))
88.     {
89.         if(T[0] == '.') break;
90.         lenT = strlen(T);
91.         get_next(T, lenT);
92.         int times = sqrt(lenT);
93.         int ans = 1;
94.         int tp = lenT - 1 - next[lenT-1];
95.         if(lenT % tp == 0)
96.             ans = lenT / tp;
97.
98.         printf("%d\n", ans);
99.     }
100.    return 0;
101. }
102. /**
103. 当模式串为: abababcabcbabb next[]={-1, -1, 0, 1, 2, 3, -1, 0, 1, -1, 0, 1, -1 }
104. */

```

2.2 exKMP

2.3 后缀数组 (DA)

```

1.  #include <iostream>
2.  #include <cstdio>
3.  #include <cstring>
4.  #include <algorithm>
5.
6.  #define MAXN 1000005
7.
8.  using namespace std;
9.
10. /**
11.  * rank 下标从 0 开始, 值 0 - n-1 的排列
12.  * sa   从 1 开始,因为最后一个字符(最小的)排在第 0 位
13.  * high 从 1 开始,因为表示的是 sa[i-1]和 sa[i]
14.  */
15.
16. int rank[MAXN], sa[MAXN], X[MAXN], Y[MAXN], high[MAXN], init[MAXN];
17. int buc[MAXN];
18.
19. void calhigh(int n)
20. {
21.     int i, j, k = 0;
22.     for(i = 1; i <= n; i++) rank[sa[i]] = i;
23.     for(i = 0; i < n; high[rank[i++]] = k)
24.         for(k ? k-- : 0, j = sa[rank[i]-1]; init[i+k] == init[j+k]; k++)
25.             ;
26. }
27.
28. bool cmp(int *r,int a,int b,int l)

```

```

29. {
30.     return (r[a] == r[b] && r[a+l] == r[b+l]);
31. }
32.
33. void suffix(int n,int m = 128) {
34.
35.     int i , l , p , *x = X , *y = Y;
36.     for(i = 0 ; i < m ; i++) buc[i] = 0;
37.     for(i = 0 ; i < n ; i++) buc[ x[i] = init[i] ] ++;
38.     for(i = 1 ; i < m ; i++) buc[i] += buc[i-1];
39.     for(i = n - 1; i >= 0 ; i--) sa[ --buc[ x[i] ] ] = i;
40.     for(l = 1,p = 1 ; p < n ; m = p , l *= 2)
41.     {
42.         p = 0;
43.         for(i = n-l ; i < n ; i++) y[p++] = i;
44.         for(i = 0 ; i < n ; i++) if(sa[i] >= l) y[p++] = sa[i] - l;
45.         for(i = 0 ; i < m ; i++) buc[i] = 0;
46.         for(i = 0 ; i < n ; i++) buc[ x[y[i]] ] ++;
47.         for(i = 1 ; i < m ; i++) buc[i] += buc[i-1];
48.         for(i = n - 1; i >= 0 ; i--) sa[ --buc[ x[y[i]] ] ] = y[i];
49.         for(swap(x,y) , x[sa[0]] = 0 , i = 1 , p = 1 ; i < n ; i++)
50.             x[ sa[i] ] = cmp(y,sa[i-1],sa[i],l) ? p-1 : p++;
51.     }
52.     calhigh(n-1);           //后缀数组关键是求出 high,所以求 sa 的时候顺
    便把 rank 和 high 求出来
53. }
54.
55. int dp[MAXN][20];
56. int lg2[MAXN];           //求 log2
57.

```

```

58. //初始化 RMQ, b 数组下标从 1 开始, 从 0 开始简单修改
59. void initRMQ(int n, int b[])
60. {
61.     lg2[0] = -1;
62.     for(int i = 1; i <= n; i++)
63.     {
64.         lg2[i] = ((i&(i-1)) == 0)?lg2[i-1]+1:lg2[i-1];
65.         dp[i][0] = b[i];
66.     }
67.     for(int j = 1; j <= lg2[n]; j++)
68.         for(int i = 1; i + (1<<j) - 1 <= n; i++)
69.             dp[i][j] = min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
70. }
71. //查询最大值
72. int rmq(int x,int y)
73. {
74.     int k = lg2[y-x+1];
75.     return min(dp[x][k], dp[y-(1<<k)+1][k]);
76. }
77.
78. // 如果要求 lcp 先初始化 ST 表
79. // 询问直接用 rmq(int x, int y)
80. void cal_lcp(int n)
81. {
82.     initRMQ(n, high);
83. }
84.
85.
86. /*****
87. **   n 为数组长度,下标 0 开始

```

```

88. ** 将初始数据,保存在 init 里,并且保证每个数字都比 0 大
89. **  m = max{ init[i] } + 1
90. **  一般情况下大多是字符操作,所以 128 足够了
91. *****/
92.
93. int num[MAXN];
94. int n, k;
95. int cnt, idx[1000005];
96.
97. /**
98. *    check: 分组 check 函数 注意: i <= n 即使这里 high[n]的值为 0, 且无
    实际意义, 也不能漏。
99. *    因为当 i == n 的时候即是检查最后一个分组。
100. */
101.
102. bool check(int x)
103. {
104.     int cnt = 1;
105.     for(int i = 1; i <= n; i++)
106.     {
107.         if(high[i] < x)
108.         {
109.             if(cnt >= k) return true;
110.             cnt = 1;
111.         }
112.         else
113.             cnt++;
114.     }
115.     return false;
116. }

```

```

117.
118.
119. void test()
120. {
121.     char str[10] = {"aabaaaab"};
122.     int len = strlen(str);
123.
124.     for(int i = 0; i < len; i++)
125.         init[i] = str[i];
126.     init[len] = 0;
127.     suffix(len+1);
128.     for(int i = 1; i <= len; i++)
129.         printf("%d ", high[i]);
130.     printf("\n");
131.     cal_lcp(len);
132.     printf("lcp(1, 6)=%d\n", rmq(1, 6));
133. }
134.
135. int main()
136. {
137.
138.     test();
139.
140.     while(~scanf("%d%d", &n, &k))
141.     {
142.         cnt = 1;
143.         for(int i = 0; i < n; i++) {
144.             scanf("%d", num+i);
145.             init[i] = num[i];
146.         }

```

```

147.     sort(num, num+n);
148.     idx[num[0]] = cnt;
149.     for(int i = 1; i < n; i++)
150.         idx[num[i]] = num[i] == num[i-1] ? cnt : ++ cnt;
151.     for(int i = 0; i < n; i++)
152.         init[i] = idx[init[i]];
153.     init[n++] = 0;                // 最后一位补 0   n 表示 0 的下一位
    (用来分组) 实际只要用到 n-1
154.     suffix(n, cnt+1);            // n 必须是 最后一位 0 的下一位
155.     int bi;
156.     int l = 1, r = n, ans = 1;
157.     while(l <= r)
158.     {
159.         bi = (l + r) >> 1;
160.         bool res = check(bi);
161.         if(res)
162.         {
163.             l = bi + 1;
164.             ans = max(ans, bi);
165.         }
166.         else
167.             r = bi - 1;
168.     }
169.     printf("%d\n", ans);
170. }
171.
172. return 0;
173. }
174.
175. /**

```

```

176. *   sa[1-len]      = {3 4 5 0 6 1 7 2};
177. *   rank[0-len-1] = {4 6 8 1 2 3 5 7};
178. *   high[1-len]   = {0 3 2 3 1 2 0 1};
179. *
180. *   aaaab
181. *   aaab
182. *   aab
183. *   aabaaaaab
184. *   ab
185. *   abaaaaab
186. *   b
187. *   baaaaab
188. */

```

后缀数组 (DC3)

```

1.  #include <cstdlib>
2.  #include <cstdio>
3.  #include <iostream>
4.  #include <cstring>
5.  #include <string>
6.  #include <cmath>
7.  #include <algorithm>
8.
9.  #define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
10. #define G(x) ((x) < tb ? (x) * 3 + 1 : ((x) - tb) * 3 + 2)
11. #define cmp1(r, a, b) (r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2])

```

```

12. #define cmp3(r, a, b) (r[a] < r[b] || r[a] == r[b] && wv[a+1] < wv[b+1])
13. #define cmp2(k, r, a, b) (k == 2 ? (r[a] < r[b] || r[a] == r[b] && cmp3(r, a+1,
    b+1)):cmp3(r, a, b))
14.
15. using namespace std;
16. typedef long long LL;
17. const int M = 20;
18. const int N = (1<M);
19.
20. /**
21. *   sa 数组从 sa[1]到 sa[n], 存储的是 0 到 n-1 的排列
22. *   rank 数组从 rank[0]到 rank[n-1], 存储的是 1 到 n 的排列
23. *   rank[i]记录的是以 i 为起点的后缀的排名
24. *   high[i]记录 lcp(i, i-1)
25. */
26.
27. class SA
28. {
29. public:
30.     int rank[N], sa[3*N], init[3*N], high[N], n;
31.     int buc[N], m, wv[N], i, j, k;
32.     int log[N], rmq[M][N];
33.     SA()
34.     {
35.         log[0] = -1;
36.         for(int i = 1; i < N; i++)
37.             log[i] = (i & (i-1)) ? log[i-1] : log[i-1] + 1;
38.     }
39.
40.     inline void sort(int *r, int *a, int *b, int n, int m)

```

```

41.     {
42.         for(i = 0; i < n; i++) wv[i] = r[a[i]];
43.         for(i = 0; i < m; i++) buc[i] = 0;
44.         for(i = 0; i < n; i++) buc[wv[i]] ++ ;
45.         for(i = 1; i < m; i++) buc[i] += buc[i-1];
46.         for(i = n - 1; i >= 0; i--) b[-- buc[wv[i]]] = a[i];
47.         return ;
48.     }
49.
50.     inline void suffix_dc3(int *r, int *sa, int n, int m)
51.     {
52.         int *rn = r + n;
53.         int *san = sa + n, ta = 0, tb = (n + 1) / 3;
54.         int tbc = 0, p, *wa = rank, *wb = high;
55.         r[n] = r[n+1] = 0;
56.         for(int i = 0; i < n; i++)
57.             if(i % 3 != 0) wa[tbc++] = i;
58.         sort(r+2, wa, wb, tbc, m);
59.         sort(r+1, wb, wa, tbc, m);
60.         sort(r, wa, wb, tbc, m);
61.         for(p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)
62.             rn[F(wb[i])] = cmp1(r, wb[i-1], wb[i]) ? p - 1 : p ++ ;
63.         if(p < tbc)
64.             suffix_dc3(rn, san, tbc, p);
65.         else
66.             for(i = 0; i < tbc; i++)
67.                 san[rn[i]] = i;
68.         for(i = 0; i < tbc; i++)
69.             if(san[i] < tb)
70.                 wb[ta++] = san[i] * 3;

```

```

71.     if(n % 3 == 1)
72.         wb[ta++] = n - 1;
73.     sort(r, wb, wa, ta, m);
74.     for(i = 0; i < tbc; i++)
75.         wv[wb[i]] = G(san[i]);
76.     for(i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
77.         sa[p] = cmp2(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
78.     for( ; i < ta; sa[p++] = wa[i++])
79.         ;
80.     for( ; j < tbc; sa[p++] = wb[j++])
81.         ;
82. }
83.
84. inline int exec(char *in)
85. {
86.     for(int &p = n = m = 0; in[p]; p++)
87.     {
88.         init[p] = in[p];
89.         m = max(m, init[p]+1);
90.     }
91.     init[n] = 0;
92.     suffix_dc3(init, sa, n+1, m);
93.     for(i = 1; i <= n; i++)
94.         rank[sa[i]] = i;
95.     for(i = 0, k = 0; i < n; high[rank[i++]] = k)
96.         for(k ? k-- : 0, j = sa[rank[i]-1]; init[i+k] == init[j+k]; k++)
97.             ;
98.     for(i = 1; i <= n; i++)
99.         rmq[0][i] = high[i];
100.    for(int i = 1; i <= log[n]; i++)

```

```

101.        for(int j = 1; j <= n-(1<<i)+1; j++)
102.            rmq[i][j] = min(rmq[i-1][j], rmq[i-1][j+(1<<i>>1)]);
103.    return n;
104. }
105.
106. /** lcp(rank[i],rank[j]) 询问 i,j 后缀的最长公共前缀 */
107. inline int lcp(int a, int b)
108. {
109.     if(a == b) return n - sa[a];
110.     if(a > b)
111.         swap(a, b);
112.     int t = log[b-a];
113.     return min(rmq[t][a+1], rmq[t][b-(1<<t)+1]);
114. }
115. } sa;
116.
117. char str[N] = {"aabaaaab"};
118. int len = 8;
119. int main()
120. {
121.     sa.exec(str);
122.     freopen("out.txt", "w", stdout);
123.
124.     for(int i = 1; i <= len; i++)
125.         printf("%s\n", str+sa.sa[i]);
126.
127.     for(int i = 1; i <= len; i++)
128.         printf("%d ", sa.sa[i]);
129.     printf("\n");
130.     for(int i = 0; i < len; i++)

```

```

131.     printf("%d ", sa.rank[i]);
132.     printf("\n");
133.     for(int i = 1; i <= len; i++)
134.         printf("%d ", sa.high[i]);
135.
136.     return 0;
137. }
138.
139. /**
140. *   sa[1-len]      = {3 4 5 0 6 1 7 2};
141. *   rank[0-len-1] = {4 6 8 1 2 3 5 7};
142. *   high[1-len]   = {0 3 2 3 1 2 0 1};
143. *
144. *   aaaab
145. *   aaab
146. *   aab
147. *   aabaaaab
148. *   ab
149. *   abaaaab
150. *   b
151. *   baaaab
152. */

```

2.4 AC 自动机 (array)

```

1. #include <iostream>
2. #include <cstdio>
3. #include <cstring>
4. #include <queue>

```

```

5.
6. #define MAXN 500005
7. #define CHILDREN 26
8. using namespace std;
9.
10. class ACAutomaton
11. {
12. public:
13.     int sz;
14.     int val[MAXN];
15.     int fail[MAXN];
16.     int next[MAXN][CHILDREN];
17.     int ID[300];
18.     queue<int> que;
19.
20.     /** 初始化 ID 映射 */
21.     ACAutomaton()
22.     {
23.         fail[0] = 0;
24.         sz = 1;
25.     }
26.
27.     /** 初始化 AC 机, 如 sz, val[], queue */
28.     void reset()
29.     {
30.         sz = 1;
31.         memset(next[0], 0, sizeof(next[0]));
32.     }
33.
34.     void insert(char *str, int d)

```

```

35.  {
36.      int tp = 0;
37.      for(int i = 0; str[i]; i++)
38.      {
39.          int idx = str[i] - 'a';    //取映射, 如果有 ID[], 则 ID[str[i]]
40.          if(!next[tp][idx])
41.          {
42.              //printf("%c %d", str[i], sz);
43.              next[tp][idx] = sz;
44.              memset(next[sz], 0, sizeof(next[sz]));
45.              val[sz++] = 0;
46.          }
47.          tp = next[tp][idx];
48.      }
49.      val[tp] += d;                // 考虑重复模式串
50.  }
51.
52.  void build()
53.  {
54.      que.push(0);
55.      while(!que.empty())
56.      {
57.          int cur = que.front(); que.pop();
58.          for(int i = 0; i < CHILDREN; i++)
59.          {
60.              int &tp = next[cur][i];
61.              if(tp)
62.              {
63.                  que.push(tp);
64.                  fail[tp] = cur ? next[fail[cur]][i] : 0;

```

```

65.                      // 判断 cur 是否为 root
66.              }
67.          else if(cur)
68.              tp = next[fail[cur]][i];
69.      }
70.  }
71.  }
72.
73.  /** HDU 2222 */
74.  int solve(char *str)
75.  {
76.      int ret = 0, tp = 0;
77.      for(int i = 0; str[i]; i++)
78.      {
79.          int idx = str[i] - 'a';    //OR ID[str[i]];
80.          while(tp && !next[tp][idx])
81.              tp = fail[tp];
82.          if(next[tp][idx])
83.              tp = next[tp][idx];
84.          for(int cur = tp; cur && val[cur] != -1; cur = fail[cur])
85.          {
86.              ret += val[cur];
87.              val[cur] = -1;
88.          }
89.      }
90.      return ret;
91.  }
92.
93.  }ac;
94.  char str[55], main_str[1000005];

```



```

95. int main()
96. {
97.     int c; scanf("%d", &c);
98.     while(c --)
99.     {
100.         int n; scanf("%d", &n);
101.         ac.reset();
102.         while(n --)
103.         {
104.             scanf("%s", str);
105.             ac.insert(str, 1);
106.         }
107.         ac.build();
108.         scanf("%s", main_str);
109.         printf("%d\n", ac.solve(main_str));
110.     }
111.     return 0;
112. }

```

2.5 AC 自动机（pointer）

```

1. #include <cstdio>
2. #include <cstring>
3. #include <iostream>
4.
5. #define CHILDREN 26
6. #define MAXN 1000005
7.
8. struct Node {

```

```

9.     int cnt; //是否为单词最后一个节点
10.    Node *next[CHILDREN];
11.    Node *fail;
12.    Node() {
13.        fail = NULL;
14.        cnt = 0;
15.        memset(next, NULL, sizeof(next));
16.    }
17. };
18.
19. class ACAutomaton
20. {
21. public:
22.     Node *root;
23.     Node *que[MAXN]; //队列，bfs 构造失败指针
24.     int head, rear;
25.     void reset()
26.     {
27.         head = rear = 0;
28.         root = new Node();
29.     }
30.
31.     void insert(char *str)
32.     {
33.         Node *tp = root;
34.         int idx;
35.         for(int i = 0; str[i]; i++)
36.         {
37.             int idx = str[i] - 'a';
38.             if(tp->next[idx] == NULL)

```

```

39.         tp->next[idx] = new Node();
40.         tp = tp->next[idx];
41.     }
42.     tp->cnt ++;
43. }
44.
45. void build()
46. {
47.     root->fail = NULL;
48.     que[rear++] = root;
49.     while(head < rear)
50.     {
51.         Node *tp = que[head++];
52.         Node *p = NULL;
53.         for(int i = 0; i < 26; i ++)
54.         {
55.             if(tp->next[i])
56.             {
57.                 if(tp == root)
58.                     tp->next[i]->fail = root;
59.                 else
60.                 {
61.                     p = tp->fail;
62.                     while(p) {
63.                         if(p->next[i])
64.                         {
65.                             tp->next[i]->fail = p->next[i];
66.                             break;
67.                         }
68.                         p = p->fail;

```

```

69.         }
70.         if(!p)
71.             tp->next[i]->fail = root;
72.     }
73.     que[rear++] = tp->next[i];
74. }
75. }
76. }
77. }
78.
79. /** HDU 2222
80.  ** 输入 n 个模式串，一个主串，AC 机跑一遍，找出现了多少个单词，
    一个单词最多只出现一次
81.  **/
82. int solve(char *str)
83. {
84.     int ret = 0, idx;
85.     Node *p = root;
86.     for(int i = 0; str[i]; i ++)
87.     {
88.         idx = str[i] - 'a';
89.         while(!p->next[idx] && p != root)
90.             p = p->fail;
91.         p = p->next[idx];
92.         p = !p ? root : p;
93.         Node *tp = p;
94.         while(tp != root && tp->cnt != -1)
95.         {
96.             ret += tp->cnt;
97.             tp->cnt = -1;

```

```

98.         tp = tp->fail;
99.     }
100. }
101.     return ret;
102. }
103.
104. /** 测试用.. */
105. char for_print[30];
106. void print(Node *cur, char *str, int idx)
107. {
108.     for(int i = 0; i < CHILDREN; i++)
109.     {
110.         if(cur->next[i])
111.         {
112.             str[idx] = (char)(i+'a');
113.             if(cur->next[i]->cnt == 1) {
114.                 str[idx+1] = '\0';
115.                 printf("%s\n", str);
116.                 cur->next[i]->cnt = 0;
117.             }
118.             print(cur->next[i], str, idx+1);
119.         }
120.     }
121. }
122.
123. };
124. char main_str[MAXN];
125. ACAutomaton ac;
126. int main()
127. {

```

```

128.     int t; scanf("%d", &t);
129.     while(t--)
130.     {
131.         int n;
132.         scanf("%d", &n);
133.         getchar();
134.
135.         ac.reset();
136.         for(int i = 0; i < n; i++)
137.         {
138.             char str[55];
139.             scanf("%s", str);
140.             ac.insert(str);
141.         }
142.         //ac.print(ac.root, ac.for_print, 0);
143.         ac.build();
144.         scanf("%s", main_str);
145.         printf("%d\n", ac.solve(main_str));
146.         ac.del(ac.root);
147.     }
148.     return 0;
149. }

```

2.6 字符串 Hash

```

1. unsigned int BKDRHash(char *str)
2. {
3.     unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
4.     unsigned int hash = 0;

```

```

5.
6.     while (*str)
7.     {
8.         hash = hash * seed + (*str++);
9.     }
10.
11.     return (hash & 0x7FFFFFFF) % MOD;
12. }

```

3 图论

3.1 SPFA

```

1. #include <cstdio>
2. #include <iostream>
3. #include <cstring>
4. #include <cmath>
5. #include <string>
6. #include <queue>
7. #include <map>
8. #include <vector>
9. #include <algorithm>
10. #define DEBUG 0
11. #define INF 0x1fffffff
12. #define MAXS 105
13.
14. typedef long long LL;
15. using namespace std;

```

```

16.
17. struct Edge {
18.     int v, w;
19.     Edge() {}
20.     Edge(int vv, int ww) {v = vv; w = ww; }
21. };
22. vector<Edge> ver[MAXS];
23. int n, m;
24. int inq[MAXS], times[MAXS], dis[MAXS], gra[MAXS][MAXS];
25.
26. bool creat_graph()
27. {
28.     scanf("%d%d", &n, &m);
29.     if(n == 0 && m == 0) return false;
30.     for(int i = 1; i <= n; i++) ver[i].clear();
31.     for(int i = 1; i <= m; i++)
32.     {
33.         int u, v, w;
34.         scanf("%d%d%d", &u, &v, &w);
35.         ver[u].push_back(Edge(v, w));
36.         ver[v].push_back(Edge(u, w));
37.     }
38.     return true;
39. }
40.
41. bool creat_graph_juzhen()
42. {
43.     scanf("%d%d", &n, &m);
44.     if(n == 0 && m == 0) return false;
45.     for(int i = 1; i <= n; i++)

```

```

46.         for(int j = 1; j <= n; j++)
47.             gra[i][j] = INF;
48.
49.     for(int i = 1; i <= m; i++)
50.     {
51.         int u, v, w;
52.         scanf("%d%d%d", &u, &v, &w);
53.         gra[u][v] = gra[v][u] = w;
54.     }
55.     return true;
56. }
57.
58.
59. bool spfa(int s)
60. {
61.     for(int i = 0; i <= n; i++)
62.     {
63.         dis[i] = INF;
64.         inq[i] = times[i] = 0;
65.     }
66.     queue<int> q;
67.     //while(!q.empty()) q.pop();
68.     q.push(s);
69.     dis[s] = 0;
70.     inq[s] = 1;
71.     while(!q.empty())
72.     {
73.         int cur = q.front(); q.pop();
74.         inq[cur] = 0;
75.         times[cur]++;

```

```

76.         if(times[cur] > n) return false;    // negative?
77.         ///** Vector graph.
78.         for(int i = 0; i != ver[cur].size(); i++)
79.         {
80.             int v = ver[cur][i].v, w = ver[cur][i].w;
81.             if( dis[v] > dis[cur] + w) {
82.                 dis[v] = dis[cur] + w;
83.                 if(!inq[v])
84.                 {
85.                     inq[v] = 1;
86.                     q.push(v);
87.                 }
88.             }
89.         }
90.         /** JuZhenal graph
91.         for(int i = 1; i <= n; i++)
92.         {
93.             if(dis[i] > dis[cur] + gra[cur][i])
94.                 dis[i] = dis[cur] + gra[cur][i];
95.             if(!inq[i]) {
96.                 inq[i] = 1;
97.                 q.push(i);
98.             }
99.         }*/
100.     }
101.     return true;
102. }
103.
104. int main()
105. {

```

```

106. while(creat_graph())
107. {
108.     spfa(1);
109.     printf("%d\n", dis[n]);
110. }
111. return 0;
112. }

```

3.2 Floyd(改进版：可求最小环权值)

```

1.  #include <cstdio>
2.  #include <algorithm>
3.  #define DEBUG 0
4.  #define INF 0x3f3f3f3f
5.  #define MAXN 1000
6.
7.  typedef long long LL;
8.  using namespace std;
9.
10. int n, m;
11. int pre[MAXN][MAXN], dis[MAXN][MAXN], gra[MAXN][MAXN];
12.
13. /** 返回值为最小环权值. */
14. int Floyd(int n) {
15.     int minCircle = INF;    /** 改进后的 Floyd 可求最小环。minCircle 用于记
    录最小环权值。 */
16.
17.     for(int k = 0; k < n; k++) {
18.         /** 改进部分 求最小环权值. */

```

```

19.         for(int i = 0; i < k; i++)
20.             for(int j = 0; j < i; j++)
21.                 minCircle = min(minCircle, dis[i][j] + gra[i][k] + gra[k][j]);
22.
23.         /** 通常部分。 */
24.         for(int i = 0; i < n; i++) {
25.             for(int j = 0; j < i; j++) {
26.                 int tmp = dis[i][k] + dis[k][j];
27.                 if(tmp < dis[i][j]) {
28.                     dis[i][j] = dis[j][i] = tmp;
29.                     pre[i][j] = pre[j][i] = k;
30.                 }
31.             }
32.         }
33.     }
34.     return minCircle;
35. }
36.
37. void init()
38. {
39.     for(int i = 0; i < n; i++) {
40.         for(int j = 0; j < n; j++) {
41.             dis[i][j] = INF;
42.             pre[i][j] = j;
43.         }
44.     }
45. }
46.
47. int main()
48. {

```

```

49. while(scanf("%d", &n))
50. {
51.     init();
52.     for(int i = 0; i < n; i++) {
53.         for(int j = 0; j < n; j++) {
54.             scanf("%d", &dis[i][j]);
55.             if(dis[i][j] == 0) dis[i][j] = INF;
56.         }
57.     }
58.     int x, y;
59.     Floyd(n);
60.     scanf("%d%d", &x, &y);
61.
62.     int swaped = 0;
63.     if(x < y) {
64.         swap(x, y);
65.         swaped = 1;
66.     }
67.
68.     int t, path[100], cnt = 1;
69.     path[0] = x;
70.     t = pre[x][y];
71.     while(t != y) {
72.         path[cnt++] = t;
73.         t = pre[t][y];
74.     } path[cnt++] = y;
75.     if(dis[x][y] == INF) {
76.         printf("NO PATH\n");
77.     }
78.     else if(!swaped) {

```

```

79.         printf("distance: %d\n", dis[y][x]);
80.         printf("Path:\n");
81.         for(int i = 0; i < cnt; i++)
82.             printf("%d ", path[i]);
83.     } else {
84.         printf("distance: %d\n", dis[y][x]);
85.         printf("Path:\n");
86.         for(int i = cnt - 1; i >= 0; i--)
87.             printf("%d ", path[i]);
88.     }
89. }
90. return 0;
91. }
92.
93. /**
94.  * 求当前最小环权值在更新 dis 之前是因为。。当前的 k 不应该被连入
    dis[i][j]中。
95.  * 也就是说当前迭代求 minCircle 是对之前已求 dis[i][j]包含 k-1 以前的点的
    最短路，
96.  * 不能包含 k! 如果包含 k (也就是说，如果求 minCircle 的代码在更新 dis
    后面的话，
97.  * dis[i][j]就已经包含了 k)，那么这次更新 minCircle 时
98.  * minCircle = min(minCircle, dis[i][j] + gra[i][k] + gra[k][j]); 就已经无效了
99.  * 因为 gra[i][k]和 gra[k][j]很可能已经被加入到这个最短路了。这样的话就无
    法构成环了。
100. */

```

3.3 Tarjan (无向图求割点)

```
1.  #include <cstdio>
2.  #include <iostream>
3.  #include <cstring>
4.  #include <cmath>
5.  #include <string>
6.  #include <queue>
7.  #include <map>
8.  #include <vector>
9.  #include <algorithm>
10. #define DEBUG 0
11. #define INF 0x3fffffff
12. #define MAXS 1005
13.
14. typedef long long LL;
15. using namespace std;
16.
17. int dfn[MAXS], low[MAXS], cut[MAXS];
18. int n, m, root;    /** n 点, m 边, root 根. */
19. vector<int> ver[MAXS];
20. int DFN;
21.
22.
23. /** 邻接表(ver)存储的图的 tarjan:   (无向图求割点)  */
24. void tarjan_cut(int u, int fa)
25. {
26.     int son = 0;
27.     dfn[u] = low[u] = ++DFN;
28.     for(int i = 0; i != ver[u].size(); i ++)
```

```
29.     {
30.         int v = ver[u][i];
31.         if(!dfn[v]) {
32.             tarjan_cut(v, u);
33.             son ++;
34.             low[u] = min(low[u], low[v]);
35.             if((u == root && son > 1) || (u != root && dfn[u] <= low[v]))
36.                 cut[u] = 1;
37.         } else if(dfn[u] > dfn[v] && v != fa) {
38.             low[u] = min(low[u], dfn[v]);
39.         }
40.     }
41. }
42.
43.
44. int bridge[MAXS][MAXS];
45. void tarjan_bridge(int u, int fa)
46. {
47.     int v;
48.     dfn[u] = low[u] = ++DFN;
49.     for(int i = 0; i != ver[u].size(); i ++){
50.         {
51.             v = ver[u][i];
52.             if(!dfn[v])
53.             {
54.                 tarjan_bridge(v, u);
55.                 low[u] = min(low[u], low[v]);
56.                 if(low[v] > dfn[u])
57.                 {
58.                     bridge[u][v] = bridge[v][u] = 1;
```



```

59.     }
60. }
61. else if(dfn[v] < dfn[u] && v != fa)
62. {
63.     low[u] = min(low[u], dfn[v]);
64. }
65. }
66. }
67.
68.
69. /** 初始化. */
70. void init()
71. {
72.     DFN = 0;
73.     memset(dfn, 0, sizeof(dfn));
74.     memset(low, 0, sizeof(low));
75.     memset(bridge, 0, sizeof(bridge));
76.     for(int i = 1; i <= n; i++)
77.     {
78.         ver[i].clear();
79.     }
80. }
81.
82. void creat_graph()
83. {
84.     scanf("%d%d", &n, &m);
85.     for(int i = 1; i <= m; i++)
86.     {
87.         int u, v;
88.         scanf("%d%d", &u, &v);

```

```

89.         ver[u].push_back(v);
90.         ver[v].push_back(u);
91.     }
92. }
93.
94. void print_cut()
95. {
96.     printf("CUT:\n");
97.     for(int i = 1; i <= n; i++)
98.     {
99.         if(cut[i])
100.             printf("%d ", i);
101.     }
102.     puts("");
103. }
104.
105. void print_bridge()
106. {
107.     printf("BRIDGE:\n");
108.     for(int i = 1; i <= n; i++)
109.     {
110.         for(int j = i + 1; j <= n; j++)
111.         {
112.             if(bridge[i][j])
113.                 printf("%d-%d, ", i, j);
114.         }
115.     }
116.     puts("");
117. }
118.

```

```

119. int main()
120. {
121.     init();
122.     creat_graph();
123.
124.     if(DEBUG)
125.     {
126.         for(int i = 1; i <= n; i++)
127.         {
128.             for(vector<int>::size_type j = 0; j != ver[i].size(); j++)
129.                 printf("%d ", ver[i][j]);
130.             printf("\n");
131.         }
132.     }
133.     root = 1;
134.     //tarjan_cut(root, -1);
135.     print_cut();
136.     tarjan_bridge(root, -1);
137.     print_bridge();
138.     return 0;
139. }
140.
141. /**
142. *      当探寻当前节点的下一个节点的时候，先判断是否已 vis 过，如果没
        有则继续 tarjan
143. *      如果 vis 过了。!!： 还需判断是否不是其父节点!! 再更新 low[u] =
        min(low[u], dfn[v])
144. */

```

3.4 BFS 判断二分图

```

1.  #include <queue>
2.  #include <cstring>
3.  #include <iostream>
4.  using namespace std;
5.
6.  const int N = 510;
7.  int col[N],g[N][N];
8.
9.  /** 0 为白色， 1 为黑色 */
10. bool bfs(int s, int n){
11.     queue<int> p;
12.     p.push(s);
13.     col[s] = 1;
14.     while(!p.empty()){
15.         int r = p.front();
16.         p.pop();
17.         for(int i = 1;i <= n;i++){
18.             if(g[r][i]&&col[i] == -1){
19.                 p.push(i);
20.                 col[i] = 1 - col[r]; /** 染成不同的颜色 */
21.             }
22.             if(g[r][i]&&col[r] == col[i]) /** 颜色有相同，则不是二分图 */
23.                 return false;
24.         }
25.     }
26.     return true;
27. }
28.

```

```

29. int main(){
30.     int n, m, a, b, i;
31.     memset(col, -1, sizeof(col));
32.     cin >> n >> m;
33.     for(i = 0; i < m; i++){
34.         cin >> a >> b;
35.         g[a][b] = g[b][a] = 1;
36.     }
37.     bool flag = false;
38.     for(i = 1; i <= n; i++)
39.         if(col[i] == -1 && !bfs(i, n)){//遍历各个连通分支
40.             flag = true;
41.             break;
42.         }
43.     if(flag)
44.         cout << "NO" << endl;
45.     else
46.         cout << "YES" << endl;
47.     return 0;
48. }

```

4 其他

4.1 大数类

```

1. #include <iostream>
2. #include <cstdio>

```

```

3. #include <cstdlib>
4. #include <cstring>
5. #include <cmath>
6. #include <string>
7. #include <queue>
8. #include <map>
9. #include <vector>
10. #include <algorithm>
11. #include <cstdlib>
12. #include <ctime>
13.
14. #define DEBUG 0
15. #define LSON(x) (x) << 1
16. #define RSON(x) (x) << 1 | 1
17. #define INF 0x1fffffff
18.
19. #define MAXN 9999
20. #define MAXSIZE 10
21. #define DLEN 4
22.
23. typedef long long LL;
24. using namespace std;
25.
26.
27. class BigNum
28. {
29. private:
30.     int a[500];    //可以控制大数的位数
31.     int len;        //大数长度
32. public:

```

```

33.     BigNum(){ len = 1;memset(a,0,sizeof(a)); }    //构造函数
34.     BigNum(const long long);           //将一个 int 类型的变量转化为大数
35.     BigNum(const char*);
36. //将一个字符串类型的变量转化为大数
37.     BigNum(const BigNum &);
38. //拷贝构造函数
39.     BigNum &operator=(const BigNum &);
40. //重载赋值运算符，大数之间进行赋值运算
41.
42.     friend istream& operator>>(istream&,  BigNum&);
43. //重载输入运算符
44.     friend ostream& operator<<(ostream&,  BigNum&);
45. //重载输出运算符
46.
47.     BigNum operator+(const BigNum &) const;
48. //重载加法运算符，两个大数之间的相加运算
49.     BigNum operator-(const BigNum &) const;
50. //重载减法运算符，两个大数之间的相减运算
51.     BigNum operator*(const BigNum &) const;
52. //重载乘法运算符，两个大数之间的相乘运算
53.     BigNum operator/(const int    &) const;
54. //重载除法运算符，大数对一个整数进行相除运算
55.
56.     BigNum operator^(const int    &) const;
57. //大数的 n 次方运算
58.     int    operator%(const int    &) const;
59. //大数对一个 int 类型的变量进行取模运算
60.     bool    operator>(const BigNum & T)const;
61. //大数和另一个大数的大小比较
62.     bool    operator>(const int & t)const;

```

```

63. //大数和一个 int 类型的变量的大小比较
64.
65.     void print();           //输出大数
66. };
67. BigNum::BigNum(const long long b)
68. //将一个 int 类型的变量转化为大数
69. {
70.     long long c,d = b;
71.     len = 0;
72.     memset(a,0,sizeof(a));
73.     while(d > MAXN)
74.     {
75.         c = d - (d / (MAXN + 1)) * (MAXN + 1);
76.         d = d / (MAXN + 1);
77.         a[len++] = c;
78.     }
79.     a[len++] = d;
80. }
81. BigNum::BigNum(const char*s)
82. //将一个字符串类型的变量转化为大数
83. {
84.     int t,k,index,l,i;
85.     memset(a,0,sizeof(a));
86.     l=strlen(s);
87.     len=l/DLEN;
88.     if(l%DLEN)
89.         len++;
90.     index=0;
91.     for(i=l-1;i>=0;i-=DLEN)
92.     {

```

```

93.         t=0;
94.         k=i-DLEN+1;
95.         if(k<0)
96.             k=0;
97.         for(int j=k;j<=i;j++)
98.             t=t*10+s[j]-'0';
99.         a[index++]=t;
100.     }
101. }
102. BigNum::BigNum(const BigNum & T) : len(T.len) //拷贝构造函数
103. {
104.     int i;
105.     memset(a,0,sizeof(a));
106.     for(i = 0 ; i < len ; i++)
107.         a[i] = T.a[i];
108. }
109. BigNum & BigNum::operator=(const BigNum & n)
110. //重载赋值运算符，大数之间进行赋值运算
111. {
112.     int i;
113.     len = n.len;
114.     memset(a,0,sizeof(a));
115.     for(i = 0 ; i < len ; i++)
116.         a[i] = n.a[i];
117.     return *this;
118. }
119. istream& operator>>(istream & in, BigNum & b)
120. //重载输入运算符
121. {
122.     char ch[MAXSIZE*4];

```

```

123.     int i = -1;
124.     in>>ch;
125.     int l=strlen(ch);
126.     int count=0,sum=0;
127.     for(i=l-1;i>=0;)
128.     {
129.         sum = 0;
130.         int t=1;
131.         for(int j=0;j<4&& i>=0;j++,i--,t*=10)
132.         {
133.             sum+=(ch[i]-'0')*t;
134.         }
135.         b.a[count]=sum;
136.         count++;
137.     }
138.     b.len =count++;
139.     return in;
140.
141. }
142. ostream& operator<<(ostream& out, BigNum& b)
143. //重载输出运算符
144. {
145.     int i;
146.     cout << b.a[b.len - 1];
147.     for(i = b.len - 2 ; i >= 0 ; i--)
148.     {
149.         cout.width(DLEN);
150.         cout.fill('0');
151.         cout << b.a[i];
152.     }

```

```

153.     return out;
154. }
155.
156. BigNum BigNum::operator+(const BigNum & T) const
157. //两个大数之间的相加运算
158. {
159.     BigNum t(*this);
160.     int i,big;        //位数
161.     big = T.len > len ? T.len : len;
162.     for(i = 0 ; i < big ; i++)
163.     {
164.         t.a[i] +=T.a[i];
165.         if(t.a[i] > MAXN)
166.         {
167.             t.a[i + 1]++;
168.             t.a[i] -=MAXN+1;
169.         }
170.     }
171.     if(t.a[big] != 0)
172.         t.len = big + 1;
173.     else
174.         t.len = big;
175.     return t;
176. }
177. BigNum BigNum::operator-(const BigNum & T) const
178. //两个大数之间的相减运算
179. {
180.     int i,j,big;
181.     bool flag;
182.     BigNum t1,t2;

```

```

183.     if(*this>T)
184.     {
185.         t1=*this;
186.         t2=T;
187.         flag=0;
188.     }
189.     else
190.     {
191.         t1=T;
192.         t2=*this;
193.         flag=1;
194.     }
195.     big=t1.len;
196.     for(i = 0 ; i < big ; i++)
197.     {
198.         if(t1.a[i] < t2.a[i])
199.         {
200.             j = i + 1;
201.             while(t1.a[j] == 0)
202.                 j++;
203.             t1.a[j--]--;
204.             while(j > i)
205.                 t1.a[j--] += MAXN;
206.             t1.a[i] += MAXN + 1 - t2.a[i];
207.         }
208.         else
209.             t1.a[i] -= t2.a[i];
210.     }
211.     t1.len = big;
212.     while(t1.a[len - 1] == 0 && t1.len > 1)

```

```

213. {
214.     t1.len--;
215.     big--;
216. }
217. if(flag)
218.     t1.a[big-1]=0-t1.a[big-1];
219. return t1;
220. }
221.
222. BigNum BigNum::operator*(const BigNum & T) const
223. //两个大数之间的相乘运算
224. {
225.     BigNum ret;
226.     int i,j,up;
227.     int temp,temp1;
228.     for(i = 0 ; i < len ; i++)
229.     {
230.         up = 0;
231.         for(j = 0 ; j < T.len ; j++)
232.         {
233.             temp = a[i] * T.a[j] + ret.a[i + j] + up;
234.             if(temp > MAXN)
235.             {
236.                 temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
237.                 up = temp / (MAXN + 1);
238.                 ret.a[i + j] = temp1;
239.             }
240.             else
241.             {
242.                 up = 0;

```

```

243.                 ret.a[i + j] = temp;
244.             }
245.         }
246.         if(up != 0)
247.             ret.a[i + j] = up;
248.     }
249.     ret.len = i + j;
250.     while(ret.a[ret.len - 1] == 0 && ret.len > 1)
251.         ret.len--;
252.     return ret;
253. }
254. BigNum BigNum::operator/(const int & b) const
255. //大数对一个整数进行相除运算
256. {
257.     BigNum ret;
258.     int i,down = 0;
259.     for(i = len - 1 ; i >= 0 ; i--)
260.     {
261.         ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
262.         down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
263.     }
264.     ret.len = len;
265.     while(ret.a[ret.len - 1] == 0 && ret.len > 1)
266.         ret.len--;
267.     return ret;
268. }
269. int BigNum::operator%(const int & b) const
270. //大数对一个 int 类型的变量进行取模运算
271. {
272.     int i,d=0;

```

```

273.     for (i = len-1; i>=0; i--)
274.     {
275.         d = ((d * (MAXN+1))% b + a[i])% b;
276.     }
277.     return d;
278. }
279. BigNum BigNum::operator^(const int & n) const
280. //大数的 n 次方运算
281. {
282.     BigNum t,ret(1);
283.     int i;
284.     if(n<0)
285.         exit(-1);
286.     if(n==0)
287.         return 1;
288.     if(n==1)
289.         return *this;
290.     int m=n;
291.     while(m>1)
292.     {
293.         t=*this;
294.         for( i=1;i<<1<=m;i<<=1)
295.         {
296.             t=t*t;
297.         }
298.         m-=i;
299.         ret=ret*t;
300.         if(m==1)
301.             ret=ret*(t);
302.     }

```

```

303.     return ret;
304. }
305. bool BigNum::operator>(const BigNum & T) const
306. //大数和另一个大数的大小比较
307. {
308.     int ln;
309.     if(len > T.len)
310.         return true;
311.     else if(len == T.len)
312.     {
313.         ln = len - 1;
314.         while(a[ln] == T.a[ln] && ln >= 0)
315.             ln--;
316.         if(ln >= 0 && a[ln] > T.a[ln])
317.             return true;
318.         else
319.             return false;
320.     }
321.     else
322.         return false;
323. }
324. bool BigNum::operator >(const int & t) const
325. //大数和一个 int 类型的变量的大小比较
326. {
327.     BigNum b(t);
328.     return *this>b;
329. }
330.
331. void BigNum::print()    //输出大数
332. {

```



```

333.     int i;
334.     printf("%d", a[len-1]);
335.     //cout << a[len - 1];
336.     for(i = len - 2 ; i >= 0 ; i--)
337.     {
338.         printf("%00004d", a[i]);
339.         //      cout.width(DLEN);
340.         //      cout.fill('0');
341.         //      cout << a[i];
342.     }
343.     //cout << endl;
344.     printf("\n");
345. }
346.
347. int main(void)
348. {
349.     //      freopen("out.txt", "r", stdin);
350.     //      freopen("out1.txt", "w", stdout);
351.     int t; scanf("%d", &t);
352.     for(int c = 1; c <= t; c++)
353.     {
354.         long long n, lim = 1000000000;
355.         scanf("%l64d", &n);
356.         if(n == 0)
357.         {
358.             printf("Case #d: 1\n", c);
359.             continue;
360.         }
361.         if(n < lim)
362.         {

```

```

363.             printf("Case #d: %l64d\n", c, 8 * n * n - 7*n + 1);
364.             continue;
365.         }
366.         BigNum tp(n);
367.         BigNum ans, six(8), mins(n*7-1);
368.         ans = tp * tp;
369.         ans = ans * six;
370.         ans = ans - mins;
371.         printf("Case #d: ", c);
372.         ans.print();
373.     }
374.     return 0;
375. }

```

4.2 输入挂

```

1.
2. template <class T>
3. inline bool scan_d(T &ret) {
4.     char c; int sgn;
5.     if(c=getchar(),c==EOF) return 0; //EOF
6.     while(c!='-'&&(c<'0' || c>'9')) c=getchar();
7.     sgn=(c=='-')?-1:1;
8.     ret=(c=='-')?0:(c-'0');
9.     while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
10.    ret*=sgn;
11.    return 1;
12. }
13.

```