

SAJIHPTS Code Snippets

SAJIHPTS Team

October 13, 2016

Contents

| | | |
|----------|-------------------------------------|-----------|
| 1 | Cal Geo | 3 |
| 1.1 | jisuan | 3 |
| 2 | Data Structures | 20 |
| 2.1 | Cartesian Tree | 20 |
| 2.2 | mergeable heap | 22 |
| 2.3 | Segment Tree | 24 |
| 2.4 | Splay | 27 |
| 2.5 | Treap | 32 |
| 2.6 | Tree Cut(edge) | 35 |
| 2.7 | Tree Cut(vertex) | 38 |
| 3 | Graph | 42 |
| 3.1 | (Euler)Fluery | 42 |
| 3.2 | 0-1 分数规划 | 44 |
| 3.3 | Biggest Tuan | 45 |
| 3.4 | dijkstra(nlogn) | 46 |
| 3.5 | Directed-MST | 47 |
| 3.6 | Hamilton | 49 |
| 3.7 | K-shortest path | 54 |
| 3.8 | Second-MST | 56 |
| 3.9 | Stable Wedding | 58 |
| 4 | Bipartite Graph Match | 60 |
| 4.1 | Bipartite Graph Match-BFS | 60 |
| 4.2 | Bipartite Graph Match-DFS | 62 |
| 5 | Flow | 64 |
| 5.1 | dinic | 64 |
| 5.2 | isap | 66 |
| 6 | LCA | 69 |
| 6.1 | Double | 69 |
| 6.2 | ST | 72 |
| 6.3 | Tarjan | 74 |
| 7 | Tarjan | 76 |
| 7.1 | bridge cut-vertex | 76 |
| 7.2 | edge-double connected | 77 |
| 7.3 | SCC | 78 |
| 7.4 | vertex-double connected | 79 |

| | | |
|-----------|---------------------------------|------------|
| 8 | Math | 81 |
| 8.1 | FFT | 81 |
| 8.2 | Integer High Accuracy | 84 |
| 8.3 | math | 96 |
| 8.4 | Matrix Fast Mi | 101 |
| 8.5 | NTT CRT | 103 |
| 9 | Prime | 106 |
| 9.1 | Euler Prime | 106 |
| 9.2 | Miller-Rabin Pollard | 107 |
| 10 | String | 110 |
| 10.1 | AC Automation | 110 |
| 10.2 | AC-Auto(Compressed) | 112 |
| 10.3 | KMP | 115 |
| 10.4 | LCS | 116 |
| 10.5 | manacher | 117 |
| 10.6 | palindrome automation | 118 |
| 10.7 | SAM(Compressed) | 120 |
| 10.8 | SAM | 122 |
| 10.9 | Suffix Tree | 124 |
| 10.10 | Trie(Compressed) | 125 |
| 10.11 | Trie | 127 |
| 11 | Others | 129 |
| 11.1 | Attention | 129 |
| 11.2 | operations | 130 |
| 11.3 | Read | 131 |
| 11.4 | Tags | 132 |
| 11.5 | vimrc | 133 |
| 11.6 | Wrong | 134 |

1 Cal Geo

1.1 jisuan

```
1  #include<iostream>
2  #include<cmath>
3  #include<algorithm>
4  #include<vector>
5  using namespace std;
6  #define EPS 1e-10
7  #define INF 1e10
8  #define PI 3.14159265358979323846
9  inline bool EQUAL(double t1, double t2){
10     return t1 - t2 < EPS && t1 - t2 > -EPS;
11 }
12 inline bool LESS(double t1, double t2){
13     return t1 <= t2 - EPS;
14 }
15 inline bool LESS_EQUAL(double t1, double t2){
16     return t1 < t2 + EPS;
17 }
18 inline int SGN(double t){
19     return LESS(t, 0) ? -1 : LESS(0, t) ? 1 : 0;
20 }
21 class Point
22 {
23 public:
24     double x, y;
25     Point(){}
26     Point(double x, double y) :x(x), y(y){}
27
28     bool operator == (const Point& p)const{
29         return EQUAL(x, p.x) && EQUAL(y, p.y);
30     }
31     bool operator < (const Point& p)const{
32         return LESS_EQUAL(x, p.x) && (LESS(x, p.x) || LESS(y, p.y));
33     }
34     Point operator + (const Point& p)const{
35         return Point(x + p.x, y + p.y);
36     }
37     Point operator - (const Point& p)const{
38         return Point(x - p.x, y - p.y);
39     }
40     double operator * (const Point& p)const{
41         return x*p.y - y*p.x;
```

```

42     }
43     Point operator * (double value) const {
44         return Point(x*value, y*value);
45     }
46     Point operator / (double value) const {
47         return Point(x / value, y / value);
48     }
49     double dot(const Point& p) const {
50         return x*p.x + y*p.y;
51     }
52     double r2() const { return x*x + y*y; }
53     double r() const { return hypot(x, y); }
54     double dis2(const Point& p) const {
55         return (*this - p).r2();
56     }
57     double dis(const Point& p) const {
58         return (*this - p).r();
59     }
60
61     bool onLine(const Point& p1, const Point& p2) const {
62         return EQUAL((*this - p1)*(*this - p2), 0);
63     }
64     bool onLineSeg(const Point& p1, const Point& p2) const {
65         //include extream points
66         return onLine(p1, p2) && inRect(p1, p2);
67     }
68     double lineRelation(const Point& p1, const Point& p2) const {
69         Point t = p2 - p1;
70         return t.dot(*this - p1) / t.r2();
71         //ret 0, *this=p1; ret 1,*this=p2;
72         //ret (0,1), *this is interior to p1p2
73     }
74     Point footPoint(const Point& p1, const Point& p2) const {
75         double r = lineRelation(p1, p2);
76         return p1 + (p2 - p1)*r;
77     }
78     double lineDis(const Point& p1, const Point& p2) const {
79         return abs((p1 - *this)*(p2 - *this)) / p1.dis(p2);
80     }
81     double lineSegDis(const Point& p1, const Point& p2, Point&
82         ↪ ret) const;
83     double lineSegArrayDis(const Point* p, int lineNum, Point&
84         ↪ ret) const;
85     bool lineSegArrayDisCmp(const Point* p, int lineNum, double
86         ↪ value) const;
87     Point mirror(Point& p1, Point& p2) {

```

```

85     Point foot = footPoint(p1, p2);
86     return foot * 2 - *this;
87 }
88
89 Point rotate(double angle) const {
90     Point f(sin(angle), cos(angle));
91     return Point(*this * f, dot(f));
92 }
93 Point rotate90() const {
94     return Point(-y, x);
95 }
96 double cosAngle(const Point& p1, const Point& p2) const {
97     Point t1 = *this - p1, t2 = *this - p2;
98     return t1.dot(t2) / sqrt(t1.r2()*t2.r2());
99 }
100 double tanAngle(const Point& o = Point(0, 0)) const {
101     if (EQUAL(x, o.x)) return y - o.y >= 0 ? INF : -INF;
102     return (y - o.y) / (x - o.x);
103 }
104 double angle(const Point& p1, const Point& p2) const {
105     return acos(cosAngle(p1, p2));
106 }
107 double angle(const Point& o = Point(0, 0)) const {
108     return atan2(y - o.y, x - o.x);
109 }
110 //left return 1, right return -1, on line return 0.
111 int direction(const Point& p1, const Point& p2) const {
112     return SGN(x*(p1.y - p2.y) + p1.x*(p2.y - y) + p2.x*(y -
        ↪ p1.y));
113 }
114
115 bool inRect(const Point& p1, const Point& p2) const {
116     return LESS_EQUAL((p1.x - x)*(p2.x - x), 0) &&
        ↪ LESS_EQUAL((p1.y - y)*(p2.y - y), 0);
117 }
118 int inPolygon(const Point* p, int n) const;
119 int inConvex(const Point* p, int n) const;
120 int inCircle(const Point& o, double r) const {
121     double dist = dis2(o);
122     return SGN(r*r - dist);
123 }
124 void pointcut(const Point& o, double r, Point& ret1, Point&
    ↪ ret2) const;
125 Point nearestPoint(const Point& o, double r) const;
126 };

```

```

127 double Point::lineSegDis(const Point& p1, const Point& p2, Point&
    ↪ ret)const
128 {
129     double r = lineRelation(p1, p2);
130     if (LESS_EQUAL(r, 0))ret = p1;
131     else if (LESS_EQUAL(1, r))ret = p2;
132     else ret = footPoint(p1, p2);
133     return dis(ret);
134 }
135 //input lineNum+1 points
136 double Point::lineSegArrayDis(const Point* p, int lineNum, Point&
    ↪ ret)const
137 {
138     Point tp;
139     double td, mind = INF;
140     for (int i = 0; i < lineNum; i++){
141         td = lineSegDis(p[i], p[i + 1], tp);
142         if (LESS(td, mind)){
143             mind = td; ret = tp;
144         }
145     }
146     return mind;
147 }
148 //input lineNum+1 points
149 bool Point::lineSegArrayDisCmp(const Point* p, int lineNum, double
    ↪ value)const
150 {
151     Point tp;
152     double td;
153     int flag = 1;
154     for (int i = 0; i < lineNum; i++){
155         td = lineSegDis(p[i], p[i + 1], tp);
156         if (LESS_EQUAL(td, value))
157             return true;
158     }
159     return false;
160 }
161
162 //donnot include extream points, and donnot include coincidence.
163 inline bool lineSegLineSegIntersect(const Point& p1, const Point&
    ↪ p2, const Point& q1, const Point& q2){
164     Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;
165     return SGN(pq1*q12)*SGN((p2 - q1)*q12) < 0 &&
        ↪ SGN(pq1*p12)*SGN((p1 - q2)*p12) < 0;
166 }
167 //include extream points and coincidence.

```

```

168 inline bool lineSegLineSegIntersect2(const Point& p1, const Point&
    ↪ p2, const Point& q1, const Point& q2){
169     if (!(LESS_EQUAL(min(q1.x, q2.x), max(p1.x, p2.x)) &&
    ↪ LESS_EQUAL(min(p1.x, p2.x), max(q1.x, q2.x))
170         && LESS_EQUAL(min(q1.y, q2.y), max(p1.y, p2.y)) &&
    ↪ LESS_EQUAL(min(p1.y, p2.y), max(q1.y, q2.y))))
171         return false;
172     Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;
173     return SGN(pq1*q12)*SGN((p2 - q1)*q12) <= 0 &&
    ↪ SGN(pq1*p12)*SGN((p1 - q2)*p12) <= 0;
174 }
175 //donot include extream points, and donot include coincidence.
176 inline bool lineLineSegIntersect(const Point& l1, const Point& l2,
    ↪ const Point& p1, const Point& p2){
177     Point line = l2 - l1;
178     return SGN((p1 - l1)*line)*SGN((p2 - l1)*line) < 0;
179 }
180 //donnot include coincidence.
181 inline bool lineLineIntersect(const Point& p1, const Point& p2,
    ↪ const Point& q1, const Point& q2){
182     return !EQUAL((p2 - p1)*(q2 - q1), 0);
183 }
184 inline Point lineLineIntersectPoint(const Point& p1, const Point&
    ↪ p2, const Point& q1, const Point& q2){
185     Point q12 = q2 - q1;
186     double k = (p2 - p1)*q12;
187     if (EQUAL(k, 0)) return Point(INF*INF, INF*INF);
188     double r = ((q1 - p1)*q12) / k;
189     return p1 + (p2 - p1) * r;
190 }
191
192 Point circumcenter(const Point& p1, const Point& p2, const Point&
    ↪ p3)
193 {
194     Point t1 = (p1 + p2)*0.5, t2, t3 = (p2 + p3)*0.5, t4;
195     t2 = t1 + (p1 - p2).rotate90();
196     t4 = t3 + (p2 - p3).rotate90();
197     return lineLineIntersectPoint(t1, t2, t3, t4);
198 }
199 Point incenter(const Point& p1, const Point& p2, const Point& p3)
200 {
201     double r12 = p1.dis(p2), r23 = p2.dis(p3), r31 = p3.dis(p1);
202     Point t1 = (p2*r31 + p3*r12) / (r12 + r31), t2 = (p1*r23 +
    ↪ p3*r12) / (r12 + r23);
203     return lineLineIntersectPoint(p1, t1, p2, t2);
204 }

```



```

205 Point prepercenter(const Point& p1, const Point& p2, const Point&
    ↪ p3)
206 {
207     Point t1 = p1 + (p2 - p3).rotate90();
208     Point t2 = p2 + (p1 - p3).rotate90();
209     return lineLineIntersectPoint(p1, t1, p2, t2);
210 }
211 inline Point barycenter(const Point& p1, const Point& p2, const
    ↪ Point& p3){
212     return (p1 + p2 + p3) / 3;
213 }
214 inline double apothem(const Point& p1, const Point& p2, const
    ↪ Point& p3){
215     Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;
216     return abs(p12*p23) / (p12.r() + p13.r() + p23.r());
217 }
218 inline double circumradius(const Point& p1, const Point& p2, const
    ↪ Point& p3){
219     Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;
220     return sqrt(p12.r2()*p23.r2()*p13.r2()) / (2 * abs(p12*p23));
221 }
222
223 int getPolygonDirection(const Point* p, int n)
224 {
225     int index = 0;
226     for (int i = 1; i < n; i++){
227         if (p[i] < p[index])index = i;
228     }
229     return p[index].direction(p[index + 1 < n ? index + 1 : 0],
    ↪ p[index - 1 >= 0 ? index - 1 : n - 1]);
230 }
231 bool checkConvex(const Point* p, int n)
232 {
233     int direction = p[0].direction(p[n - 1], p[1]);
234     if (direction == 0)return false;
235     if (p[n - 1].direction(p[n - 2], p[0]) != direction)return
    ↪ false;
236     for (int i = n - 2; i > 0; i--){
237         if (p[i].direction(p[i - 1], p[i + 1]) != direction)
238             return false;
239     }
240     return true;
241 }
242 bool checkConvex(const Point* p, int n, bool *ret)
243 {
244     bool retValue = true;

```

```

245     int direction = getPolygonDirection(p, n);
246     if (!(ret[n - 1] = p[n - 1].direction(p[0], p[n - 2]) ==
        ↪ direction))
247         retValue = false;
248     if (!(ret[0] = p[0].direction(p[1], p[n - 1]) == direction))
249         retValue = false;
250     for (int i = n - 2; i > 0; i--){
251         if (!(ret[i] = p[i].direction(p[i + 1], p[i - 1]) ==
        ↪ direction))
252             retValue = false;
253     }
254     return retValue;
255 }
256 double polygonArea(const Point* p, int n)
257 {
258     double area = 0;
259     for (int i = n - 2; i > 0; i--)
260         area += p[i].y * (p[i - 1].x - p[i + 1].x);
261     area += p[0].y * (p[n - 1].x - p[1].x);
262     area += p[n - 1].y * (p[n - 2].x - p[0].x);
263     return area / 2;
264 }
265 int Point::inPolygon(const Point* p, int n) const
266 {
267     int i, j = n - 1, odd = -1;
268     for (i = 0; i < n; j = i++){
269         if (LESS(p[i].y, y) != LESS(p[j].y, y)){
270             double tx = (y - p[j].y) / (p[i].y - p[j].y) * (p[i].x -
        ↪ p[j].x) + p[j].x;
271             if (LESS_EQUAL(tx, x)){
272                 if (LESS(tx, x)) odd = -odd;
273                 else return 0;
274             }
275         }
276         else if (onLineSeg(p[i], p[j])) return 0;
277     }
278     return odd;
279 }
280 int Point::inConvex(const Point* p, int n) const
281 {
282     int _direction = p[1].direction(p[2], p[0]);
283     if (direction(p[0], p[1]) != _direction){
284         if (onLineSeg(p[0], p[1])) return 0;
285         return -1;
286     }
287     if (direction(p[n - 1], p[0]) != _direction){

```

```

288     if (onLineSeg(p[n - 1], p[0]))return 0;
289     return -1;
290 }
291 int left = 2, right = n - 1;
292 while (left < right){
293     int mid = (left + right) >> 1;
294     if (direction(p[0], p[mid]) == _direction)left = mid + 1;
295     else right = mid;
296 }
297 int ret = direction(p[left-1],p[left]);
298 return ret == _direction ? 1 : ret == 0 ? 0 : -1;
299 }
300 Point lineConvexIntersectPointInternal(const Point& p1, const
    ↪ Point& p2, const Point* p, int n, int start, int end)
301 {
302     Point p12 = p2 - p1;
303     if (end < start)end += n;
304     double value = SGN((p[start] - p1)*p12);
305     while (start + 1 < end){
306         int mid = (start + end) / 2;
307         Point cur = p[mid < n ? mid : mid - n];
308         double t = (cur - p1)*p12*value;
309         if (LESS(0, t))start = mid;
310         else if (LESS(t, 0))end = mid;
311         else return cur;
312     }
313     if (start >= n)start -= n;
314     return lineLineIntersectPoint(p1, p2, p[start], p[start + 1]);
315 }
316 int lineConvexIntersectPoint(const Point& p1, const Point& p2,
    ↪ const Point* p, int n, Point& ret1, Point& ret2)
317 {
318     Point p12 = p2 - p1;
319     int pos = 0, step = n * 2 / 3;
320     double d = (p[pos] - p1)*p12;
321     int zero = -1, pos2 = -1;
322     while (step > 1){
323         step=(step + 1) / 2;
324         int i = pos + step, k = pos - step;
325         if (i >= n)i -= n;
326         if (k < 0)k += n;
327         double di = (p[i] - p1)*p12, dk = (p[k] - p1)*p12;
328         if (SGN(di)*SGN(d) < 0){ pos2 = i; break; }
329         if (SGN(dk)*SGN(d) < 0){ pos2 = k; break; }
330         if (abs(di) < abs(d)){ d = di; pos = i; }
331         if (abs(dk) < abs(d)){ d = dk; pos = k; }

```

```

332     if (EQUAL(d, 0)){ zero = pos; break; }
333 }
334 if (zero != -1){
335     ret1 = p[zero];
336     int left = zero - 1 >= 0 ? zero - 1 : n - 1;
337     int right = zero + 1 < n ? zero + 1 : 0;
338     double dl = (p[left] - p1)*p12, dr = (p[right] - p1)*p12;
339     if (EQUAL(dl, 0)){ ret2 = p[left]; return 3; }
340     else if (EQUAL(dr, 0)){ ret2 = p[right]; return 3; }
341     else if (dl*dr < 0)return 1;
342     else{ pos = left; pos2 = right; }
343 }
344 if (pos2 == -1)return 0;
345 ret1 = lineConvexIntersectPointInternal(p1, p2, p, n, pos,
    ↪ pos2);
346 ret2 = lineConvexIntersectPointInternal(p1, p2, p, n, pos2,
    ↪ pos);
347 return 2;
348 }
349
350 bool lineSegInPolygon(const Point& p1, const Point& p2, const
    ↪ Point* p, int n)
351 {
352     bool flag = false;
353     Point minPoint;
354     switch (p1.inPolygon(p, n)){
355     case -1:return false;
356     case 0:flag = true;
357     }
358     switch (p2.inPolygon(p, n)){
359     case -1:return false;
360     case 1:flag = false;
361     }
362     if (flag)minPoint = max(p1, p2);
363     for (int i = 0, j = n - 1; i < n; j = i++){
364         if (p[i].onLineSeg(p1, p2) && !(p[i] == p1 || p[i] == p2)){
365             if (p[i > 0 ? i - 1 : n - 1].direction(p1, p2) * p[i + 1 < n
    ↪ ? i + 1 : 0].direction(p1, p2) < 0)
366                 return false;
367             if (flag && p[i] < minPoint)minPoint = p[i];
368         }
369         else if (lineSegLineSegIntersect(p[i], p[j], p1, p2))
370             return false;
371     }
372     if (flag){
373         const Point& t = min(p1, p2);

```

```

374     Point mid = (t + minPoint)*0.5;
375     if (mid.inPolygon(p, n) == -1) return false;
376 }
377 return true;
378 }
379 Point gravityCenter(const Point* p, int n)
380 {
381     if (n < 3){
382         if (n == 1) return p[0];
383         else return (p[0] + p[1])*0.5;
384     }
385     double area = 0;
386     Point ret(0, 0);
387     for (int i = 0, j = n - 1; i < n; j = i++){
388         double t = p[i] * p[j];
389         area += t;
390         ret.x += (p[i].x + p[j].x)*t;
391         ret.y += (p[i].y + p[j].y)*t;
392     }
393     return ret / (3 * area);
394 }
395 //ret[n] must be available to visit.
396 int convexHullSorted(const Point* p, int n, Point* ret)
397 {
398     int j = 0;
399     for (int i = 0; i < n; i++){
400         while (j >= 2 && p[i].direction(ret[j - 2], ret[j - 1]) !=
401             ↪ 1) j--;
402         ret[j++] = p[i];
403     }
404     int mid = j + 1;
405     for (int i = n - 2; i >= 0; i--){
406         while (j >= mid && p[i].direction(ret[j - 2], ret[j - 1]) !=
407             ↪ 1) j--;
408         ret[j++] = p[i];
409     }
410     return j - 1;
411 }
412 void convexHullSorted(const Point* p, int n, Point* up, int&
413     ↪ retUp, Point* down, int& retDown)
414 {
415     retUp = retDown = 0;
416     for (int i = 0; i < n; i++){
417         while (retUp >= 2 && p[i].direction(up[retUp - 2], up[retUp -
418             ↪ 1]) != -1) retUp--;

```

```

415     while (retDown >= 2 && p[i].direction(down[retDown - 2],
        ↪ down[retDown - 1]) != 1) retDown--;
416     up[retUp++] = p[i];
417     down[retDown++] = p[i];
418 }
419 }
420 int halfPlainIntersectInternal(vector<pair<double, const Point*>>&
    ↪ v, int n, Point* ret)
421 {
422     for (int i = 0; i < n; i++)
423         v[i].first = v[i].second[1].angle(v[i].second[0]);
424     sort(v.begin(), v.end());
425     vector<const Point*> line(n);
426     vector<Point> point(n);
427     int first = 0, last = 0;
428     line[0] = v[0].second;
429     for (unsigned int i = 1; i < v.size(); i++){
430         while (first < last && point[last -
            ↪ 1].direction(v[i].second[0], v[i].second[1]) == -1)
            ↪ last--;
431         while (first < last && point[first].direction(v[i].second[0],
            ↪ v[i].second[1]) == -1) first++;
432         line[++last] = v[i].second;
433         if (!lineLineIntersect(line[last - 1][0], line[last - 1][1],
            ↪ line[last][0], line[last][1])){
434             last--;
435             if (v[i].second[0].direction(line[last][0], line[last][1])
                ↪ == 1) line[last] = v[i].second;
436         }
437         if (first < last)
438             point[last - 1] = lineLineIntersectPoint(line[last - 1][0],
                ↪ line[last - 1][1], line[last][0], line[last][1]);
439     }
440     while (first < last && point[last - 1].direction(line[first][0],
        ↪ line[first][1]) == -1) last--;
441     if (last - first <= 1) return 0;
442     point[last] = lineLineIntersectPoint(line[first][0],
        ↪ line[first][1], line[last][0], line[last][1]);
443     int num = unique(&*point.begin() + first, &*point.begin() + last
        ↪ + 1) - &*point[first];
444     while (num > 1 && point[first] == point[first + num - 1]) num--;
445     memcpy(ret, &point[first], sizeof(Point)*num);
446     return num;
447 }
448 int halfPlainIntersect(const Point(*p)[2], int n, Point* ret)
449 {

```

```

450     vector<pair<double, const Point*>> v(n + 4);
451     Point ext[4][2] = { { { -INF, -INF }, { INF, -INF } }, { { INF,
        ↪ -INF }, { INF, INF } },
452     { { INF, INF }, { -INF, INF } }, { { -INF, INF }, { -INF, -INF }
        ↪ } };
453     for (int i = 0; i < 4; i++)
454         v[i].second = ext[i];
455     for (int i = 0; i < n; i++)
456         v[i + 4].second = p[i];
457     return halfPlainIntersectInternal(v, n + 4, ret);
458 }
459 int polygonKernel(const Point* p, int n, Point* ret)
460 {
461     vector<pair<double, const Point*>> v;
462     Point ext[2] = { p[n - 1], p[0] };
463     v[0].second = ext;
464     for (int i = 1; i < n; i++)
465         v[i].second = &p[i - 1];
466     return halfPlainIntersectInternal(v, n, ret);
467 }
468
469 struct NearestPointsStruct{
470     Point p1, p2;
471     double d2;
472     vector<Point> v;
473 };
474 inline bool nearestPointsCmp(const Point& p1, const Point& p2){
475     return LESS_EQUAL(p1.y, p2.y) && (LESS(p1.y, p2.y) || LESS(p1.x,
        ↪ p2.x));
476 }
477 void nearestPointsInternal(const Point* p, int left, int right,
    ↪ NearestPointsStruct& s)
478 {
479     if (right - left < 8){
480         for (int i = left; i < right; i++){
481             for (int j = i + 1; i < right; j++){
482                 double td2 = p[j].dis2(p[i]);
483                 if (td2 < s.d2){
484                     s.d2 = td2;
485                     s.p1 = p[i]; s.p2 = p[j];
486                 }
487             }
488         }
489         return;
490     }
491     int mid = (left + right) >> 1;

```

```

492 nearestPointsInternal(p, left, mid, s);
493 nearestPointsInternal(p, mid, right, s);
494 s.v.clear();
495 double l = (p[mid - 1].x + p[mid].x) / 2;
496 for (int i = mid - 1; i >= left && (p[i].x - l)*(p[i].x - l) <
    ↪ s.d2; i++)
497     s.v.push_back(p[i]);
498 for (int i = mid; i < right && (p[i].x - l)*(p[i].x - l) < s.d2;
    ↪ i++)
499     s.v.push_back(p[i]);
500 sort(s.v.begin(), s.v.end(), nearestPointsCmp);
501 for (unsigned int i = 0; i < s.v.size(); i++){
502     for (unsigned int j = i + 1; j < s.v.size() && (p[j].y -
    ↪ p[i].y)*(p[j].y - p[i].y) < s.d2; j++){
503         double td2 = p[j].dis2(p[i]);
504         if (td2 < s.d2){
505             s.d2 = td2;
506             s.p1 = p[i]; s.p2 = p[j];
507         }
508     }
509 }
510 }
511 double nearestPointsSorted(const Point* p, int n, Point& ret1,
    ↪ Point& ret2)
512 {
513     NearestPointsStruct s;
514     s.d2 = INF;
515     s.v.reserve(n);
516     nearestPointsInternal(p, 0, n, s);
517     ret1 = s.p1; ret2 = s.p2;
518     return sqrt(s.d2);
519 }
520 double farthestPointsConvex(const Point* p, int n, Point& ret1,
    ↪ Point& ret2)
521 {
522     double d2 = 0;
523     for (int i = n - 1, j = n - 2; i > 0; i--){
524         while (1){
525             double td2 = p[i].dis2(p[j]);
526             if (td2 > d2){
527                 d2 = td2;
528                 ret1 = p[i]; ret2 = p[j];
529             }
530             if (!j)break;
531             j--;
532         }

```



```

533     }
534     return sqrt(d2);
535 }
536 double farthestPointsSorted(const Point* p, int n, Point& ret1,
    ↪ Point& ret2)
537 {
538     vector<Point> v;
539     v.reserve(n);
540     //convexHullSorted(p, n, &*v.begin());
541     return farthestPointsConvex(&*v.begin(), v.size(), ret1, ret2);
542 }
543
544 int circleLineRelation(const Point& o, double r, const Point& p1,
    ↪ const Point& p2)
545 {
546     double d = o.lineDis(p1, p2);
547     if (LESS(d, r))return 1;
548     if (LESS(r, d))return 3;
549     return 2;
550 }
551 int circleCircleRelation(const Point& o1, double r1, const Point&
    ↪ o2, double r2)
552 {
553     double r = o1.dis(o2);
554     if (LESS(r1 + r2, r))return 4;
555     if (!LESS_EQUAL(r1 + r2, r))return 3;
556     double sub = abs(r1 - r2);
557     if (LESS(sub, r))return 2;
558     if (!LESS_EQUAL(sub, r))return 1;
559     return 0;
560 }
561 bool circleLineSegIntersect(const Point& o, double r, const Point&
    ↪ p1, const Point& p2)
562 //include extream points.
563 {
564     int t1 = p1.inCircle(o, r), t2 = p2.inCircle(o, r);
565     if (t1 >= 0 || t2 >= 0)
566         return t1 != 1 || t2 != 1;
567     double t = o.lineRelation(p1, p2);
568     if (t >= 1 || t <= 0)return false;
569     Point foot = p1 + (p2 - p1)*r;
570     return foot.inCircle(o, r) >= 0;
571 }
572 void circleLineIntersect(const Point& o, double r, const Point&
    ↪ p1, const Point& p2, Point& ret1, Point& ret2)
573 {

```

```

574     Point foot = o.footPoint(p1, p2);
575     double t = sqrt((r*r - o.dis2(foot)) / p1.dis2(p2));;
576     ret1 = foot + (p2 - p1)*t;
577     ret2 = foot * 2 - ret1;
578 }
579 void circleCircleIntersect(const Point& o1, double r1, const
    ↪ Point& o2, double r2, Point& ret1, Point& ret2)
580 {
581     double d2 = o1.dis2(o2);
582     double t1 = (r1*r1 - r2*r2) / (2 * d2) + 0.5;
583     double t2 = sqrt(r1*r1 / d2 - t1*t1);
584     Point foot = o1 + (o2 - o1)*t1;
585     ret1 = foot + (o2 - o1).rotate90()*t2;
586     ret2 = foot * 2 - ret1;
587 }
588 void Point::pointcut(const Point& o, double r, Point& ret1, Point&
    ↪ ret2)const
589 {
590     double t1 = r*r / dis2(o);
591     Point foot = o + (o - *this)*t1;
592     double t2 = sqrt(t1 - t1*t1);
593     ret1 = foot + (*this - o).rotate90()*t2;
594     ret2 = foot * 2 - ret1;
595 }
596 Point Point::nearestPoint(const Point& o, double r)const
597 {
598     Point p = *this - o;
599     double d = p.r();
600     if (EQUAL(d, 0))return o;
601     return o + p*(r / d);
602 }
603 //Upset the order before using this function.
604 double minCoveringCircle(const Point* p, int n, Point& ret)
605 {
606     if (n == 1){ ret = p[0]; return 0; }
607     double r2 = p[0].dis2(p[1]);
608     ret = (p[0] + p[1]) * 0.5;
609     for (int i = 2; i < n; i++){
610         if (LESS(r2, ret.dis2(p[i]))){
611             ret = (p[0] + p[i]) * 0.5;
612             r2 = p[0].dis2(p[i]);
613             for (int j = 1; j < i; j++){
614                 if (LESS(r2, ret.dis2(p[j]))){
615                     ret = (p[i] + p[j]) * 0.5;
616                     r2 = p[i].dis2(p[j]);
617                     for (int k = 0; k < j; k++){

```

```

618         if (LESS(r2, ret.dis2(p[k]))){
619             ret = circumcenter(p[i], p[j], p[k]);
620             r2 = ret.dis2(p[k]);
621         }
622     }
623 }
624 }
625 }
626 }
627 return sqrt(r2);
628 }
629 int unitCoveringCircle(const Point* p, int n, double r)
630 {
631     int ret = 0;
632     vector<pair<double, bool>> v;
633     v.reserve(2 * n);
634     double t = r*r * 4;
635     for (int i = 0; i < n; i++){
636         v.clear();
637         int value = 0;
638         for (int j = 0; j < n; j++){
639             if (LESS_EQUAL(p[i].dis2(p[j]), t) && i != j){
640                 double a = p[j].angle(p[i]);
641                 double b = acos(p[i].dis(p[j]) / r / 2);
642                 double t1 = a - b, t2 = a + b;
643                 if (t1 < -PI / 2){
644                     if (t2 < -PI / 2){
645                         a += 2 * PI;
646                         b += 2 * PI;
647                     }
648                     else value++;
649                 }
650                 v.push_back(make_pair(t1, true));
651                 v.push_back(make_pair(t2, false));
652             }
653         }
654         sort(v.begin(), v.end());
655         if (value > ret)ret = value;
656         for (unsigned int j = 0; j < v.size(); j++){
657             if (v[j].second){
658                 value++;
659                 if (value > ret)ret = value;
660             }
661             else value--;
662         }
663     }

```

```
664     return ret;  
665 }
```

2 Data Structures

2.1 Cartesian Tree

```
1  /*
2  @title: Cartesian Tree 笛卡尔树
3  @description:
4      Cartesian Tree 笛卡尔树
5      可以实现线性时间内建立具有 BST 性质的树
6  @structure:
7      CartesianTreeNode:
8          parent: 父指针
9          l: 左孩子指针
10         r: 右孩子指针
11  @arguments:
12      BuildFromArray:
13          value: 源数组
14          N: 数组大小
15          index: 源数组的逆映射数组
16          tree: 目标建树数组内存首地址
17          stack: 堆栈空间
18  @performance:
19      BuildFromArray:
20          Time:  $O(N)$ 
21          Space:  $O(N)$ 
22  @dependence: null
23  @range:
24      for  $i$  in  $[0, N)$ 
25          value[i] in  $[0, N)$ 
26          index[i] in  $[0, N)$ 
27          |value| = |index| = |tree| = |stack| =  $N$ 
28  @note:
29      value 与 index 互为逆映射故满足双射性质
30          index[value[i]] == i
31          value[index[i]] == i
32      index 无须在函数外初始化, 建树过程可以计算 index
33      stack 无须在函数外初始化, 但建树过程对 stack 有污染
34      最后结束迭代的时候栈底一定为 value[0]
35      笛卡尔树的树根一定为 value[0]
36      因此笛卡尔树的 parent 不一定要保存, 仅保存孩子指针也可以完成遍历
37  */
38
39  struct CartesianTreeNode {
40      int parent, left, right;
41  };
```

```

42
43 void BuildFromArray(int *value, int N, int *index,
    ↳ CartesianTreeNode *tree,
44                     int *stack) {
45     // 计算逆映射
46     for (int i = 0; i < N; i++) {
47         index[value[i]] = i;
48     }
49     // 初始化节点
50     for (int i = 0; i < N; i++) {
51         tree[i].parent = tree[i].left = tree[i].right = -1;
52     }
53     int size = 0; // 初始化清空栈
54     for (int i = 0; i < N; i++) {
55         int nextSize = size;
56         // 维护单调栈
57         while (nextSize > 0 && index[stack[nextSize - 1]] > index[i])
58             ↳ {
59                 nextSize--;
60             }
61         // 下面两个 if 语句块的顺序可变
62         if (nextSize > 0) { // 栈中有元素
63             // 当前元素为栈顶元素的右孩子
64             int top = stack[nextSize - 1];
65             tree[i].parent = top;
66             tree[top].right = i;
67         }
68         if (nextSize < size) { // 弹过栈
69             // 最后出栈的元素为当前元素的左孩子
70             int lastPop = stack[nextSize];
71             tree[lastPop].parent = i;
72             tree[i].left = lastPop;
73         }
74         stack[nextSize++] = i; // 入栈
75         size = nextSize;      // 更新栈大小
76     }
77 }

```

2.2 mergeable heap

```
1  int n;           //点数
2  struct node
3  {
4      int v,dis;
5      node *l,*r;
6  }mem[maxn],*head[maxn];
7  int cnt;
8  node* merge(node* a,node* b)
9  {
10     if (a==mem) return b;
11     if (b==mem) return a;
12     if (a->v<b->v) swap(a,b);
13     a->r=merge(a->r,b);
14     if (a->r->dis>a->l->dis) swap(a->l,a->r);
15     if (a->r==mem) a->dis=0;
16     else a->dis=a->r->dis+1;
17     return a;
18 }
19 void init()
20 {
21     mem[0].dis=-1;
22     mem[0].l=mem[0].r=mem;
23     for (int i=1;i<=n;i++)
24     {
25         mem[i].l=mem[i].r=mem;
26         head[i]=mem+i;
27     }
28 }
29 //BZOJ 2809
30 int m;
31 queue<int> q;
32 int f[maxn],c[maxn],l[maxn],ind[maxn],ths[maxn];
33 long long cost[maxn],ans;
34 int main()
35 {
36     freopen("in.txt","r",stdin);
37     freopen("out.txt","w",stdout);
38     scanf("%d%d",&n,&m);
39     init();
40     for (int i=1;i<=n;i++)
41     {
42         scanf("%d%d%d",&f[i],&c[i],&l[i]);
43         ind[f[i]]++;
44     }
```

```

45     for (int i=1;i<=n;i++)
46     {
47         if (ind[i]==0) q.push(i);
48         ths[i]=1;cost[i]=c[i];
49         mem[i].v=c[i];
50     }
51     while(!q.empty())
52     {
53         int now=q.front();q.pop();
54         while(cost[now]>m)
55         {
56             cost[now]-=head[now]->v;
57             head[now]=merge(head[now]->l,head[now]->r);
58             ths[now]--;
59         }
60         ans=max(ans,1ll*l[now]*ths[now]);
61         if (f[now]!=0)
62         {
63             head[f[now]]=merge(head[f[now]],head[now]);
64             ths[f[now]]+=ths[now];
65             cost[f[now]]+=cost[now];
66             if ((--ind[f[now]])==0) q.push(f[now]);
67         }
68     }
69     printf("%lld",ans);
70     return 0;
71 }

```


2.3 Segment Tree

```
1 struct tree
2 {
3     int mi,ls,rs,ll,rr,add;
4 }a[3*maxn];
5
6 int c[maxn];
7 int i,j,k,l,m,n,t,T;
8
9 void update(int x)
10 {
11     if(a[x].ls+a[x].rs==0)return;
12     a[x].mi=min(a[a[x].ls].mi,a[a[x].rs].mi);
13 }
14
15 void downdate(int x)
16 {
17     if(a[x].ls+a[x].rs==0)return;
18     if(a[x].add==0)return;
19     a[a[x].ls].add+=a[x].add;
20     a[a[x].ls].mi+=a[x].add;
21     a[a[x].rs].add+=a[x].add;
22     a[a[x].rs].mi+=a[x].add;
23     a[x].add=0;
24 }
25
26 void mt(int l,int r)
27 {
28     if(l==r)
29     {
30         a[k].ll=a[k].rr=l;
31         a[k].mi=c[l];
32         a[k].ls=a[k].rs=0;
33         return;
34     }
35     int t=k;
36     int mid=(l+r)>>1;
37     a[t].ll=l;a[t].rr=r;
38     k++;a[t].ls=k;mt(l,mid);
39     k++;a[t].rs=k;mt(mid+1,r);
40     update(t);
41 }
42
43 void add(int l,int r,int nu,int x)
44 {
```

```

45     if(a[x].ll==l && a[x].rr==r)
46     {
47         a[x].add+=nu;
48         a[x].mi+=nu;
49         return;
50     }
51     downdate(x);
52     int mid=(a[x].ll+a[x].rr)>>1;
53     if(mid<l)add(l,r,nu,a[x].rs);
54     else if(mid>=r)add(l,r,nu,a[x].ls);
55     else
56     {
57         add(l,mid,nu,a[x].ls);
58         add(mid+1,r,nu,a[x].rs);
59     }
60     update(x);
61 }
62
63 int find(int l,int r,int x)
64 {
65     if(a[x].ll==l && a[x].rr==r)return a[x].mi;
66     downdate(x);
67     update(x);
68     int mid=(a[x].ll+a[x].rr)>>1;
69     if(mid<l)return find(l,r,a[x].rs);
70     if(mid>=r)return find(l,r,a[x].ls);
71     return min(find(l,mid,a[x].ls),find(mid+1,r,a[x].rs));
72 }
73
74 inline void read(int &x) {
75     char ch=getchar();
76     while(ch<'0' || ch>'9') ch=getchar();
77     x=0;
78     while(ch<='9'&&ch>='0'){
79         x=x*10+ch-'0';
80         ch=getchar();
81     }
82 }
83
84 int main()
85 {
86     read(n);read(m);
87     for(i=1;i<=n;i++)read(c[i]);
88     k=1;mt(1,n);
89     for(i=1;i<=m;i++)
90     {

```

```
91     read(l);read(j);read(k);
92     if(find(j,k,1)<l)
93     {
94         printf("-1\n%d",i);
95         break;
96     }
97     add(j,k,-l,1);
98 }
99 if(i>m)printf("0");
100 return 0;
101 }
```

2.4 Splay

```
1 struct tree
2 {
3     int key,size,le,ri,add,rev,min,pre;
4 }a[maxn];
5 int n,T,node;
6 int s[maxn];
7
8 void pushdown(int cur)
9 {
10     int ls=a[cur].le,rs=a[cur].ri;
11     if(a[cur].add>0)
12     {
13         a[ls].add+=a[cur].add;
14         a[rs].add+=a[cur].add;
15         a[ls].key+=a[cur].add;
16         a[rs].key+=a[cur].add;
17         a[ls].min+=a[cur].add;
18         a[rs].min+=a[cur].add;
19         a[cur].add=0;
20     }
21     if(a[cur].rev>0)
22     {
23         a[ls].rev^=1;
24         a[rs].rev^=1;
25         a[cur].le=rs;
26         a[cur].ri=ls;
27         a[cur].rev=0;
28     }
29 }
30
31 void update(int cur)
32 {
33     int ls=a[cur].le,rs=a[cur].ri;
34     a[cur].size=a[ls].size+a[rs].size+1;
35     a[cur].min=a[cur].key;
36     if(ls&&a[ls].min<a[cur].min)a[cur].min=a[ls].min;
37     if(rs&&a[rs].min<a[cur].min)a[cur].min=a[rs].min;
38 }
39
40 void leftrotate(int x)
41 {
42     int y=a[x].ri,p=a[x].pre;
43     a[x].ri=a[y].le;
44     if(a[x].ri)a[a[x].ri].pre=x;
```

```

45     a[y].le=x;
46     a[x].pre=y;
47     a[y].pre=p;
48     if(!p)T=y;
49     else
50         a[p].ri==x?a[p].ri=y:a[p].le=y;
51     update(x);
52 }
53
54 void rightrotate(int x)
55 {
56     int y=a[x].le,p=a[x].pre;
57     a[x].le=a[y].ri;
58     if(a[x].le)a[a[x].le].pre=x;
59     a[y].ri=x;
60     a[x].pre=y;
61     a[y].pre=p;
62     if(!p)T=y;
63     else
64         a[p].ri==x?a[p].ri=y:a[p].le=y;
65     update(x);
66 }
67
68 void splay(int x,int goal)
69 {
70     int y,z;
71     while(1)
72     {
73         if((y=a[x].pre)==goal)break;
74         if((z=a[y].pre)==goal)
75             a[y].ri==x?leftrotate(y):rightrotate(y);
76         else
77         {
78             if(a[z].ri==y)
79             {
80                 if(a[y].ri==x)
81                     leftrotate(z),leftrotate(y);
82                 else
83                     rightrotate(y),leftrotate(z);
84             }
85             else
86             {
87                 if(a[y].le==x)
88                     rightrotate(z),rightrotate(y);
89                 else
90                     leftrotate(y),rightrotate(z);

```

```

91     }
92 }
93 }
94 update(x);
95 }
96
97 void rotateto(int k,int goal)
98 {
99     int i=T;
100    while(1)
101    {
102        pushdown(i);
103        if(a[a[i].le].size+1==k)break;
104        if(k<=a[a[i].le].size)i=a[i].le;
105        else k-=a[a[i].le].size+1,i=a[i].ri;
106    }
107    splay(i,goal);
108 }
109
110 void newnode(int &cur,int v)
111 {
112     cur=++node;
113     a[cur].min=a[cur].key=v;
114     a[cur].size=1;
115     a[cur].le=a[cur].ri=a[cur].rev=a[cur].add=0;
116 }
117
118 void build(int &cur,int x,int y,int p)
119 {
120     int mid=(x+y)>>1;
121     newnode(cur,s[mid]);
122     a[cur].pre=p;
123     if(x==y)return;
124     if(x<mid)build(a[cur].le,x,mid-1,cur);
125     if(y>mid)build(a[cur].ri,mid+1,y,cur);
126     update(cur);
127 }
128
129 void init(int n)
130 {
131     int i;
132     memset(s,0,sizeof(s));
133     memset(a,0,sizeof(a));
134     for(i=1;i<=n;i++)scanf("%d",&s[i]);
135     T=node=0;
136     build(T,0,n+1,0);

```

```

137 }
138
139 void Add(int x,int y,int z)
140 {
141     int k;
142     rotateto(x,0);rotateto(y+2,T);
143     k=a[a[T].ri].le;
144     a[k].add+=z;a[k].key+=z;a[k].min+=z;
145 }
146
147 void Reverse(int x,int y)
148 {
149     int k;
150     rotateto(x,0);rotateto(y+2,T);
151     k=a[a[T].ri].le;
152     a[k].rev^=1;
153 }
154
155 void Revolve(int x,int y,int z)
156 {
157     int k=z%(y-x+1),t;
158     if(k)
159     {
160         rotateto(x,0);rotateto(y-k+2,T);
161         t=a[a[T].ri].le;
162         a[a[T].ri].le=0;
163         update(a[T].ri);update(T);
164         rotateto(x+k,0);rotateto(x+k+1,T);
165         a[a[T].ri].le=t;a[t].pre=a[T].ri;
166         update(a[T].ri);update(T);
167     }
168 }
169
170 void Insert(int x,int y)
171 {
172     rotateto(x+1,0);rotateto(x+2,T);
173     newnode(a[a[T].ri].le,y);
174     a[a[a[T].ri].le].pre=a[T].ri;
175     update(a[T].ri);update(T);
176 }
177
178 void Delete(int x)
179 {
180     rotateto(x,0);rotateto(x+2,T);
181     a[a[T].ri].le=0;
182     update(a[T].ri);update(T);

```

```
183 }
184
185 void Min(int x, int y)
186 {
187     rotateto(x, 0); rotateto(y+2, T);
188     printf("%d\n", a[a[a[T].ri].le].min);
189 }
```


2.5 Treap

```
1  struct data{
2      int l,r,v,size,rnd,w;
3  }tr[100005];
4  int n,size,root,ans;
5
6  void update(int k)//更新结点信息
7  {
8      tr[k].size=tr[tr[k].l].size+tr[tr[k].r].size+tr[k].w;
9  }
10
11 void rturn(int &k)
12 {
13     int t=tr[k].l;tr[k].l=tr[t].r;tr[t].r=k;
14     tr[t].size=tr[k].size;update(k);k=t;
15 }
16
17 void lturn(int &k)
18 {
19     int t=tr[k].r;tr[k].r=tr[t].l;tr[t].l=k;
20     tr[t].size=tr[k].size;update(k);k=t;
21 }
22
23 void insert(int &k,int x)
24 {
25     if(k==0)
26     {
27         size++;k=size;
28         tr[k].size=tr[k].w=1;tr[k].v=x;tr[k].rnd=rand();
29         return;
30     }
31     tr[k].size++;
32     if(tr[k].v==x)tr[k].w++;
33     else if(x>tr[k].v)
34     {
35         insert(tr[k].r,x);
36         if(tr[tr[k].r].rnd<tr[k].rnd)lturn(k);
37     }
38     else
39     {
40         insert(tr[k].l,x);
41         if(tr[tr[k].l].rnd<tr[k].rnd)rturn(k);
42     }
43 }
44
```

```

45 void del(int &k, int x)
46 {
47     if(k==0) return;
48     if(tr[k].v==x)
49     {
50         if(tr[k].w>1)
51         {
52             tr[k].w--; tr[k].size--; return;
53         }
54         if(tr[k].l*tr[k].r==0) k=tr[k].l+tr[k].r;
55         else if(tr[tr[k].l].rnd<tr[tr[k].r].rnd)
56             rturn(k), del(k, x);
57         else lturn(k), del(k, x);
58     }
59     else if(x>tr[k].v)
60         tr[k].size--, del(tr[k].r, x);
61     else tr[k].size--, del(tr[k].l, x);
62 }
63
64 int query_rank(int k, int x)
65 {
66     if(k==0) return 0;
67     if(tr[k].v==x) return tr[tr[k].l].size+1;
68     else if(x>tr[k].v)
69         return tr[tr[k].l].size+tr[k].w+query_rank(tr[k].r, x);
70     else return query_rank(tr[k].l, x);
71 }
72
73 int query_num(int k, int x)
74 {
75     if(k==0) return 0;
76     if(x<=tr[tr[k].l].size)
77         return query_num(tr[k].l, x);
78     else if(x>tr[tr[k].l].size+tr[k].w)
79         return query_num(tr[k].r, x-tr[tr[k].l].size-tr[k].w);
80     else return tr[k].v;
81 }
82
83 void query_pro(int k, int x)
84 {
85     if(k==0) return;
86     if(tr[k].v<x)
87     {
88         ans=k; query_pro(tr[k].r, x);
89     }
90     else query_pro(tr[k].l, x);

```

```

91 }
92
93 void query_sub(int k,int x)
94 {
95     if(k==0)return;
96     if(tr[k].v>x)
97     {
98         ans=k;query_sub(tr[k].l,x);
99     }
100     else query_sub(tr[k].r,x);
101 }
102
103 int main()
104 {
105     scanf("%d",&n);
106     int opt,x;
107     for(int i=1;i<=n;i++)
108     {
109         scanf("%d%d",&opt,&x);
110         switch(opt)
111         {
112             case 1:insert(root,x);break;
113             case 2:del(root,x);break;
114             case 3:printf("%d\n",query_rank(root,x));break;
115             case 4:printf("%d\n",query_num(root,x));break;
116             case
117                 ↪ 5:ans=0;query_pro(root,x);printf("%d\n",tr[ans].v);break;
118             case
119                 ↪ 6:ans=0;query_sub(root,x);printf("%d\n",tr[ans].v);break;
120         }
121     }
122     return 0;
123 }

```

2.6 Tree Cut(edge)

```
1  #include<cstdio>
2  #include<vector>
3  using namespace std;
4  vector<pair<int, int>> v[200001]; //边及该边的编号
5  int w[200001]; //边权
6  int n, cnt;
7  int father[200001], depth[200001], top[200001], id[200001];
8  int f[200001]; //边在树状数组 ( 线段树 ) 中的位置
9  int tmp[200001];
10 int dfs1(int i, int fa)
11 {
12     father[i] = fa;
13     depth[i] = depth[fa] + 1;
14     tmp[i] = -1;
15     int ret = 0, maxSize = 0;
16     for (unsigned int j = 0; j < v[i].size(); j++){
17         int t = v[i][j].first;
18         if (t == fa) continue;
19         int size = dfs1(t, i);
20         ret += size;
21         if (size > maxSize){
22             maxSize = size;
23             tmp[i] = j;
24         }
25     }
26     return ret + 1;
27 }
28 void dfs2(int i, int tp, int index)
29 {
30     top[i] = tp;
31     id[i] = cnt;
32     f[index] = cnt++;
33     if (tmp[i] != -1)
34         dfs2(v[i][tmp[i]].first, tp, v[i][tmp[i]].second);
35     for (unsigned int j = 0; j < v[i].size(); j++){
36         int t = v[i][j].first;
37         if (t != father[i] && j != tmp[i])
38             dfs2(t, t, v[i][j].second);
39     }
40 }
41 int queryTree(int s, int t)
42 {
43     int ret = 0;
44     int top1 = top[s], top2 = top[t];
```

```

45 while (top1 != top2){
46     if (depth[top1] < depth[top2]){
47         ret += sum(id[t]) - sum(id[top2] - 1);
48         t = father[top2]; top2 = top[t];
49     }
50     else{
51         ret += sum(id[s]) - sum(id[top1] - 1);
52         s = father[top1]; top1 = top[s];
53     }
54 }
55 if (s != t){
56     if (depth[s] > depth[t])swap(s, t);
57     ret += sum(id[t]) - sum(id[s]);
58 }
59 return ret;
60 }
61 void init()
62 {
63     cnt = 0;
64     dfs1(1, 1);
65     dfs2(1, 1, 0);
66     for (int i = 1; i < n; i++)
67         tree[f[i]] = w[i];
68     build();
69 }
70 int main()
71 {
72     int q, cur;
73     scanf("%d%d", &n, &q, &cur);
74     for (int i = 1; i < n; i++){
75         int s, t, value;
76         scanf("%d%d", &s, &t, &value);
77         v[s].push_back(make_pair(t, i));
78         v[t].push_back(make_pair(s, i));
79         w[i] = value;
80     }
81     init();
82     for (int i = 0; i < q; i++){
83         int t, u, value;
84         scanf("%d", &t, &u);
85         if (t == 0){
86             printf("%d\n", queryTree(cur, u));
87             cur = u;
88         }
89         else{
90             scanf("%d", &value);

```

```
91         add(f[u], value - w[u]);
92         w[u] = value;
93     }
94 }
95 return 0;
96 }
```

2.7 Tree Cut(vertex)

```
1  #include<cstdio>
2  #include<cstring>
3  #include<vector>
4  using namespace std;
5  vector<int> v[100001];
6  int n, cnt, color;
7  int father[100001], depth[100001], top[100001], id[100001],
   → son[100001];
8  struct Tree{
9      int maxValue, maxId, delta;
10     bool set;
11 }tree[1 << 18];
12 int treeLen;
13 int dfs1(int i, int fa)
14 {
15     father[i] = fa;
16     depth[i] = depth[fa] + 1;
17     son[i] = 0;
18     int ret = 0, maxSize = 0;
19     for (unsigned int j = 0; j < v[i].size(); j++){
20         int t = v[i][j];
21         if (t == fa)continue;
22         int size = dfs1(t, i);
23         ret += size;
24         if (size > maxSize){
25             maxSize = size;
26             son[i] = t;
27         }
28     }
29     return ret + 1;
30 }
31 void dfs2(int i, int tp)
32 {
33     top[i] = tp;
34     id[i] = cnt++;
35     if (son[i])dfs2(son[i], tp);
36     for (unsigned int j = 0; j < v[i].size(); j++){
37         int t = v[i][j];
38         if (t != father[i] && t != son[i])dfs2(t, t);
39     }
40 }
41 void init()
42 {
43     cnt = 0; depth[1] = 0;
```

```

44     dfs1(1, 1);
45     dfs2(1, 1);
46     for (treeLen = 1; treeLen < n; treeLen *= 2);
47     memset(tree, 0, sizeof(Tree) * 2 * treeLen);
48 }
49 void pushDown(int i)
50 {
51     if (tree[i].set){
52         tree[2 * i].set = tree[2 * i + 1].set = true;
53         tree[2 * i].delta = tree[2 * i + 1].delta = tree[i].delta;
54         tree[i].delta = 0; tree[i].set = false;
55     }
56     else if (tree[i].delta){
57         tree[2 * i].delta += tree[i].delta;
58         tree[2 * i + 1].delta += tree[i].delta;
59         tree[i].delta = 0;
60     }
61 }
62 int queryL, queryR;
63 void addInternal(int i, int l, int len)
64 {
65     if (queryL <= l && queryR >= l + len){
66         tree[i].delta++;
67         return;
68     }
69     len >>= 1; pushDown(i);
70     int mid = l + len;
71     if (mid > queryL)addInternal(2 * i, l, len);
72     if (mid < queryR)addInternal(2 * i + 1, mid, len);
73 }
74 inline void addValue(int l, int r){
75     queryL = l; queryR = r;
76     addInternal(1, 0, treeLen);
77 }
78 int addTree(int s, int t)
79 {
80     int ret = 0;
81     int top1 = top[s], top2 = top[t];
82     while (top1 != top2){
83         if (depth[top1] < depth[top2]){
84             addValue(id[top2], id[t] + 1);
85             t = father[top2]; top2 = top[t];
86         }
87         else{
88             addValue(id[top1], id[s] + 1);
89             s = father[top1]; top1 = top[s];

```



```

90     }
91 }
92 if (depth[s] > depth[t]) swap(s, t);
93 addValue(id[s], id[t] + 1);
94 return ret;
95 }
96 void process(int i)
97 {
98     if (tree[i].set){
99         if (tree[i].delta > tree[i].maxValue){
100             tree[i].maxValue = tree[i].delta;
101             tree[i].maxId = color;
102         }
103         return;
104     }
105     pushDown(i);
106     process(2 * i);
107     process(2 * i + 1);
108 }
109 inline void pushDownColor(int i, int j){
110     if (tree[j].maxValue < tree[i].maxValue
111         || (tree[j].maxValue == tree[i].maxValue && tree[i].maxId <
112             tree[j].maxId)){
113         tree[j].maxValue = tree[i].maxValue;
114         tree[j].maxId = tree[i].maxId;
115     }
116 }
117 void getAns(int i)
118 {
119     if (i < treeLen){
120         pushDownColor(i, 2 * i);
121         pushDownColor(i, 2 * i + 1);
122         getAns(2 * i);
123         getAns(2 * i + 1);
124     }
125 }
126 vector<pair<int, int>> z[100001];
127 int main()
128 {
129     int m;
130     while (scanf("%d%d", &n, &m) == 2 && n){
131         for (int i = 1; i <= n; i++){
132             v[i].clear();
133             for (int i = 1; i < n; i++){
134                 int s, t;
135                 scanf("%d%d", &s, &t);

```

```

135     v[s].push_back(t);
136     v[t].push_back(s);
137 }
138 for (int i = 0; i < m; i++){
139     int s, t, w;
140     scanf("%d%d%d", &s, &t, &w);
141     z[w].push_back(make_pair(s, t));
142 }
143 init();
144 for (color = 1; color <= 100000; color++){
145     tree[1].set = true; tree[1].delta = 0;
146     for (unsigned int j = 0; j < z[color].size(); j++)
147         addTree(z[color][j].first, z[color][j].second);
148     process(1);
149     z[color].clear();
150 }
151 getAns(1);
152 for (int i = 1; i <= n; i++)
153     printf("%d\n", tree[treeLen + id[i]].maxId);
154 }
155 return 0;
156 }

```

3 Graph

3.1 (Euler)Fluery

```
1 //起点很重要
2 //stack 栈深度会超过点数，最大为边数
3 const int maxn=10005;
4 int stac[maxn],sta;
5 struct edge
6 {
7     int p,q;
8     edge(int p=0,int q=0):p(p),q(q){}
9 }edg[2*maxn];
10 bool used[2*maxn];
11 vector<int> g[maxn];
12 int du[maxn];
13 int i,j,k,l,m,n;
14
15 int other(int num,int x)
16 {
17     return x==edg[num].p?edg[num].q:edg[num].p;
18 }
19
20 void dfs(int x)
21 {
22     stac[++sta]=x;
23     for(unsigned int i=0;i<g[x].size();i++)
24     {
25         if(used[g[x][i]])continue;
26         used[g[x][i]]=true;
27         dfs(other(g[x][i],x));
28         break;
29     }
30 }
31
32 void Fleury(int x)
33 {
34     sta=1;stac[sta]=x;
35     while(sta>=1)
36     {
37         x=stac[sta];
38         bool f=false;
39         for(unsigned int i=0;i<g[x].size();i++)
40         {
41             if(!used[g[x][i]]){f=true;break;}
```

```

42     }
43     if(!f)printf("%d ",stac[sta--]);
44     else
45     {
46         sta--;
47         dfs(stac[sta+1]);
48     }
49 }
50 }
51
52 int main()
53 {
54     //未判断图是否连通，为严谨可加入 bfs 检查
55     scan2(n,m);
56     memset(edg,0,sizeof(edg));
57     for(i=1;i<=n;i++)g[i].clear();
58     memset(used,0,sizeof(used));
59     memset(du,0,sizeof(du));
60     for(i=1;i<=m;i++)
61     {
62         scan2(j,k);
63         edg[i]=edge(j,k);
64         du[j]++;du[k]++;
65         g[j].push_back(i);
66         g[k].push_back(i);
67     }
68     int tot=0,st=0;
69     for(i=1;i<=n;i++)
70     {
71         if(du[i]==0){tot=3;break;}
72         if(du[i]%2==1){tot++;st=i;}
73     }
74     if(st==0)st=1;
75     if(tot>2)printf("HeHeDa!");else Fleury(st);
76     return 0;
77 }

```

3.2 0-1 分数规划

```
1  const int maxn=1005;
2  double a[maxn],b[maxn];
3  struct hei
4  {
5      double num;
6      int pos;
7      hei(double num=0,int pos=0):num(num),pos(pos){}
8      bool operator < (struct hei p)const
9      {return num>p.num;}
10 }d[maxn];
11 double p,q,ans,l;
12 int i,n,m;
13
14 int main()
15 {
16     while(scan2(n,m)==2 && n+m>0)
17     {
18         m=n-m;
19         for(i=1;i<=n;i++)scanf("%lf",&a[i]);
20         for(i=1;i<=n;i++)scanf("%lf",&b[i]);
21         l=0;
22         while(true)
23         {
24             ans=l;
25             for(i=1;i<=n;i++)d[i]=hei(a[i]-ans*b[i],i);
26             sort(d+1,d+n+1);
27             double fz,fm;
28             fz=fm=0.0;
29             for(i=1;i<=m;i++)
30             {
31                 fz+=a[d[i].pos];
32                 fm+=b[d[i].pos];
33             }
34             l=fz/fm;
35             if(fabs(ans-l)<eps)break;
36         }
37         printf("%.0f\n",100.0*ans);
38     }
39     return 0;
40 }
```

3.3 Biggest Tuan

```
1 //最大独立集即补图的最大团
2 #include<cstdio>
3 #include<cstring>
4 #define N 1010
5 bool flag[N], a[N][N];
6 int ans, cnt[N], group[N], n, vis[N];
7 // 最大团：V 中取 K 个顶点，两点间相互连接
8 // 最大独立集：V 中取 K 个顶点，两点间不连接
9 // 最大团数量 = 补图中最大独立集数
10
11 bool dfs( int u, int pos ){
12     int i, j;
13     for( i = u+1; i <= n; i++){
14         if( cnt[i]+pos <= ans ) return 0;
15         if( a[u][i] ){
16             // 与目前团中元素比较，取 Non-N(i)
17             for( j = 0; j < pos; j++ ) if( !a[i][ vis[j] ] )
18                 ↪ break;
19             if( j == pos ){ // 若为空，则皆与 i 相邻，则此时将
20                 ↪ i 加入到 最大团中
21                 vis[pos] = i;
22                 if( dfs( i, pos+1 ) ) return 1;
23             }
24         }
25     }
26     if( pos > ans ){
27         for( i = 0; i < pos; i++ )
28             group[i] = vis[i]; // 最大团 元素
29         ans = pos;
30         return 1;
31     }
32     return 0;
33 }
34
35 void maxclique()
36 {
37     ans=-1;
38     for(int i=n;i>0;i--){
39         {
40             vis[0]=i;
41             dfs(i,1);
42             cnt[i]=ans;
43         }
44     }
45 }
```

3.4 dijkstra(nlogn)

```
1  using namespace std;
2  #define pii pair<int, int>
3  priority_queue<pii, vector<pii>, greater<pii> > heap;
4  struct edge
5  {
6      int v, l, next;
7  }e[13007];
8  int n, m, S, T;
9  int tot=2, dist[2502], head[2502];
10 bool visited[2502];
11 void addedge(int x,int y,int z)
12 {
13     e[tot].v=y, e[tot].l=z, e[tot].next=head[x], head[x]=tot++;
14     e[tot].v=x, e[tot].l=z, e[tot].next=head[y], head[y]=tot++;
15 }
16 void dij(int x)
17 {
18     if (x == T) return ;
19     visited[x] = true;
20     for (int p = head[x]; p; p = e[p].next)
21         if (!visited[e[p].v] && dist[e[p].v] > (dist[x] + e[p].l))
22             dist[e[p].v] = dist[x] + e[p].l,
23             ↪ heap.push(make_pair(dist[e[p].v], e[p].v));
24     while (!heap.empty() && visited[heap.top().second])
25         heap.pop();
26     dij(heap.top().second);
27 }
28 int main()
29 {
30     scanf("%d%d%d%d",&n,&m,&S,&T);
31     int x, y, z;
32     for (int i=1;i<=m;i++)
33         scanf("%d%d%d",&x,&y,&z), addedge(x, y, z);
34     for (int i=1;i<=n;i++)
35         dist[i]=0x7fffffff;
36     dist[S]=0;
37     dij(S);
38     printf("%d\n",dist[T]);
39     return 0;
40 }
```

3.5 Directed-MST

```
1  #define type int//type 可选择 int 或者 double
2
3  const type inf=2147483640;
4  const int maxn=1005;
5
6  int pre[maxn],id[maxn],vis[maxn];
7  type in[maxn];
8
9  struct edge
10 {
11     int from,to;
12     type cost;
13     edge(int from=0,int to=0,type
14         ↪ cost=0):from(from),to(to),cost(cost){}
15 }edg[10005];
16
17 type ZLEdmonds(int n,int m,int root)//自环在输入建图时直接忽略,如
18 ↪ 需加入,可另存
19 {
20     type tot=0.0;
21     //判断是否有树
22     while(true)
23     {
24         for(int i=1;i<=n;i++)in[i]=inf;
25         for(int i=1;i<=m;i++)
26         {
27             int u=edg[i].from;
28             int v=edg[i].to;
29             if(edg[i].cost<in[v] && u!=v){pre[v]=u;in[v]=edg[i].cost;}
30         }
31         for(int i=1;i<=n;i++)if(i!=root && in[i]==inf)return -1;
32         //找环
33         int cnt=1;
34         memset(id,0,sizeof(id));
35         memset(vis,0,sizeof(vis));
36         in[root]=0;
37         for(int i=1;i<=n;i++)//标记每个环
38         {
39             tot+=in[i];
40             int v=i;
41             while(vis[v]!=i && id[v]==0 && v!=root)
42             {vis[v]=i;v=pre[v];}
43             if(v!=root && id[v]==0)//缩点
44             {
```



```

43         for(int u=pre[v];u!=v;u=pre[u])id[u]=cnt;
44         id[v]=cnt++;
45     }
46 }
47 if(cnt==1)break;
48 for(int i=1;i<=n;i++)if(id[i]==0)id[i]=cnt++;
49 //建立新图
50 for(int i=1;i<=m;i++)
51 {
52     int u=edg[i].from;
53     int v=edg[i].to;
54     edg[i].from=id[u];
55     edg[i].to=id[v];
56     if(id[u]!=id[v])edg[i].cost-=in[v];
57 }
58 n=cnt-1;
59 root=id[root];
60 }
61 return tot;
62 }
63
64 int main()
65 {
66
67     return 0;
68 }

```

3.6 Hamilton

```
1  /*
2  【题目来源】
3  http://poj.org/problem?id=2438
4  【题目分析】
5  有敌对关系的小朋友，不能坐在一起。最后围成一个圈，吃饭。。。
6  将小朋友看成点，有敌对关系的看成没有边，最后构成一个回路。
7  哈密顿回路。
8
9  【小小总结】
10 哈密顿回路
11 充分条件：
12 无向连通图中任意 2 点度数之和大于等于顶点数，则必定存在哈密顿回路。
13
14 思路分析：
15 1. 任意找两个相邻的节点  $S$  和  $T$ ，在它们基础上扩展出一条尽量长的没有重
    ↳ 复节点的路径。
16 也就是说，如果  $S$  与节点  $v$  相邻，而且  $v$  不在路径  $S \rightarrow T$  上，则可以把该
    ↳ 路径变成  $v \rightarrow S \rightarrow T$ ，然后  $v$  成为新的  $S$ 。
17 从  $S$  和  $T$  分别向两头扩展，直到无法扩为止，即所有与  $S$  或  $T$  相邻的节点
    ↳ 都在路径  $S \rightarrow T$  上。
18 2. 若  $S$  与  $T$  相邻，则路径  $S \rightarrow T$  形成了一个回路。
19 3. 若  $S$  与  $T$  不相邻，可以构造出一个回路。设路径  $S \rightarrow T$  上有  $k+2$  个节
    ↳ 点，依次为  $S, v_1, v_2, \dots, v_k$  和  $T$ 。
20 可以证明  $v_1$  到  $v_k$  中必定存在  $v_i$ ，满足  $v_i$  与  $T$  相邻，且  $v_{i+1}$  与  $S$  相
    ↳ 邻。（其实  $v_i, v_{i+1}$  与  $S, T$  同时相邻）（怎么证明就不赘述了，反正刷
    ↳ 题肯定不会叫你证）
21 找到了满足条件的节点  $v_i$  以后，就可以把原路径变成  $S \rightarrow v_{i+1} \rightarrow T \rightarrow v_i \rightarrow S$ ，
    ↳ 即形成了一个回路。（自己画图就知道了）
22 4. 现在我们有了一条没有重复节点的回路。如果它的长度为  $N$ ，则哈密顿回路
    ↳ 就找到了。
23 如果回路的长度小于  $N$ ，由于整个图是连通的，所以在该回路上，一定存在一
    ↳ 点与回路以外的点相邻。
24 那么从该点处把回路断开，就变回了一条路径。再按照步骤 1 的方法尽量扩展
    ↳ 路径，则一定有新的节点被加进来。（画图就知道了）
25 接着回到步骤 2。
26
27 伪代码：
28 思路清楚后主要是理解好伪代码，伪代码一懂代码就写出来了。关于下面步骤
    ↳ 中为什么要倒置，自己画画图就清楚了。
29  $s$  为哈密顿回路起点， $t$  为当前哈密顿回路的终点， $ans[]$  就是哈密顿回路
    ↳ 啦，默认不包含 0 顶点
30 1. 初始化，令  $s=1, t$  为任意与  $s$  相邻的点。
31 2. 若  $ans[]$  中的元素个数小于  $n$ ，则从  $t$  开始扩展，若可扩展，则把新点
    ↳  $v$  加入  $ans[]$ ，并令  $t=v$ ，继续扩展到无法扩展。
```

32 3. 将 `ans[]` 倒置, `s, t` 互换, 从 `t` (原来的 `s`) 开始扩展, 若可扩展, 则把
 → 新点 `v` 加入 `ans[]`, 并令 `t=v`, 继续扩展到无法扩展。
 33 4. 此时 `s, t` 两头都无法扩展了, 若 `s, t` 相连, 则继续步骤 5。若 `st` 不相
 → 连, 则遍历 `ans[]`, 必定会有 2 点, `ans[i]` 与 `t` 相连, `ans[i+1]` 与
 → `s` 相连,
 34 将 `ans[i+1]` 到 `t` 倒置, `t=ans[i+1]`(未倒置前的)
 35 5. `st` 相连, 此时为一个环。若 `ans[]` 个数等于 `n`, 算法结束, `ans[]` 为哈密
 → 顿回路, 如需要再添加一个起点。
 36 若 `ans[]` 个数小于 `n`, 遍历 `ans[]`, 寻找 `ans[i]`, 使得 `ans[i]` 与 `ans[]`
 → 外一点 `j` 相连, 倒置 `ans[]` 中 `s` 到 `ans[i-1]` 部分, 令 `s =`
 → `ans[i-1]`,
 37 再倒置 `ans[]` 中 `ans[i]` 到 `t` 的部分, `j` 加入 `ans[]`, `t = j`. 继续步骤 2

38
 39 下面去掉 `main` 函数, 就是求解哈密顿回路的模版了。

```
40 */
41 #include <iostream>
42 #include <cstring>
43 #include <algorithm>
44 using namespace std;
45
46 #define Max 500
47
48 int map[Max][Max];
49 int ans[Max];
50 bool vis[Max];
51
52 //ans 数组的 index
53 int index;
54 int n, m;
55 int s, t;
56
57 void init()
58 {
59     for (int i = 0; i < Max; ++i)
60         for (int j = 0; j < Max; ++j)
61             if (i == j)
62                 map[i][j] = 0;
63             else
64                 map[i][j] = 1;
65
66     memset(ans, 0, sizeof(ans));
67     memset(vis, 0, sizeof(vis));
68     index = 0;
69 }
70
```

```

71 void reverse(int a, int b)
72 {
73     while (a < b)
74     {
75         swap(ans[a], ans[b]);
76         a++;
77         b--;
78     }
79 }
80
81 void expand()
82 {
83     while (true)
84     {
85         int i;
86         for (i = 1; i <= n; ++i)
87         {
88             if (!vis[i] && map[i][t])//未被访问且与 t 相连
89             {
90                 ans[index++] = i;
91                 vis[i] = true;
92                 t = i;
93                 break;
94             }
95         }
96         if (i > n) break;//无法扩展
97     }
98 }
99
100 void Hamilton()
101 {
102     //初始化 s = 1
103     s = 1;
104
105     //取任意连接 s 的点
106     for (int i = 1; i <= n; ++i)
107     {
108         if (map[i][s])
109         {
110             t = i;
111             break;
112         }
113     }
114     vis[s] = true;
115     vis[t] = true;

```

```

116     ans[index++] = s;
117     ans[index++] = t;
118
119     while (true)
120     {
121         //从 t 向外扩展
122         expand();
123
124         //t 扩展完毕, 倒置 ans 并交换 s,t
125         reverse(0, index-1);
126
127         swap(s, t);
128
129         //从另一头, t(原来的 s) 继续扩展
130         expand();
131
132         //若 s,t 不相连, 处理成相连
133         if (!map[s][t])
134         {
135             //在 ans[1] 到 ans[index-2] 中寻找两个相邻的且与 st 同
136             //    ↪ 时相连的点 (必存在) 因为涉及 i+1 所以 i < index-2
137             for (int i = 1; i < index-2; ++i)
138             {
139                 if (map[ans[i+1]][s] && map[ans[i]][t])
140                 {
141                     reverse(i+1, index-1); //倒置 ans[i+1] 到
142                     //    ↪ ans[index-1]
143                     t = ans[index-1]; //更新 t
144                     break;
145                 }
146             }
147         }
148
149         //若 ans 元素有 n 个, 说明算法完成
150         if (index == n) return;
151
152         //若 ans 元素不满 n 个, ans[] 中寻找与未被遍历过的点相连的
153         //    ↪ 点, 但这一点必定不是 s,t. 因为 s,t 已经遍历到无法遍历才
154         //    ↪ 能走到这一步
155         for (int j = 1; j <= n; ++j)
156         {
157             if (!vis[j])
158             {
159                 int i;
160                 for (i = 1; i < index-1; ++i) //排除 st

```

```

157         {
158             if (map[ans[i]][j])
159             {
160                 s = ans[i-1];
161                 t = j;
162                 reverse(0, i-1);
163                 reverse(i, index-1);
164                 ans[index++] = j;
165                 vis[j] = true;
166                 break;
167             }
168         }
169         if (map[ans[i]][j]) break; //记得有 2 个循环, 要
        ↪ break 两次
170     }
171 }
172 //继续返回, 从 t 扩展。。
173 }
174 }
175
176 int main()
177 {
178     while (cin >> n >> m, n||m)
179     {
180         n *= 2;
181         init();
182         int temp1, temp2;
183         for (int i = 0; i < m; ++i)
184         {
185             cin >> temp1 >> temp2;
186             map[temp1][temp2] = 0;
187             map[temp2][temp1] = 0;
188         }
189         Hamilton();
190         cout << ans[0];
191         for (int i = 1; i < index; ++i)
192             cout << ' ' << ans[i];
193         cout << endl;
194     }
195 }

```

3.7 K-shortest path

```
1  int n,m,s,t,k,dis[MAXN];
2  struct node
3  {
4      int v,c;
5      node(int v,int c):v(v),c(c){}
6      inline bool operator<(const node &b) const//用于优先队列先
        ↳ 出的条件
7      {
8          return c+dis[v]>b.c+dis[b.v];
9      }
10 };
11 vector<node> map1[MAXN];//用于 dijkstra 算法
12 vector<node> map2[MAXN];//用于 A_star 算法
13 void dijkstra()
14 {
15     int i,find[MAXN],v;
16     for(i=1;i<=n;i++)dis[i]=INF;
17     memset(find,0,sizeof(find));
18     priority_queue<node> heap;
19     dis[t]=0;
20     heap.push(node(t,0));
21     while(!heap.empty())
22     {
23         v=heap.top().v;
24         heap.pop();
25         if(find[v])continue;
26         find[v]=1;
27         for(i=0;i<map1[v].size();i++)
28             if(!find[map1[v][i].v] &&
                ↳ dis[v]+map1[v][i].c<dis[map1[v][i].v])
29             {
30
31                 ↳ dis[map1[v][i].v]=dis[v]+map1[v][i].c;
32
33                 ↳ heap.push(node(map1[v][i].v,dis[map1[v][i].v]));
34             }
35     }
36 }
37 int A_star()
38 {
39     int i,cnt[MAXN],v,g;
40     if(dis[s]==INF)return -1;
41     priority_queue<node> heap;
42     memset(cnt,0,sizeof(cnt));
```

```

41     heap.push(node(s,0)); //0 是 g(x)
42     while(!heap.empty())
43     {
44         v=heap.top().v;
45         g=heap.top().c;
46         heap.pop();
47         cnt[v]++;
48         if(cnt[t]==k)return g;
49         if(cnt[v]>k)continue;
50         for(i=0;i<map2[v].size();i++)
51             ↪ heap.push(node(map2[v][i].v,g+map2[v][i].c));
52     }
53     return -1;
54 }
55 int main()
56 {
57     int i,u,v,c;
58     cin>>n>>m;
59     for(i=0;i<m;i++)
60     {
61         cin>>u>>v>>c;
62         map2[u].push_back(node(v,c));
63         map1[v].push_back(node(u,c)); //反向储存求各节点
64             ↪ 到目标节点的最短距离
65     }
66     cin>>s>>t>>k;
67     if(s==t)k++;
68     dijkstra();
69     int ans=A_star();
70     cout<<ans<<endl;
71     return 0;
72 }

```


3.8 Second-MST

```
1  int a[maxn];
2  int cost[maxn][maxn];
3  int lowcost[maxn], fat[maxn], maxd[maxn][maxn];
4  bool vis[maxn];
5  int i, j, k, l, m, n, T, u, v, ans, mini;
6
7  int main()
8  {
9      scan(T);
10     while(T--)
11     {
12         memset(lowcost, inf, sizeof(lowcost));
13         memset(a, 0, sizeof(a));
14         memset(fat, 0, sizeof(fat));
15         memset(maxd, 0, sizeof(maxd));
16         memset(cost, inf, sizeof(cost));
17         memset(vis, 0, sizeof(vis));
18         scan2(n, m); ans = 0;
19         for(i = 1; i <= m; i++)
20         {
21             scan3(j, k, l);
22             cost[j][k] = cost[k][j] = l;
23         }
24         vis[1] = true; a[k = 1] = 1;
25         for(i = 2; i <= n; i++) { maxd[i][1] = maxd[1][i] = lowcost[i] = cost[1][i]; fat[i] = 1; }
26         for(i = 1; i <= n; i++) maxd[i][i] = cost[i][i] = 0;
27         for(u = 1, i = 1; i <= n - 1; i++)
28         {
29             mini = inf, v = -1;
30             for(j = 1; j <= n; j++)
31                 if(!vis[j] && lowcost[j] < mini)
32                     { mini = lowcost[j]; v = j; }
33             vis[v] = true;
34             ans += mini;
35             for(j = 1; j <= k; j++)
36                 maxd[a[j]][v] = maxd[v][a[j]] = max(mini, maxd[fat[v]][a[j]]);
37             a[++k] = v;
38             for(j = 1; j <= n; j++)
39                 if(!vis[j] && cost[v][j] < lowcost[j])
40                     { lowcost[j] = cost[v][j]; fat[j] = v; }
41         }
42         mini = inf;
43         for(i = 1; i <= n - 1; i++)
44             for(j = i + 1; j <= n; j++)
```

```

45         {
46             if(fat[i]==j || fat[j]==i || cost[i][j]==inf)continue;
47             mini=min(mini,cost[i][j]-maxd[i][j]);
48         }
49         if(mini==0)printf("Not Unique!\n");else printf("%d\n",ans);
50     }
51     return 0;
52 }

```

3.9 Stable Wedding

```
1 //对男性 (na) 最优
2 const int maxn=2005;
3
4 int na[maxn][maxn],nv[maxn][maxn];
5 queue<int> q;
6 int i,j,k,m,n,T;
7
8 int main()
9 {
10     scanf("%d",&T);
11     while(T-->0)
12     {
13         scanf("%d",&n);
14         memset(na,0,sizeof(na));
15         memset(nv,0,sizeof(nv));
16         for(i=1;i<=n;i++)
17             for(j=1;j<=n;j++)scanf("%d",&na[i][j]);
18         for(i=1;i<=n;i++)
19             for(j=1;j<=n;j++)
20             {
21                 scanf("%d",&m);
22                 nv[i][m]=j;
23             }
24         while(!q.empty())q.pop();
25         for(i=1;i<=n;i++)q.push(i);
26         while(!q.empty())
27         {
28             m=q.front();
29             q.pop();
30             k=na[m][++na[m][0]];
31             if(nv[k][0]==0)
32             {
33                 nv[k][0]=m;
34                 continue;
35             }else
36             {
37                 j=nv[k][0];
38                 if(nv[k][m]<nv[k][j])
39                 {
40                     q.push(j);
41                     nv[k][0]=m;
42                     continue;
43                 }else q.push(m);
44             }
```

```
45     }
46     for(i=1;i<=n;i++)na[nv[i][0]][0]=i;
47     for(i=1;i<=n;i++)printf("%d\n",na[i][0]);
48     if(T>0)printf("\n");
49 }
50 return 0;
51 }
```

4 Bipartite Graph Match

4.1 Bipartite Graph Match–BFS

```
1 struct Edge
2 {
3     int from;
4     int to;
5     int weight;
6
7     Edge(int f, int t, int w):from(f), to(t), weight(w) {}
8 };
9
10 vector<int> G[__maxNodes]; /* G[i] 存储顶点 i 出发的边的编号 */
11 vector<Edge> edges;
12 typedef vector<int>::iterator iterator_t;
13 int num_nodes;
14 int num_left;
15 int num_right;
16 int num_edges;
17
18 queue<int> Q;
19 int prev[__maxNodes];
20 int Hungarian()
21 {
22     int ans = 0;
23     memset(matching, -1, sizeof(matching));
24     memset(check, -1, sizeof(check));
25     for (int i=0; i<num_left; ++i) {
26         if (matching[i] == -1) {
27             while (!Q.empty()) Q.pop();
28             Q.push(i);
29             prev[i] = -1; // 设 i 为路径起点
30             bool flag = false; // 尚未找到增广路
31             while (!Q.empty() && !flag) {
32                 int u = Q.front();
33                 for (iterator_t ix = G[u].begin(); ix !=
34                     ↪ G[u].end() && !flag; ++ix) {
35                     int v = edges[*ix].to;
36                     if (check[v] != i) {
37                         check[v] = i;
38                         Q.push(matching[v]);
39                         if (matching[v] >= 0) { // 此点为匹配点
40                             prev[matching[v]] = u;
41                         } else { // 找到未匹配点, 交替路变为增广路
```

```

41         flag = true;
42         int d=u, e=v;
43         while (d != -1) {
44             int t = matching[d];
45             matching[d] = e;
46             matching[e] = d;
47             d = prev[d];
48             e = t;
49         }
50     }
51 }
52 }
53     Q.pop();
54 }
55     if (matching[i] != -1) ++ans;
56 }
57 }
58     return ans;
59 }
60
61 int main()
62 {
63
64     return 0;
65 }

```

4.2 Bipartite Graph Match-DFS

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  vector<int> g[1005];
6  int vis[1005];
7  int vv[1005],res[1005];
8  int i,j,k,l,m,n,T;
9
10 void dfs(int x)//染色区别二分图
11 {
12     for(unsigned int i=0;i<g[x].size();i++)
13     {
14         int v=g[x][i];
15         if(!vis[v])
16         {
17             vis[v]=-vis[x];dfs(v);
18             continue;
19         }else if(vis[v]==vis[x])return;
20     }
21 }
22
23 bool dfss(int x)
24 {
25     for(unsigned int i=0;i<g[x].size();i++)
26     {
27         int v=g[x][i];
28         if(vv[v])continue;
29         vv[v]=1;//如果当前结点已被搜索过 (剪枝)
30         if(res[v]==0||dfss(res[v]))//寻找增广路
31         {
32             res[v]=x;//res 表示与该点匹配的点编号
33             return true;
34         }
35     }
36     return false;
37 }
38
39 int main()//本程序默认二分图，非二分图会出错
40 {
41     memset(vis,0,sizeof(vis));
42     scanf("%d%d",&n,&m);
43     for(i=1;i<=n;i++)g[i].clear();
44     for(i=1;i<=m;i++)
```

```

45     {
46         scanf("%d%d",&j,&k);
47         g[j].push_back(k);
48         g[k].push_back(j);
49     }
50     for(i=1;i<=n;i++)
51         if(!vis[i]){vis[i]=1;dfs(i);} //先进行黑白染色，区分二分图
52     memset(res,0,sizeof(res));k=0;
53     for(i=1;i<=n;i++) //进行增广
54         if(vis[i]==1)
55             {
56                 memset(vv,0,sizeof(vv));
57                 if(dfss(i))k++;
58             }
59     printf("%d",k); //最大匹配数
60     return 0;
61 }

```


5 Flow

5.1 dinic

```
1  #include<iostream>
2  #include<cstdio>
3  #include<queue>
4  #define maxn 1005           //点数
5  #define maxm 80005         //边数
6  #define INF 0x3f3f3f3f
7  #define rever(x) (mem+((x-mem)^1))
8  using namespace std;
9  struct edge
10 {
11     int s,t,v,c;
12     edge* next;
13 }mem[maxn],*head[maxn],*prev[maxn];
14 queue<int> q;
15 int cnt=-1,n;
16 int dis[maxn];
17 int S,T;
18 void add_edge(int s,int t,int v,int c)
19 {
20     mem[++cnt].s=s;mem[cnt].t=t;mem[cnt].v=v;mem[cnt].c=c;mem[cnt].next=head[s];head[s]=mem+cnt;
21     mem[++cnt].s=t;mem[cnt].t=s;mem[cnt].v=0;mem[cnt].c=-
    ↪ c;mem[cnt].next=head[t];head[t]=mem+cnt;
22 }
23 bool bfs()
24 {
25     for (int i=0;i<=n;i++) dis[i]=INF;
26     q.push(S);dis[S]=0;
27     while(!q.empty())
28     {
29         for (edge *it=head[q.front()];it;it=it->next)
30             if (it->v&&dis[q.front()]+it->c<dis[it->t])
31             {
32                 dis[it->t]=dis[q.front()]+it->c;
33                 prev[it->t]=it;
34                 q.push(it->t);
35             }
36         q.pop();
37     }
38     return (dis[T]!=INF);
39 }
40 int cost=0;
```

```

41 int dinic()
42 {
43     int flow=0;
44     while(bfs())
45     {
46         int augflow=INF,tmpcost=0;
47         for (edge* it=prev[T];it;it=prev[it->s])
48         {
49             augflow=min(augflow,it->v);
50             tmpcost+=it->c;
51         }
52         for (edge* it=prev[T];it;it=prev[it->s])
53         {
54             it->v-=augflow;
55             rever(it)->v+=augflow;
56         }
57         flow+=augflow;cost+=augflow*tmpcost;
58     }
59     return flow;
60 }
61 int N,M,A,B,C;
62 int main()
63 {
64     scanf("%d%d",&N,&M);
65     S=0;T=N+1;n=T;
66     add_edge(S,1,2,0);add_edge(N,T,2,0);
67     for (int i=1;i<=M;i++)
68     {
69         scanf("%d%d%d",&A,&B,&C);
70         add_edge(A,B,1,C);
71         add_edge(B,A,1,C);
72     }
73     dinic();
74     printf("%d\n",cost);
75     return 0;
76 }

```

5.2 isap

```
1  #include<iostream>
2  #include<cstdio>
3  #include<queue>
4  #include<algorithm>
5  #include<cstring>
6  #include<cmath>
7  using namespace std;
8  #define maxn 205          //最大点数
9  #define maxm 205          //最大边数
10 #define rever(x) (mem+((x-mem)^1))
11 struct edge
12 {
13     int s,t,v;
14     edge* next;
15 }mem[maxm*2],*head[maxn];
16 int cnt=-1;
17 void add_edge(int s,int t,int v)
18 {
19     mem[++cnt].s=s;mem[cnt].t=t;mem[cnt].v=v;mem[cnt].next=head[s];head[s]=mem+cnt;
20     mem[++cnt].s=t;mem[cnt].t=s;mem[cnt].v=0;mem[cnt].next=head[t];head[t]=mem+cnt;
21 }
22 int n,m;
23
24 int S,T;
25 int numbs[maxn];
26 int d[maxn];
27 edge* cur[maxn],*revpath[maxn];
28
29 void bfs()
30 {
31     queue<int> q;
32     while(!q.empty()) q.pop();
33     for (int i=1;i<=n;i++) d[i]=maxn-1;          //由初始下标决定 01
34     d[T]=0;q.push(T);
35     while(!q.empty())
36     {
37         int u=q.front();
38         q.pop();
39         for (edge* it=head[u];it;it=it->next)
40         {
41             edge *now=rever(it);
42             if (now->v==0||d[now->s]<n) continue;
43             d[now->s]=d[u]+1;
44             q.push(now->s);
```

```

45     }
46 }
47 memset(nums,0,sizeof(nums));
48 for (int i=1;i<=n;i++) nums[d[i]]++; //由初始下标决定
    ↪ 01
49 }
50
51 int isap()
52 {
53     int flow=0;
54     for (int i=1;i<=n;i++) cur[i]=head[i]; //由初始下标决定 01
55     int u=S;
56     while(d[S]<n)
57     {
58         if (u==T)
59         {
60             int augflow=2147483647;
61             for (int i=S;i!=T;i=cur[i]->t)
62                 augflow=min(augflow,cur[i]->v);
63             for (int i=S;i!=T;i=cur[i]->t)
64             {
65                 cur[i]->v-=augflow;
66                 rever(cur[i])->v+=augflow;
67             }
68             flow+=augflow;u=S;
69         }
70         edge *e;
71         for (e=cur[u];e;e=e->next)
72             if (e->v&& d[u]==(d[e->t]+1)) break;
73         if (e)
74         {
75             cur[u]=e;
76             revpath[e->t]=rever(e);
77             u=e->t;
78         }
79         else
80         {
81             nums[d[u]]--;
82             if (nums[d[u]]==0) break;
83             cur[u]=head[u];
84             int mindist=n;
85             for (edge* it=head[u];it;it=it->next)
86                 if (it->v) mindist=min(mindist,d[it->t]);
87             d[u]=mindist+1;
88             nums[d[u]]++;
89             if (u!=S) u=revpath[u]->t;

```

```

90     }
91 }
92 return flow;
93 }
94 int main()
95 {
96     while(~scanf("%d%d",&m,&n))
97     {
98         cnt=-1;
99         memset(head,0,sizeof(head));
100         for (int i=1;i<=m;i++)
101         {
102             int s,e,c;
103             scanf("%d%d%d",&s,&e,&c);
104             add_edge(s,e,c);
105         }
106         S=1;T=n;n=n;
107         bfs();
108         printf("%d\n",isap());
109     }
110     return 0;
111 }

```

6 LCA

6.1 Double

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  struct tree
6  {
7      vector<int> son;
8      int fat,dep;
9  }a[100005];
10
11  map<string,int> my;
12  map<int,string> ym;
13
14  int i,j,k,l,m,n,T,o,md,sta;
15  queue<int> q;
16  bool vis[100005];
17  int ans[100005];
18  int lca[100005][20];
19
20  void bfs(int x)
21  {
22      while(!q.empty())q.pop();
23      q.push(x);
24      while(!q.empty())
25      {
26          int t=q.front();q.pop();
27          a[t].dep=a[a[t].fat].dep+1;
28          if(a[t].dep>md)md=a[t].dep;
29          for(unsigned int i=0;i<a[t].son.size();i++)
30              q.push(a[t].son[i]);
31      }
32  }
33
34  int find(int x,int y)
35  {
36      if(a[x].dep<a[y].dep)
37          {int temp=x;x=y;y=temp;}
38      while(a[x].dep>a[y].dep)
39      {
40          int j=0;
41          while(a[lca[x][j]].dep>a[y].dep)j++;
```

```

42     if(a[lca[x][j]].dep==a[y].dep){x=lca[x][j];break;}
43     x=lca[x][--j];
44 }
45 if(x==y)return y;
46 while(x!=y)
47 {
48     j=0;
49     while(lca[x][j]!=lca[y][j])j++;
50     if(j==0)break;j--;
51     x=lca[x][j];y=lca[y][j];
52 }
53 return a[x].fat;
54 }
55
56 int main()
57 {
58     memset(a,sizeof(a),0);
59     memset(vis,0,sizeof(vis));
60     cin>>n;
61     for(i=1;i<=100005;i++)a[i].son.clear();
62     my.clear();ym.clear();l=0;
63     for(i=1;i<=n;i++)
64     {
65         string s1,s2;
66         cin>>s1>>s2;
67         j=cl(s1);k=cl(s2);
68         a[k].fat=j;
69         a[j].son.push_back(k);
70         vis[k]=true;
71     }
72     for(i=1;i<=l;i++)if(!vis[i]){sta=i;break;}
73     memset(vis,0,sizeof(vis));
74     md=a[0].dep=a[0].fat=a[sta].fat=0;bfs(sta);
75     for(i=1;i<=l;i++)lca[i][0]=a[i].fat;
76     for(j=1;(1<<j)<=md;j++)
77         for(i=1;i<=l;i++)
78             lca[i][j]=lca[lca[i][j-1]][j-1];
79     T=0=0;
80     cin>>m;
81     for(i=1;i<=m;i++)
82     {
83         string s1,s2;
84         cin>>s1>>s2;
85         j=cl(s1);k=cl(s2);
86         cout<<ym[find(j,k)]<<endl;
87     }

```

```
88     return 0;  
89 }
```


6.2 ST

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  struct tree
6  {
7      vector<int> son;
8      int fat,dep;
9  }a[100005];
10
11  int i,j,k,l,m,n,T,o,md,sta;
12  bool vis[100005];
13  int ans[100005];
14  int st[100005][20];
15  int se[210000],no[100005],fir[100005];
16
17  void dfs(int x)
18  {
19      no[++md]=x;
20      int t=md;
21      se[++T]=t;
22      fir[x]=T;
23      for(unsigned int i=0;i<a[x].son.size();i++)
24      {
25          dfs(a[x].son[i]);
26          se[++T]=t;
27      }
28  }
29
30  void getST()
31  {
32      for(int i=1;i<=T;i++)st[i][0]=se[i];
33      for(int j=1;(1<<j)<=T;j++)
34          for(int i=1;i<=T-(1<<j)+1;i++)
35              st[i][j]=min(st[i][j-1],st[i+(1<<(j-1))][j-1]);
36  }
37
38  int find(int x,int y)
39  {
40      if(x>y){int temp=x;x=y;y=temp;}
41      int j=0;
42      while((1<<j)<=(y-x+1))j++;
43      j--;
44      return min(st[x][j],st[y-(1<<j)+1][j]);
```

```

45 }
46
47 int main()
48 {
49     memset(a,sizeof(a),0);
50     memset(vis,0,sizeof(vis));
51     cin>>n;md=T=0;
52     for(i=1;i<=100005;i++)a[i].son.clear();
53     my.clear();ym.clear();l=0;
54     for(i=1;i<=n;i++)
55     {
56         string s1,s2;
57         cin>>s1>>s2;
58         j=cl(s1);k=cl(s2);
59         a[k].fat=j;
60         a[j].son.push_back(k);
61         vis[k]=true;
62     }
63     for(i=1;i<=l;i++)if(!vis[i]){sta=i;break;}
64     memset(st,0,sizeof(st));
65     dfs(sta);cin>>m;
66     getST();
67     for(i=1;i<=m;i++)
68     {
69         string s1,s2;
70         cin>>s1>>s2;
71         j=cl(s1);k=cl(s2);
72         cout<<ym[no[find(fir[j],fir[k])]]<<endl;
73     }
74     return 0;
75 }

```

6.3 Tarjan

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  struct edge
6  {
7      int to,dist;
8  };
9
10 vector<edge> g[40005];
11 vector<edge> q[40005];
12
13 int i,j,k,l,m,n,T,o;
14 bool vis[40005];
15 int dis[40005];
16 int ans[205];
17 int bcj[40005];
18
19 int find(int x)
20 {
21     if(bcj[x]==x)return x;
22     else return bcj[x]=find(bcj[x]);
23 }
24
25 void dfs(int x)
26 {
27     vis[x]=true;
28     for(unsigned int i=0;i<q[x].size();i++)
29     {
30         edge r=q[x][i];
31         if(vis[r.to])
32         {
33             int zx=find(r.to);
34             ans[r.dist]=dis[x]+dis[r.to]-2*dis[zx];
35         }
36     }
37     for(unsigned int i=0;i<g[x].size();i++)
38     {
39         edge v=g[x][i];
40         if(!vis[v.to])
41         {
42             dis[v.to]=dis[x]+v.dist;
43             dfs(v.to);
44             bcj[v.to]=x;
```

```

45     }
46 }
47 }
48
49 int main()
50 {
51     scanf("%d",&T);
52     while(T--)
53     {
54         memset(ans,0,sizeof(ans));
55         memset(vis,0,sizeof(vis));
56         memset(dis,0,sizeof(dis));
57
58         scanf("%d%d",&n,&m);
59         for(i=1;i<=n;i++)q[i].clear();
60         for(i=1;i<=n;i++)bcj[i]=i;
61         for(i=1;i<=n;i++)g[i].clear();
62
63         for(i=1;i<=n-1;i++)
64         {
65             scanf("%d%d%d",&j,&k,&l);
66             edge r;
67             r.to=k;r.dist=l;
68             g[j].push_back(r);
69             r.to=j;r.dist=l;
70             g[k].push_back(r);
71         }
72
73         for(i=1;i<=m;i++)
74         {
75             scanf("%d%d",&j,&k);
76             edge r;
77             r.to=k;r.dist=i;
78             q[j].push_back(r);
79             r.to=j;r.dist=i;
80             q[k].push_back(r);
81         }
82         memset(vis,0,sizeof(vis));
83         dfs(1);
84         for(i=1;i<=m;i++)printf("%d\n",ans[i]);
85     }
86     return 0;
87 }

```

7 Tarjan

7.1 bridge cut-vertex

```
1  int dfs(int u,int fat)
2  {
3      int lowu,lowv;
4      lowu=pre[u]=++dfs_clock;
5      int child=0;
6      for(unsigned int i=0;i<g[u].size();i++)
7      {
8          int v=g[u][i];
9          if(!pre[v])
10         {
11             child++;
12             lowv=dfs(v,u);
13             lowu=min(lowu,lowv);
14             if(lowv>pre[u])p.push(edge(min(u,v),max(u,v)));
15             if(lowv>=pre[u])iscut[u]=true;
16         }else if(v!=fat) lowu=min(lowu,pre[v]);
17     }
18     if(fat==-1 && child<=1)iscut[u]=false;
19     return lowu;
20 }
21
22 void tarjan(int n)
23 {
24     dfs_clock=0;
25     memset(pre,0,sizeof(pre));
26     memset(iscut,0,sizeof(iscut));
27     for(int i=1;i<=n;i++)if(!pre[i])dfs(i,-1);
28 }
```

7.2 edge-double connected

1 //DFS 遍历不走桥即可

7.3 SCC

```
1 //有向图
2
3 int pre[maxn], low[maxn], a[maxn], sccno[maxn], tot[maxn];
4 int edge[100005][2];
5 int dfs_clock, scc_cnt, maxx;
6 vector<int> g[maxn];
7 stack<int> s;
8
9 int n, m, p0, p1, i, j, k, l;
10
11 void dfs(int u)
12 {
13     pre[u] = low[u] = ++dfs_clock;
14     s.push(u);
15     for(unsigned int i = 0; i < g[u].size(); i++)
16     {
17         int v = g[u][i];
18         if(!pre[v])
19         {
20             dfs(v);
21             low[u] = min(low[u], low[v]);
22         } else if(!sccno[v]) low[u] = min(low[u], pre[v]);
23     }
24     if(low[u] == pre[u])
25     {
26         scc_cnt++;
27         for(;;)
28         {
29             int x = s.top(); s.pop();
30             sccno[x] = scc_cnt;
31             if(x == u) break;
32         }
33     }
34 }
35
36 void find(int n)
37 {
38     dfs_clock = scc_cnt = 0;
39     memset(sccno, 0, sizeof(sccno));
40     memset(pre, 0, sizeof(pre));
41     memset(low, 0, sizeof(low));
42     while(!s.empty()) s.pop();
43     for(i = 1; i <= n; i++) if(!pre[i]) dfs(i);
44 }
```

7.4 vertex-double connected

```
1 struct edge
2 {
3     int p,q;
4     edge(int p=0,int q=0):p(p),q(q){}
5 }edg[maxm];
6
7 vector<int> g[maxn];
8 int bcc[maxm],pre[maxn];
9 int p[maxm],s[maxm];
10 int dfs_clock,bcc_cnt;
11
12 int dfs(int u,int fa)
13 {
14     int lowu=pre[u]=++dfs_clock;
15     for(unsigned int i=0;i<g[u].size();i++)
16     {
17         int side=g[u][i];
18         int v=other(side,u);
19         if(!pre[v])
20         {
21             s[++s[0]]=side;
22             int lowv=dfs(v,u);
23             lowu=min(lowu,lowv);
24             if(lowv>=pre[u])
25             {
26                 bcc_cnt++;
27                 for(;;)
28                 {
29                     int x=s[s[0]];s[0]--;
30                     bcc[x]=bcc_cnt;
31                     p[bcc_cnt]=min(p[bcc_cnt],x);
32                     if(x==side)break;
33                 }
34             }
35             }else if(pre[v]<pre[u] && v!=fa)
36             {
37                 s[++s[0]]=side;lowu=min(lowu,pre[v]);
38             }
39     }
40     return lowu;
41 }
42
43 void tarjan(int n)
44 {
```



```
45     dfs_clock=bcc_cnt=0;
46     memset(pre,0,sizeof(pre));
47     memset(bcc,0,sizeof(bcc));
48     memset(p,0x3f3f3f3f,sizeof(p));
49     s[0]=0;
50     for(int i=1;i<=n;i++)if(!pre[i])dfs(i,-1);
51 }
```

8 Math

8.1 FFT

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  const double eps=1e-10;
6  const double pi=3.1415926535897932384626433832795;
7  const double eln=2.718281828459045235360287471352;
8
9  const int maxn=105000;
10
11 complex<double> epsilon[maxn];
12 complex<double> arti_epsilon[maxn];
13 complex<double> a[maxn],b[maxn],c[maxn],temp[maxn];
14
15 int n1,n2,m;
16
17 void init_epsilon(int n)
18 {
19     for(int i=0;i!=n;i++)
20     {
21         epsilon[i]=complex<double>(cos(2.0*pi*i/n),sin(2.0*pi*i/n));
22         arti_epsilon[i]=conj(epsilon[i]);
23     }
24 }
25
26 int calc(int t)
27 {
28     int j=0;
29     while((1<<j)<=t)j++;
30     return 1<<j;
31 }
32
33
34 void DFT(int n,complex<double>* buffer,int offset,int
    ↪ step,complex<double>* epsilon)
35 {
36     if(n==1)return;
37     int m=n>>1;
38     DFT(m,buffer,offset,step<<1,epsilon);
39     DFT(m,buffer,offset+step,step<<1,epsilon);
40     for(int k=0;k!=m;k++)
```

```

41     {
42         int pos=2*step*k;
43         temp[k]=buffer[pos+offset]+epsilon[k*step]*buffer[pos+offset+step];
44         temp[k+m]=buffer[pos+offset]-
            ↪ epsilon[k*step]*buffer[pos+offset+step];
45     }
46     for(int i=0;i!=n;i++)buffer[i*step+offset]=temp[i];
47 }
48
49 //IDFT 将 DFT 的 epsilon 改为 arti_epsilon
50
51 void FFT(int m,complex<double>* a,complex<double>*
    ↪ b,complex<double>* c)
52 {
53     init_epsilon(m);
54     DFT(m,a,0,1,epsilon);
55     DFT(m,b,0,1,epsilon);
56     for(int i=0;i<=m;i++)c[i]=a[i]*b[i];
57     IDFT(m,c,0,1,epsilon);
58     double mm=m;
59     for(int i=0;i<=m;i++)c[i]/=mm;
60 }
61
62 int init()//n1,n2 表示多项式次数
63 {
64     double x,y;
65     scanf("%d%d",&n1,&n2);
66     memset(a,0,sizeof(a));
67     memset(b,0,sizeof(b));
68     for(int i=0;i<=n1;i++)
69     {
70         scanf("%lf %lf",&x,&y);
71         a[i].real(x);
72         a[i].imag(y);
73     }
74     for(int i=0;i<=n2;i++)
75     {
76         scanf("%lf %lf",&x,&y);
77         b[i].real(x);
78         b[i].imag(y);
79     }
80     m=calc(n1+n2);
81     return m;
82 }
83
84 void print()

```

```
85 {
86     for(int i=0;i<m;i++)printf("%lf %lf\n",real(c[i]),imag(c[i]));
87 }
88
89 int main()
90 {
91     m=init();
92     FFT(m,a,b,c);
93     print();
94 }
```

8.2 Integer High Accuracy

```
1  #include<cstdio>
2  #include<vector>
3  #include<cstring>
4  #include<algorithm>
5  using namespace std;
6  typedef unsigned int UInt;
7  typedef unsigned long long ULL;
8  class Number :public vector<UInt>
9  {
10     bool flag;
11     Number(UInt value){ flag = 0; if (value)push_back(value); }
12 public:
13     int cmp(const Number& num)const;
14     void add(const Number& num);
15     void sub(const Number& num);
16     Number mul(const Number& num)const;
17     Number div(const Number& num, Number& mod)const;
18     Number divInt(UInt num, UInt& mod)const;
19     void shr(UInt num);
20     void shl(UInt num);
21     void shr31(UInt num);
22     void shl31(UInt num);
23     void and(const Number& num);
24     void or(const Number& num);
25     Number not(UInt k)const;
26     void xor(const Number& num);
27     static ULL link(ULL x, UInt y){ return (x << 31) | y; }
28 public:
29     Number(){ flag = 0; }
30     Number(int value);
31     Number(long long value);
32     Number(const char *s);
33     Number(const char *s, UInt k);
34     void convert10(char *s);
35     void convert2k(char*s, UInt k);
36     bool operator < (const Number& num)const;
37     bool operator <= (const Number& num)const;
38     bool operator == (const Number& num)const;
39     Number operator + (const Number& num)const;
40     Number operator - (const Number& num)const;
41     Number operator * (const Number& num)const;
42     Number operator / (const Number& num)const;
43     Number operator % (const Number& num)const;
44     Number operator >> (UInt num)const;
```

```

45     Number operator << (UInt num) const;
46     Number operator & (const Number& num) const;
47     Number operator | (const Number& num) const;
48     Number operator ^ (const Number& num) const;
49 };
50 Number::Number(const char *s)
51 {
52     if (s[0] == '-') { flag = 1; s++; }
53     else flag = 0;
54     vector<char> str(strlen(s));
55     for (int i = str.size() - 1; i >= 0; i--)
56         str[i] = s[str.size() - i - 1] - '0';
57     while (str.size()) {
58         ULL sum = 0;
59         for (int i = str.size() - 1; i >= 0; i--) {
60             sum = sum * 10 + str[i];
61             str[i] = (char)(sum >> 31);
62             sum &= ~(1 << 31);
63         }
64         push_back((UInt)sum);
65         while (str.size() && !str.back()) str.pop_back();
66     }
67     if (!back()) pop_back();
68 }
69 Number::Number(const char *s, UInt k)
70 {
71     if (s[0] == '-') { flag = 1; s++; }
72     else flag = 0;
73     UInt cnt = 0;
74     ULL value = 0;
75     for (int i = strlen(s) - 1; i >= 0; i--) {
76         value |= (ULL)(s[i] <= '9' ? s[i] - '0' : s[i] - 'A' + 10) <<
77             cnt;
78         cnt += k;
79         if (cnt >= 31) {
80             push_back((UInt)value & ~(1 << 31));
81             value >>= 31; cnt -= 31;
82         }
83     }
84     if (value) push_back((UInt)value);
85 }
86 Number::Number(int value)
87 {
88     if (value) push_back(value > 0 ? value : -value);
89     flag = value < 0;
90 }

```

```

90  Number::Number(long long value)
91  {
92      flag = value < 0;
93      if (flag)value = -value;
94      while (value){
95          push_back(value & ~(1 << 31));
96          value >>= 31;
97      }
98  }
99  void Number::convert10(char *s)
100 {
101     if (flag)*s++ = '-';
102     vector<UInt> copy = *this;
103     UInt len = 0;
104     while (copy.size()){
105         ULL sum = 0;
106         for (int i = copy.size() - 1; i >= 0; i--){
107             sum = link(sum, copy[i]);
108             copy[i] = (UInt)(sum / 10);
109             sum %= 10;
110         }
111         s[len++] = (char)sum + '0';
112         if (!copy.back())copy.pop_back();
113     }
114     if (len == 0)s[len++] = '0';
115     reverse(s, s + len);
116     s[len] = 0;
117 }
118 void Number::convert2k(char*s, UInt k)
119 {
120     const char *table = "0123456789ABCDEF";
121     UInt len = 0, cnt = 0, bound = (1 << k) - 1;
122     ULL value = 0;
123     if (flag)*s++ = '-';
124     if (empty())s[len++] = '0';
125     else{
126         for (UInt i = 0;; cnt -= k){
127             if (cnt < k){
128                 if (i == size())break;
129                 value |= (ULL)(*this)[i++] << cnt;
130                 cnt += 31;
131             }
132             s[len++] = table[value & bound];
133             value >>= k;
134         }
135         s[len++] = table[value];

```

```

136     while (s[len - 1] == '0')len--;
137     reverse(s, s + len);
138 }
139 s[len] = 0;
140 }
141 int Number::cmp(const Number& num)const
142 {
143     if (size() != num.size())
144         return size() < num.size() ? -1 : 1;
145     for (int i = size() - 1; i >= 0; i--){
146         if ((*this)[i] != num[i])
147             return (*this)[i] < num[i] ? -1 : 1;
148     }
149     return 0;
150 }
151 bool Number::operator == (const Number& num)const{
152     return flag == num.flag && !cmp(num);
153 }
154 bool Number::operator < (const Number& num)const{
155     if (flag != num.flag)return flag;
156     return flag ? cmp(num) > 0 : cmp(num) < 0;
157 }
158 bool Number::operator <= (const Number& num)const{
159     if (flag != num.flag)return flag;
160     return flag ? cmp(num) >= 0 : cmp(num) <= 0;
161 }
162 //为提高效率确保 *this 位数 >=num 位数
163 void Number::add(const Number& num)
164 {
165     UInt f = 0, i = 0;
166     if (size() < num.size())resize(num.size());
167     for (; i < num.size(); i++){
168         (*this)[i] += num[i] + f;
169         f = (*this)[i] >> 31;
170         if (f)(*this)[i] ^= 1 << 31;
171     }
172     push_back(0);
173     for (; f; i++){
174         f = ++(*this)[i] >> 31;
175         if (f)(*this)[i] ^= 1 << 31;
176     }
177     if (!back())pop_back();
178 }
179 //确保 *this>=num
180 void Number::sub(const Number& num)

```



```

181 {
182     UInt f = 0, i = 0;
183     for (; i < num.size(); i++){
184         (*this)[i] -= num[i] + f;
185         f = (*this)[i] >> 31;
186         if (f)(*this)[i] ^= 1 << 31;
187     }
188     for (; f; i++){
189         f = --(*this)[i] >> 31;
190         if (f)(*this)[i] ^= 1 << 31;
191     }
192     while (size() && !back())pop_back();
193 }
194 Number Number::operator + (const Number& num)const
195 {
196     Number ret;
197     if (flag == num.flag){
198         if (size() < num.size()){ ret = num; ret.add(*this); }
199         else{ ret = *this; ret.add(num); }
200         ret.flag = flag;
201     }
202     else{
203         int t = cmp(num);
204         if (t < 0){
205             ret = num; ret.sub(*this);
206             ret.flag = num.flag;
207         }
208         else if (t > 0){
209             ret = *this; ret.sub(num);
210             ret.flag = flag;
211         }
212     }
213     return ret;
214 }
215 Number Number::operator - (const Number& num)const
216 {
217     Number ret;
218     if (flag != num.flag){
219         if (size() < num.size()){ ret = num; ret.add(*this); }
220         else{ ret = *this; ret.add(num); }
221         ret.flag = flag;
222     }
223     else{
224         int t = cmp(num);
225         if (t < 0){
226             ret = num; ret.sub(*this);

```

```

227         ret.flag = !flag;
228     }
229     else if (t > 0){
230         ret = *this; ret.sub(num);
231         ret.flag = flag;
232     }
233 }
234 return ret;
235 }
236 //为提高效率确保 *this 位数 >= num 位数
237 Number Number::mul(const Number& num)const
238 {
239     if (num.empty() || empty())return 0;
240     Number ret;
241     ret.resize(size() + num.size(), 0);
242     for (int i = num.size() - 1; i >= 0; i--){
243         ULL sum = 0;
244         for (UInt j = 0; j < size(); j++){
245             sum += (ULL)num[i] * (*this)[j];
246             ret[i + j] += sum & ~(1 << 31);
247             sum >>= 31;
248             if (ret[i + j] & (1 << 31)){
249                 sum++;
250                 ret[i + j] ^= 1 << 31;
251             }
252         }
253         ret[i + size()] += (UInt)sum;
254     }
255     for (UInt i = size(); i < ret.size(); i++){
256         if (ret[i] & (1 << 31)){
257             ret[i] ^= 1 << 31;
258             ret[i + 1]++;
259         }
260     }
261     if (!ret.back())ret.pop_back();
262     return ret;
263 }
264 Number Number::operator * (const Number& num)const
265 {
266     Number ret = size() < num.size() ? num.mul(*this) : mul(num);
267     if (ret.size())ret.flag = flag ^ num.flag;
268     return ret;
269 }
270 Number Number::div(const Number& num, Number& mod)const
271 {

```

```

272     const UInt aSize = size(), bSize = num.size();
273     if (aSize < bSize){ mod = *this; return 0; }
274     Number ret;
275     ret.resize(aSize - bSize + 1);
276     mod.assign(begin() + aSize - bSize + 1, end());
277     ULL y = num.back();
278     int bit = 0;
279     for (int i = 16; i; i >= 1){
280         if (y >> (bit + i))bit += i;
281     }
282     y = (y << (31 - bit)) + (num[bSize - 2] >> bit) + 1;
283     for (int i = ret.size() - 1; i >= 0; i--){
284         mod.shl31(1);
285         UInt oldSize = mod.size();
286         mod.resize(bSize + 1);
287         mod[0] = (*this)[i];
288         ULL x = link(mod[bSize], mod[bSize - 1]);
289         x = (x << (31 - bit)) | (mod[bSize - 2] >> bit);
290         if (!oldSize && mod[0])oldSize++;
291         mod.resize(oldSize);
292         if (ret[i] = (UInt)(x / y))mod.sub(num.mul(ret[i]));
293         if (mod.cmp(num) >= 0){
294             mod.sub(num);
295             ret[i]++;
296         }
297     }
298     if (!ret.back())ret.pop_back();
299     return ret;
300 }
301 Number Number::divInt(UInt num, UInt& mod)const
302 {
303     Number ret;
304     ret.resize(size());
305     ULL sum = 0;
306     for (int i = size() - 1; i >= 0; i--){
307         sum = link(sum, (*this)[i]);
308         ret[i] = (UInt)(sum / num);
309         sum %= num;
310     }
311     if (ret.size() && !ret.back())ret.pop_back();
312     mod = (UInt)sum;
313     return ret;
314 }
315 Number Number::operator / (const Number& num)const
316 {
317     UInt t;

```

```

318     Number ret = num.size() == 1 ? divInt(num[0], t) : div(num,
    ↪     Number());
319     if (ret.size())ret.flag = flag ^ num.flag;
320     return ret;
321 }
322 Number Number::operator % (const Number& num)const
323 {
324     Number ret;
325     if (num.size() == 1){
326         UInt t;
327         divInt(num[0], t);
328         ret = t;
329     }
330     else div(num, ret);
331     if (ret.size())ret.flag = flag;
332     return ret;
333 }
334 void Number::shr(UInt num)
335 {
336     if (!num)return;
337     UInt t = num / 31, k = num % 31;
338     if (size() <= t)clear();
339     else{
340         UInt newSize = size() - t;
341         for (UInt i = 0; i < newSize - 1; i++)
342             (*this)[i] = ((*this)[i + t] >> k) | ((*this)[i + t + 1] <<
    ↪             (31 - k)) & 0xffffffff;
343         (*this)[newSize - 1] = back() >> k;
344         resize(newSize);
345         if (!back())pop_back();
346     }
347 }
348 void Number::shr31(UInt num)
349 {
350     if (size() <= num)clear();
351     else{
352         UInt newSize = size() - num;
353         for (UInt i = 0; i < newSize; i++)
354             (*this)[i] = (*this)[i + num];
355         resize(newSize);
356     }
357 }
358 void Number::shl(UInt num)
359 {
360     if (empty() || !num)return;
361     UInt t = (num + 30) / 31, k = (num + 30) % 31 + 1;

```

```

362     UInt oldSize = size();
363     resize(oldSize + t);
364     for (int i = oldSize - 1; i >= 0; i--)
365         (*this)[i + t] = ((*this)[i + 1] << k) | ((*this)[i] >> (31 -
366             ↪ k)) & 0x7fffffff;
367     (*this)[t - 1] = (front() << k) & 0x7fffffff;
368     for (int i = t - 2; i >= 0; i--)
369         (*this)[i] = 0;
370     if (!back())pop_back();
371 }
372 void Number::shl31(UInt num)
373 {
374     if (empty())return;
375     UInt oldSize = size();
376     resize(oldSize + num);
377     for (int i = oldSize - 1; i >= 0; i--)
378         (*this)[i + num] = (*this)[i];
379     for (int i = num - 1; i >= 0; i--)
380         (*this)[i] = 0;
381 }
382 Number Number::operator >> (UInt num)const
383 {
384     bool f = false;
385     Number ret = *this;
386     if (flag){
387         UInt i, t;
388         for (i = 0; !(*this)[i]; i++);
389         t = i * 31;
390         f = t < num && ((t + 31 <= num) || ((1 << (num - t)) - 1) &
391             ↪ (*this)[i]);
392     }
393     ret.shr(num);
394     if (f)ret.add(1);
395     return ret;
396 }
397 Number Number::operator << (UInt num)const
398 {
399     Number ret = *this;
400     ret.shl(num);
401     return ret;
402 }
403 void Number::and(const Number& num)
404 {
405     if (size() > num.size())resize(num.size());
406     for (UInt i = 0; i < size(); i++)
407         (*this)[i] &= num[i];

```

```

406     while (size() && !back())pop_back();
407 }
408 //为提高效率确保 *this 位数 >=num 位数
409 void Number::or(const Number& num)
410 {
411     if (size() < num.size())resize(num.size());
412     for (UInt i = 0; i < num.size(); i++)
413         (*this)[i] |= num[i];
414 }
415 Number Number::not(UInt k)const
416 {
417     Number ret;
418     ret.resize(k, ~(1 << 31));
419     for (int i = min(size(), k) - 1; i >= 0; i--)
420         ret[i] = (*this)[i] ^ ~(1 << 31);
421     while (ret.size() && !ret.back())ret.pop_back();
422     return ret;
423 }
424 //为提高效率确保 *this 位数 >=num 位数
425 void Number::xor(const Number& num)
426 {
427     if (size() < num.size())resize(num.size());
428     for (UInt i = 0; i < num.size(); i++)
429         (*this)[i] ^= num[i];
430     while (size() && !back())pop_back();
431 }
432 Number Number::operator & (const Number& num)const
433 {
434     Number ret;
435     if (flag != num.flag){
436         if (flag){
437             ret = not(num.size()); ret.add(1);
438             ret.and(num);
439         }
440         else{
441             ret = num.not(size()); ret.add(1);
442             ret.and(*this);
443         }
444     }
445     else if (flag){
446         Number temp;
447         if (size() < num.size()){ ret = num; temp = *this; }
448         else{ ret = *this; temp = num; }
449         ret.sub(1); temp.sub(1);
450         ret.or(temp); ret.add(1);

```

```

451     }
452     else if (size() < num.size()){ ret = *this; ret.and(num); }
453     else{ ret = num; ret.and(*this); }
454     ret.flag = flag & num.flag;
455     return ret;
456 }
457 Number Number::operator | (const Number& num)const
458 {
459     Number ret;
460     if (flag != num.flag){
461         Number temp;
462         if (flag){ ret = num.not(size()); temp = *this; }
463         else{ ret = not(num.size()); temp = num; }
464         temp.sub(1);
465         ret.and(temp); ret.add(1);
466     }
467     else if (flag){
468         Number temp;
469         if (size() < num.size()){ ret = *this; temp = num; }
470         else{ ret = num; temp = *this; }
471         ret.sub(1); temp.sub(1);
472         ret.and(temp); ret.add(1);
473     }
474     else if (size() < num.size()){ ret = num; ret.or(*this); }
475     else{ ret = *this; ret.or(num); }
476     ret.flag = flag | num.flag;
477     return ret;
478 }
479 Number Number::operator ^ (const Number& num)const
480 {
481     Number ret;
482     if (flag != num.flag){
483         if (flag){
484             ret = *this; ret.sub(1);
485             ret.xor(num);
486         }
487         else{
488             ret = num; ret.sub(1);
489             ret.xor(*this);
490         }
491         ret.add(1);
492     }
493     else if(flag){
494         Number temp;
495         if (size() < num.size()){ ret = num; temp = *this; }
496         else{ ret = *this; temp = num; }

```

```
497         ret.sub(1); temp.sub(1);
498         ret.xor(temp);
499     }
500     else if (size() < num.size()){ ret = num; ret.xor(*this); }
501     else{ ret = *this; ret.xor(num); }
502     ret.flag = flag ^ num.flag;
503     return ret;
504 }
```


8.3 math

```
1  #include<bits/stdc++.h>
2
3  const int maxn=1005;
4  const double eps=1e-8;
5  #define LL long long int
6
7  using namespace std;
8
9  int phi[maxn];
10
11 void swap(double& p, double& q)
12 {
13     double t;
14     t=p;p=q;q=t;
15 }
16
17 struct Matrix
18 {
19     double a[maxn][maxn];
20     //1-n 行表示第 1-n 个方程
21     //每行第 1-n 个元素表示系数, 第 n+1 个元素表示等号右边的常数
22 }q;
23
24 int ii,jj,nn;
25
26 LL det(LL a[][maxn], int n) { //求行列式值 (整数版)
27     int i, j;
28     LL res = 1;
29     for (i = 0; i < n; i++) {
30         for (j = i + 1; j < n; j++) {
31             while (a[j][i]) {
32                 LL f = a[i][i] / a[j][i];
33                 for (int k = i; k < n; k++) a[i][k] -= f *
34                     ↪ a[j][k];
35                 for (int k = i; k < n; k++) swap(a[i][k],
36                     ↪ a[j][k]);
37                 res = -res;
38             }
39         }
40         if (a[i][i] == 0) return 0;
41         res *= a[i][i];
42     }
43     return res < 0 ? -res : res;
44 }
```

```

43
44 double FF(double x)//需积分的函数，自行修改
45 {
46     return 1.0;
47 }
48
49 double simpson(double x,double y)
50 {
51     double z=x+(y-x)/2.0;
52     return (y-x)/6.0*(FF(x)+FF(y)+4*FF(z));
53 }
54
55 double asr(double x,double y,double eeps,double A)//eeps 为精度
56 {
57     double z=x+(y-x)/2.0;
58     double L=simpson(x,z);
59     double R=simpson(z,y);
60     if(fabs(L+R-A)<=15*eeps)return (L+R)+(L+R-A)/15.0;
61     else return asr(x,z,eeps/2.0,L)+asr(z,y,eeps/2.0,R);
62 }
63
64 double simpson_zsx(double x,double y,double eeps)//自适应辛普森主函
    ↪ 数
65 {
66     return asr(x,y,eeps,simpson(x,y));
67 }
68
69 void gauss_eli(struct Matrix& p,int n)//高斯消元
70 {
71     int i,j,k,r;
72     for(i=1;i<=n;i++)
73     {
74         r=i;
75         for(j=i+1;j<=n;j++)
76             if(fabs(p.a[j][i])>fabs(p.a[r][i]))r=j;
77         if(r!=i)for(j=1;j<=n+1;j++)swap(p.a[r][j],p.a[i][j]);
78         for(k=1;k<=i-1;k++)
79         {
80             if(p.a[i][k]==0)continue;
81             for(j=n+1;j>=k;j--)
82                 p.a[i][j]-=p.a[k][j]/p.a[k][k]*p.a[i][k];
83         }
84     }
85     for(i=n;i>=1;i--)
86     {

```

```

87     for(j=i+1;j<=n;j++)
88         p.a[i][n+1]-=p.a[j][n+1]*p.a[i][j];
89     p.a[i][n+1]/=p.a[i][i];
90 }
91 }
92
93 LL gcd(LL a,LL b)
94 {
95     return b==0?a:gcd(b,a%b);
96 }
97
98 void tgcd(LL a,LL b,LL& d,LL& x,LL& y)//拓展欧几里德
99 {
100     if(!b){d=a;x=1;y=0;}
101     else{tgcd(b,a%b,d,y,x);y-=x*(a/b);}
102 }
103
104 LL pow_mod(LL a,LL p,LL n)//同余快速幂
105 {
106     if(p==0)return 1;
107     LL ans=pow_mod(a,p/2,n);
108     ans=(ans*ans)%n;
109     if(p%2==1)ans=(ans*a)%n;
110     return ans;
111 }
112
113 int euler_phi(int n)//求欧拉函数
114 {
115     int m=(int)sqrt(n+0.5);
116     int ans=n;
117     for(int i=2;i<=m;i++)
118         if(n%i==0)
119         {
120             ans=ans/i*(i-1);
121             while(n%i==0)n=n/i;
122         }
123     if(n>1)ans=ans/n*(n-1);
124     return ans;
125 }
126
127 void phi_table(int n)//欧拉函数表
128 {
129     memset(phi,0,n+1);
130     phi[1]=1;
131     for(int i=2;i<=n;i++)
132     {

```

```

133     if(phi[i])continue;
134     for(int j=i; j<=n; j+=i)
135     {
136         if(!phi[j])phi[j]=j;
137         phi[j]=phi[j]/i*(i-1);
138     }
139 }
140 }
141
142 LL inv(LL a,LL n)//a 关于 n 的逆元
143 {
144     LL d,x,y;
145     tgcd(a,n,d,x,y);
146     return d==1?(x+n)%n:-1;
147 }
148
149 // x mod m0=a0,x mod m =a,noSolution return 0
150 //初始可令 m0 = 1 ,a0 = 0
151 //布尔值返回是否有解
152 //m0,m 可以不互质
153 //若有多个方程, 做多次此剩余定理
154 //m0 , a0 返回答案
155 bool _china(LL &m0,LL &a0,LL m,LL a)
156 {
157     LL g,x,y;
158     LL c=abs(a-a0);
159     tgcd(m0,m,g,x,y);
160     if ( c % g ) return 0;
161     x*=(a-a0)/g;
162     x%=m/g;
163     a0=x*m0+a0;
164     m0*=m/g;
165     a0%=m0;
166     if(a0<0) a0+=m0;
167     return 1;
168 }
169
170 LL china(int n,int* a,int* m)//中国剩余定理
171 {
172     LL M=1,d,y,x=0;
173     for(int i=0;i<n;i++)M*=m[i];
174     for(int i=0;i<n;i++)
175     {
176         LL w=M/m[i];
177         tgcd(m[i],w,d,d,y);

```

```

178     x=(x+y*w*a[i])%M;
179 }
180 return (x+M)%M;
181 }
182
183 int log_mod(int a,int b,int n)//求解模方程  $a^x=b(mod\ n)$ ,  $n$  为素数,
    ↪ 无解返回-1
184 {
185     int m,v,e=1,i;
186     m=(int)sqrt(n+0.5);
187     v=inv(pow_mod(a,m,n),n);
188     map<int,int> x;
189     x[1]=0;
190     for(i=1;i<m;i++)
191     {
192         e=(e*a)%n;
193         if(!x.count(e))x[e]=i;
194     }
195     for(i=0;i<m;i++)
196     {
197         if(x.count(b))return (i*m+x[b]);
198         b=(b*v)%n;
199     }
200     return -1;
201 }

```

8.4 Matrix Fast Mi

```
1  struct mat
2  {
3      int n;
4      LL num[105][105];
5
6      void init0(int t)
7      {
8          n=t;
9          for(int i=0;i<=n;i++)
10             for(int j=0;j<=n;j++)
11                 num[i][j]=0;
12     }
13
14     void init1(int t)
15     {
16         n=t;
17         for(int i=0;i<=n;i++)
18             for(int j=0;j<=n;j++)
19                 if(i!=j)num[i][j]=0;else num[i][j]=1;
20     }
21
22     mat operator * (const struct mat p)const
23     {
24         struct mat ans;
25         ans.init0(n);
26         for(int i=1;i<=n;i++)
27             for(int j=1;j<=n;j++)
28                 for(int k=1;k<=n;k++)
29
30                     ↪ ans.num[i][j]=(ans.num[i][j]+num[i][k]*p.num[k][j])%mod;
31         //printf("??");ans.testprint();
32         return ans;
33     }
34
35     mat operator ^ (int t)const
36     {
37         struct mat ans,now;
38         ans.init1(n);
39         now.n=n;
40         for(int i=0;i<=n;i++)
41             for(int j=0;j<=n;j++)
42                 now.num[i][j]=num[i][j];
43         while(t>0)
44         {
```

```
44         if(t&1)ans=ans*now;
45         now=now*now;
46         t>>=1;
47     }
48     return ans;
49 }
50
51 };
```

8.5 NTT CRT

```
1  #include<cstdio>
2  #include<cmath>
3  #include<algorithm>
4  #include<vector>
5  #include<cstring>
6  using namespace std;
7  int len, bit;
8  int MOD, w[2][32];
9  inline int add(int a, int b){
10     return a + b - (a + b >= MOD ? MOD : 0);
11 }
12 inline int sub(int a, int b){
13     return a - b + (a - b < 0 ? MOD : 0);
14 }
15 inline int mul(int a, int b){
16     return (long long)a * b % MOD;
17 }
18 int power(int a, int b){
19     int ret = 1;
20     for (int t = a; b; b >>= 1){
21         if (b & 1)ret = mul(ret, t);
22         t = mul(t, t);
23     }
24     return ret;
25 }
26 int cal_root(int mod)
27 {
28     for (int i = 2;; i++){
29         if (power(i, (mod - 1) / 2) == mod - 1)
30             return i;
31     }
32 }
33 void fft_init(int n, int mod)
34 {
35     MOD = mod;
36     bit = (int)log2(n - 0.5) + 2;
37     len = 1 << bit;
38     w[0][0] = power(cal_root(mod), (mod - 1) / len);
39     int i;
40     for (i = 1; i < bit; i++){
41         w[0][i] = mul(w[0][i - 1], w[0][i - 1]);
42         i--;
43         w[1][i] = w[0][i];
44         for (i--; i >= 0; i--)
```



```

45     w[1][i] = mul(w[1][i + 1], w[0][i]);
46 }
47 void bitReverse(int a[]) {
48     for (int i = 1, j = len / 2; i < len - 1; i++) {
49         if (i < j) swap(a[i], a[j]);
50         int k = len / 2;
51         while (j >= k) { j -= k; k >>= 1; }
52         if (j < k) j += k;
53     }
54 }
55 void fft_main(int a[], bool reverse)
56 {
57     bitReverse(a);
58     for (int i = 1, s = 1; s < len; i++, s <<= 1){
59         int step = w[reverse][bit - i];
60         for (int j = 0; j < len; j += 2 * s){
61             int cur = 1;
62             for (int k = j; k < j + s; k++){
63                 int u = a[k], t = mul(cur, a[k + s]);
64                 a[k] = add(u, t);
65                 a[k + s] = sub(u, t);
66                 cur = mul(cur, step);
67             }
68         }
69     }
70     if (reverse){
71         int t = power(len, MOD - 2);
72         for (int i = 0; i < len; i++)
73             a[i] = mul(a[i], t);
74     }
75 }
76 //确保数组中的数小于 mod(mod<2^30), 数组需留足 2^(logn 向上取整 +1)
77 //    的空间, 后面填充 0
78 //并且 mod 为形如 m*2^k+1 的素数, 2^k>=2*n
79 void fft(int a[], int b[], int n, int mod)
80 {
81     fft_init(n, mod);
82     fft_main(a, 0); fft_main(b, 0);
83     for (int i = 0; i < len; i++)
84         a[i] = mul(a[i], b[i]);
85     fft_main(a, 1);
86 }
87 //确保 mod 两两互质, retmod 任意
88 void chineseRemainder(const int mod[], int *a[], int ret[], int
89     num, int n, int retMod)

```

```

88 {
89     int kk[30], mulMod[30][30], mulModr[30], mulretMod[30];
90     for (int i = 0; i < num; i++){
91         MOD = mod[i]; mulMod[i][0] = 1;
92         for (int j = 1; j <= i; j++)
93             mulMod[i][j] = mul(mulMod[i][j - 1], mod[j - 1]);
94         mulModr[i] = power(mulMod[i][i], MOD - 2);
95     }
96     mulretMod[0] = 1; MOD = retMod;
97     for (int i = 1; i < num; i++)
98         mulretMod[i] = mul(mulretMod[i - 1], mod[i - 1]);
99     for (int i = 0; i < n; i++){
100         for (int j = 1; j < num; j++){
101             MOD = mod[j];
102             int sum = a[0][i] % MOD;
103             for (int k = 1; k < j; k++)
104                 sum = add(sum, mul(mulMod[j][k], kk[k]));
105             kk[j] = mul(sub(a[j][i] % MOD, sum), mulModr[j]);
106         }
107         MOD = retMod;
108         ret[i] = a[0][i] % MOD;
109         for (int j = 1; j < num; j++)
110             ret[i] = add(ret[i], mul(kk[j] % MOD, mulretMod[j]));
111     }
112 }
113 //附满足条件大整数：167772161, 469762049, 754974721

```

9 Prime

9.1 Euler Prime

```
1  #include<stdio>
2  #define MAXN 10000001
3  int minFactor[MAXN];
4  int prime[2000000], primeNum;
5  int phi[MAXN];
6  void calPrime()
7  {
8      for (int i = 2; i < MAXN; i++){
9          if (!minFactor[i]){
10             prime[primeNum++] = i;
11             minFactor[i] = primeNum;
12         }
13         for (int j = 1; j <= minFactor[i]; j++){
14             int t = i * prime[j - 1];
15             if (t >= MAXN)break;
16             minFactor[t] = j;
17         }
18     }
19 }
20 void calPhi()
21 {
22     phi[1] = 1;
23     for (int i = 2; i < MAXN; i++){
24         if (!minFactor[i]){
25             prime[primeNum++] = i;
26             minFactor[i] = primeNum;
27             phi[i] = i - 1;
28         }
29         for (int j = 1;; j++){
30             int t = i * prime[j - 1];
31             if (t >= MAXN)break;
32             minFactor[t] = j;
33             if (j == minFactor[i]){
34                 phi[t] = phi[i] * prime[j - 1];
35                 break;
36             }
37             phi[t] = phi[i] * (prime[j - 1] - 1);
38         }
39     }
40 }
```

9.2 Miller-Rabin Pollard

```
1  #include<cstdio>
2  #include<typeinfo>
3  #include<cstdlib>
4  #include<algorithm>
5  using namespace std;
6  typedef unsigned long long ULL;
7  typedef unsigned int UInt;
8  const UInt base1[] = { 2, 7, 61, 0 };
9  const UInt base2[] = { 2, 325, 9375, 28178, 450775, 9780504,
    ↪ 1795265022, 0 };
10 const UInt prime[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
    ↪ 41, 43, 47, 53 };
11 template <typename T>
12 inline T add(T a, T b, T mod){
13     return a + b - (a + b >= mod ? mod : 0);
14 }
15 inline UInt mul(UInt a, UInt b, UInt mod){
16     return (ULL)a * b % mod;
17 }
18 ULL mul(ULL a, ULL b, ULL mod){
19     ULL ret = 0;
20     for (ULL t = a; b; b >>= 1){
21         if (b & 1)ret = add(ret, t, mod);
22         t <<= 1;
23         if (t >= mod)t -= mod;
24     }
25     return ret;
26 }
27 template <typename T>
28 T power(T a, T b, T mod){
29     T ret = 1;
30     for (T t = a; b; b >>= 1){
31         if (b & 1)ret = mul(ret, t, mod);
32         t = mul(t, t, mod);
33     }
34     return ret;
35 }
36 //n 为小于 2^63 的非 1 奇数, 正确性 100%
37 template <typename T>
38 bool millerRabin(T n)
39 {
40     int s = 0;
41     T r = n;
42     for (r--; !(r & 1); r >>= 1)s++;
```

```

43     for (const UInt *base = typeid(T) == typeid(UInt) ? base1 :
    ↪ base2; *base; base++){
44         T t = power(*base % n, r, n);
45         if (t == 0 || t == 1 || t == n - 1)continue;
46         for (int j = 1; j < s; j++){
47             t = mul(t, t, n);
48             if (t == 1)return false;
49             if (t == n - 1)break;
50         }
51         if (t != n - 1)return false;
52     }
53     return true;
54 }
55 template <typename T>
56 bool checkPrime(T n)
57 {
58     if (n == 1)return false;
59     for (int i = 0; i < sizeof(prime) / sizeof(int); i++){
60         if (n % prime[i] == 0)return n == prime[i];
61     }
62     return millerRabin(n);
63 }
64 template <typename T>
65 T gcd(T x, T y){
66     return y ? gcd(y, x % y) : x;
67 }
68 template <typename T>
69 T pollard(T n)
70 {
71     if (millerRabin(n))return n;
72     while (1){
73         T x = rand() % n, y = x, c = rand() % (n - 1) + 1;
74         for (UInt i = 1, j = 2;; i++){
75             if (i == j){ j *= 2; y = x; }
76             x = add(mul(x, x, n), c, n);
77             T d = gcd(x - y + n, n);
78             if (d != 1){
79                 if (d != n)return d;
80                 break;
81             }
82         }
83     }
84 }
85 ULL factor[64];
86 int factorNum;
87 void calFactorInternal(ULL n)

```

```

88 {
89     ULL d;
90     d = n >> 32 ? pollard(n) : pollard((UInt)n);
91     if (d == n){ factor[factorNum++] = d; return; }
92     calFactorInternal(d);
93     calFactorInternal(n / d);
94 }
95 void calFactor(ULL n)
96 {
97     factorNum = 0;
98     for (int i = 0; i < sizeof(prime) / sizeof(int); i++){
99         while (n % prime[i] == 0){
100             n /= prime[i];
101             factor[factorNum++] = prime[i];
102         }
103     }
104     if (n != 1)calFactorInternal(n);
105     sort(factor, factor + factorNum);
106 }

```

10 String

10.1 AC Automation

```
1  #include<cstdio>
2  #include<cstring>
3  #include<queue>
4  using namespace std;
5  #define LETTER 26
6  struct Trie{
7      int num, fail, match;
8      int next[LETTER];
9  }trie[500001];
10 int cnt;
11 void init(){
12     cnt = 1;
13     memset(trie, 0, 2 * sizeof(Trie));
14     trie[0].fail = 1;
15 }
16 inline int convert(char ch){ return ch - 'a'; }
17 void insert(char *s)
18 {
19     int cur = 0;
20     for (int i = 0; s[i]; i++){
21         int &pos = trie[cur].next[convert(s[i])];
22         if (!pos){
23             pos = ++cnt;
24             memset(&trie[cnt], 0, sizeof(Trie));
25         }
26         cur = pos;
27     }
28     trie[cur].num++;
29 }
30 void makeFail()
31 {
32     queue<int> q; q.push(0);
33     while (!q.empty()){
34         int t = q.front(); q.pop();
35         for (int i = 0; i < LETTER; i++){
36             int &cur = trie[t].next[i];
37             if (cur){
38                 q.push(cur);
39                 trie[cur].fail = trie[trie[t].fail].next[i];
40                 trie[cur].match = trie[cur].num ? cur :
41                     ↪ trie[trie[cur].fail].match;
```

```

41         }
42         else cur = trie[trie[t].fail].next[i];
43     }
44 }
45 }
46 int search(char *s)
47 {
48     int ret = 0, cur = 0;
49     for (int i = 0; s[i]; i++){
50         cur = trie[cur].next[convert(s[i])];
51         for (int temp = trie[cur].match; temp; temp =
52             ↪ trie[trie[temp].fail].match)
53             ret += trie[temp].num;
54     }
55     return ret;
56 }

```


10.2 AC-Auto(Compressed)

```
1  #define LETTER 26
2  struct Trie{
3      int num, next, fail;
4  }trie[1000000];
5  int cnt;
6  int pool[LETTER * 200000], poolEnd;
7  void init()
8  {
9      cnt = 0;
10     trie[0].num = 0;
11     trie[0].next = -1;
12     memset(pool, 0, sizeof(pool));
13     poolEnd = 0;
14 }
15 inline int convert(char ch){ return ch - 'a'; }
16 inline bool oneBranch(int value){ return value < LETTER; }
17 inline int child(int i, int ch){
18     if (oneBranch(trie[i].next))return trie[i].next == ch ? i + 1 :
19     ↪ 0;
20     return pool[trie[i].next + ch];
21 }
22 void insert(char *s)
23 {
24     int pos = 0, i;
25     for (i = 0; s[i]; i++){
26         int t = trie[pos].next;
27         if (oneBranch(t)){
28             if (t == convert(s[i]))pos++;
29             else{
30                 trie[pos].next = (poolEnd += LETTER);
31                 if (t != -1)pool[trie[pos].next + t] = pos + 1;
32                 break;
33             }
34         }
35         else if (pool[t + convert(s[i])])
36             pos = pool[t + convert(s[i])];
37         else break;
38     }
39     if (s[i]){
40         pool[trie[pos].next + convert(s[i])] = ++cnt;
41         for (i++; s[i]; i++, cnt++){
42             trie[cnt].num = 0;
43             trie[cnt].next = convert(s[i]);
44         }
45     }
```

```

44     trie[cnt].num = 1;
45     trie[cnt].next = -1;
46 }
47 else trie[pos].num++;
48 }
49 int getFailPoint(int father, int ch)
50 {
51     while (father){
52         father = trie[father].fail;
53         int pos = child(father, ch);
54         if (pos)return pos;
55     }
56     return 0;
57 }
58 void makeFail()
59 {
60     queue<int> q; q.push(0);
61     trie[0].fail = 0;
62     while (!q.empty()){
63         int t = q.front(); q.pop();
64         if (oneBranch(trie[t].next)){
65             if (trie[t].next != -1){
66                 trie[t + 1].fail = getFailPoint(t, trie[t].next);
67                 q.push(t + 1);
68             }
69         }
70         else for (int i = 0; i < LETTER; i++){
71             int cur = pool[trie[t].next + i];
72             if (cur){
73                 trie[cur].fail=getFailPoint(t, i);
74                 q.push(cur);
75             }
76         }
77     }
78 }
79 //统计匹配总次数，包括母串多次匹配同一模式串或多个模式串相同
80 int search(char *s)
81 {
82     int ret = 0, cur = 0;
83     for (int i = 0; s[i]; i++){
84         int ch = convert(s[i]);
85         for (; cur && !child(cur, ch); cur = trie[cur].fail);
86         cur = child(cur, ch);
87         for (int temp = cur; temp; temp = trie[temp].fail)
88             ret += trie[temp].num;

```

```
89     }  
90     return ret;  
91 }
```

10.3 KMP

```
1  int nxt[maxn];
2  char origin_string[maxn];
3  char target_string[maxn];
4  void get_nxt()
5  {
6      int n=strlen(target_string);
7      nxt[0]=0;nxt[1]=0;
8      for (int i=1;i<n;i++)
9      {
10         int j=nxt[i];
11         while(j&&target_string[i]!=target_string[j]) j=nxt[j];
12         nxt[i+1]=target_string[i]==target_string[j]?j+1:0;
13     }
14 }
15 int kmp()
16 {
17     int n=strlen(origin_string);
18     int m=strlen(target_string);
19     int j=0,cnt=0;
20     for (int i=0;i<n;i++)
21     {
22         while(j&&origin_string[i]!=target_string[j]) j=nxt[j];
23         if (origin_string[i]==target_string[j]) j++;
24         if (j==m) {cnt++;j=nxt[j];}
25     }
26     return cnt;
27 }
28 int main()
29 {
30     int _;
31     scanf("%d",&_);
32     while(--)
33     {
34         scanf("%s",target_string);
35         scanf("%s",origin_string);
36         get_nxt();
37         printf("%d\n",kmp());
38     }
39     return 0;
40 }
```

10.4 LCS

```
1  #include<algorithm>
2  using namespace std;
3  int maxCommonSubstring(char *s1, char *s2)
4  {
5      init();
6      suffixAutomation(s1);
7      int match = 0, ret = 0, cur = 0;
8      for (int i = 0; s2[i]; i++){
9          char c = convert(s2[i]);
10         if (!st[cur].next[c]){
11             while (cur != -1 && !st[cur].next[c])
12                 cur = st[cur].link;
13             if (cur == -1){ match = cur = 0; continue; }
14             match = st[cur].len;
15         }
16         cur = st[cur].next[c];
17         ret = max(ret, ++match);
18     }
19     return ret;
20 }
```

10.5 manacher

```
1  char s1[1000005],s2[2000010];
2  int p[2000010];
3  int i,j,k,l,m,n;
4  int mx,id;
5
6  int min(int x,int y)
7  {
8      return x<y?x:y;
9  }
10
11 int main()
12 {
13     gets(s1);l=0;
14     while(s1[0]!='E')
15     {
16         l++;
17         n=strlen(s1);
18         s2[0]='$';k=0;
19         for(i=0;i<n;i++)
20         {
21             s2[++k]='#';
22             s2[++k]=s1[i];
23         }
24         s2[++k]='#';s2[++k]='\0';
25         memset(p,0,sizeof(p));
26         mx=0;id=0;
27         for(i=1;s2[i]!='\0';i++)
28         {
29             p[i]=mx>i?min(p[2*id-i],mx-i):1;
30             while(s2[i+p[i]]==s2[i-p[i]])p[i]++;
31             if(i+p[i]>mx)
32             {
33                 mx=i+p[i];id=i;
34             }
35         }
36         mx=0;
37         for(i=1;s2[i]!='\0';i++)if(p[i]-1>mx)mx=p[i]-1;
38         printf("Case %d: %d\n",l,mx);
39         gets(s1);
40     }
41     return 0;
42 }
```

10.6 palindrome automation

```
1 struct node
2 {
3     int len,sum;
4     node* fail,*next[26];
5 }mem[100005],*headf,*heads,*last;
6 int tot,now;
7 char s[100005];
8
9 void init()
10 {
11     memset(mem,0,sizeof(mem));
12     headf=mem;last=heads=mem+1;
13     headf->fail=heads;heads->len=-1;
14     tot=1;now=0;
15 }
16 void add(int x,int p)
17 {
18     node* cur=last;
19     for (;s[p-cur->len-1]!=s[p];cur=cur->fail);
20     if (!cur->next[x])
21     {
22         node* ths=&mem[++tot];
23         last=cur->next[x]=ths;
24         ths->len=cur->len+2;
25         if (cur==heads) ths->fail=headf;
26         else
27         {
28             for (cur=cur->fail;s[p-cur->len-1]!=s[p];cur=cur->fail);
29             ths->fail=cur->next[x];
30         }
31         ths->sum=ths->fail->sum+1;
32     }
33     else last=cur->next[x];
34 }
35 //HDU 5157
36 long long l[100005],r[100005];
37 int main()
38 {
39     while(~scanf("%s",s))
40     {
41         int n=strlen(s);
42         init();
43         for (int i=0;i<n;i++) {add(s[i]-'a',i);l[i]=last->sum;}
44         reverse(s,s+n);
```

```

45     init();
46     for (int i=0;i<n;i++) {add(s[i]-'a',i);r[i]=last->sum+r[i-1];}
47     long long ans=0;
48     for (int i=0;i<n-1;i++) ans+=l[i]*r[n-i-2];
49     printf("%I64d\n",ans);
50 }
51 return 0;
52 }

```


10.7 SAM(Compressed)

```
1  #include<stdio>
2  #include<string>
3  using namespace std;
4  #define MAXN 1000001
5  #define LETTER 26
6  int pool[MAXN * LETTER / 3], poolNum;
7  struct State{
8      int len, link;
9      char ch[4];
10     int next[3];
11     int* child(char c){
12         if (ch[0] == -1)
13             return pool[next[0] + c] ? &pool[next[0] + c] : 0;
14         for (int i = ch[0]; i; i--){
15             if (ch[i] == c)return &next[i - 1];
16         }
17         return 0;
18     }
19     void insert(char c, int pos){
20         if (ch[0] == 3){
21             ch[0] = -1;
22             memset(pool + poolNum, 0, sizeof(int)*LETTER);
23             for (int i = 3; i; i--){
24                 pool[poolNum + ch[i]] = next[i - 1];
25             }
26             next[0] = poolNum;
27             poolNum += LETTER;
28             if (ch[0] == -1)pool[next[0] + c] = pos;
29             else{
30                 next[ch[0]] = pos;
31                 ch[++ch[0]] = c;
32             }
33         }
34     }st[MAXN * 2];
35     int cnt, last;
36     void init()
37     {
38         last = cnt = 0;
39         st[cnt].len = 0; st[cnt].link = -1;
40         st[cnt].ch[0] = 0;
41         poolNum = 0;
42     }
43     inline int convert(char ch){ return ch - 'a'; }
44     void add(char c)
```

```

45 {
46     c = convert(c);
47     int cur = ++cnt, i, *tmp;
48     st[cur].len = st[last].len + 1;
49     st[cur].ch[0] = 0;
50     for (i = last; i != -1 && !(tmp = st[i].child(c)); i =
        ↪ st[i].link)
51         st[i].insert(c, cur);
52     if (i == -1) st[cur].link = 0;
53     else{
54         int j = *tmp;
55         if (st[i].len + 1 == st[j].len)
56             st[cur].link = j;
57         else{
58             int copy = ++cnt;
59             st[copy].len = st[i].len + 1;
60             if (st[j].ch[0] == -1){
61                 st[copy].ch[0] = -1;
62                 st[copy].next[0] = poolNum;
63                 memcpy(pool + poolNum, pool + st[j].next[0],
        ↪ sizeof(int)*LETTER);
64                 poolNum += LETTER;
65             }
66             else{
67                 for (int i = 0; i < 4; i++)
68                     st[copy].ch[i] = st[j].ch[i];
69                 for (int i = st[j].ch[0]; i; i--)
70                     st[copy].next[i - 1] = st[j].next[i - 1];
71             }
72             st[copy].link = st[j].link;
73             for (; i != -1 && (tmp = st[i].child(c)) && *tmp == j; i =
        ↪ st[i].link)
74                 *tmp = copy;
75             st[j].link = st[cur].link = copy;
76         }
77     }
78     last = cur;
79 }
80 void suffixAutomation(char *s)
81 {
82     init();
83     for (int i = 0; s[i]; i++)
84         add(s[i]);
85 }

```

10.8 SAM

```
1  #include<stdio>
2  #include<string>
3  using namespace std;
4  #define MAXN 1000001
5  #define LETTER 26
6  struct State{
7      int len, link;
8      int next[LETTER];
9  }st[MAXN * 2];
10 int tree[MAXN];
11 int cnt, last;
12 void init()
13 {
14     last = cnt = 0;
15     st[cnt].len = 0; st[cnt].link = -1;
16     memset(st[0].next, 0, sizeof(st[0].next));
17 }
18 inline int convert(char ch){ return ch - 'a'; }
19 void add(char c)
20 {
21     c = convert(c);
22     int cur = ++cnt, i;
23     st[cur].len = st[last].len + 1;
24     memset(st[cur].next, 0, sizeof(st[cur].next));
25     for (i = last; i != -1 && !st[i].next[c]; i = st[i].link)
26         st[i].next[c] = cur;
27     if (i == -1)st[cur].link = 0;
28     else{
29         int j = st[i].next[c];
30         if (st[i].len + 1 == st[j].len)
31             st[cur].link = j;
32         else{
33             int copy = ++cnt;
34             st[copy].len = st[i].len + 1;
35             memcpy(st[copy].next, st[j].next, sizeof(st[j].next));
36             st[copy].link = st[j].link;
37             for (; i != -1 && st[i].next[c] == j; i = st[i].link)
38                 st[i].next[c] = copy;
39             st[j].link = st[cur].link = copy;
40         }
41     }
42     last = cur;
43 }
44 void suffixAutomation(char *s)
```

```

45 {
46     init();
47     for (int i = 0; s[i]; i++)
48         add(s[i]);
49 }
50 void suffixTree(char *s)
51 {
52     init();
53     for (int i = strlen(s) - 1; i >= 0; i--){
54         add(s[i]);
55         tree[i] = last;
56     }
57 }

```

10.9 Suffix Tree

```
1  #include<vector>
2  using namespace std;
3  int bucket[MAXN], order[MAXN];
4  int id[MAXN];
5  bool leave[MAXN];
6  vector<int> tree[MAXN];
7  void sortState()//?
8  {
9      int size = st[last].len;
10     memset(bucket, 0, sizeof(int)*(size + 1));
11     for (int i = 0; i <= cnt; i++)
12         bucket[st[i].len]++;
13     for (int i = 1; i <= size; i++)
14         bucket[i] += bucket[i - 1];
15     for (int i = cnt; i; i--)
16         order[--bucket[st[i].len]] = i;
17 }
18 void suffixTree(char *s)
19 {
20     int len = strlen(s);
21     init();
22     memset(leave, 0, len * sizeof(bool) * 2);
23     for (int i = len - 1; i >= 0; i--){
24         add(s[i]);
25         id[i] = last;
26         leave[last] = true;
27     }
28     for (int i = 0; i <= cnt; i++)
29         tree[i].clear();
30     for (int i = cnt; i; i--)
31         tree[st[i].link].push_back(i);
32 }
```

10.10 Trie(Compressed)

```
1  #include<stdio>
2  #include<string>
3  using namespace std;
4  #define LETTER 26
5  struct Trie{
6      int num, next;
7  }trie[1000000];
8  int cnt;
9  int pool[LETTER * 200000], poolEnd;
10 void init()
11 {
12     cnt = 0;
13     trie[0].num = 0;
14     trie[0].next = -1;
15     memset(pool, 0, sizeof(pool));
16     poolEnd = 0;
17 }
18 inline int convert(char ch){ return ch - 'a'; }
19 inline char convert2(int value){ return value + 'a'; }
20 inline bool oneBranch(int value){ return value < LETTER; }
21 inline int child(int i, int ch){
22     if (oneBranch(trie[i].next))return trie[i].next == ch ? i + 1 :
23     ↪ 0;
24     return pool[trie[i].next + ch];
25 }
26 void insert(char *s)
27 {
28     int pos = 0, i;
29     for (i = 0; s[i]; i++){
30         int t = trie[pos].next;
31         if (oneBranch(t)){
32             if (t == convert(s[i]))pos++;
33             else{
34                 trie[pos].next = (poolEnd += LETTER);
35                 if (t != -1)pool[trie[pos].next + t] = pos + 1;
36                 break;
37             }
38         }
39         else if (pool[t + convert(s[i])])
40             pos = pool[t + convert(s[i])];
41         else break;
42     }
43     if (s[i]){
44         pool[trie[pos].next + convert(s[i])] = ++cnt;
45     }
46 }
```

```

44     for (i++; s[i]; i++, cnt++){
45         trie[cnt].num = 0;
46         trie[cnt].next = convert(s[i]);
47     }
48     trie[cnt].num = 1;
49     trie[cnt].next = -1;
50 }
51 else trie[pos].num++;
52 }
53 int search(char* s)
54 {
55     int pos = 0;
56     for (int i = 0; s[i]; i++){
57         pos = child(pos, convert(s[i]));
58         if (!pos) return -1;
59     }
60     return trie[pos].num;
61 }
62 char temp[100];
63 void dfs(int i, int h)
64 {
65     if (trie[i].num){
66         temp[h] = 0;
67         printf("%s %d\n", temp, trie[i].num);
68     }
69     int t = trie[i].next;
70     if (oneBranch(t)){
71         if (t == -1) return;
72         temp[h] = convert2(t);
73         dfs(i + 1, h + 1);
74     }
75     else for (int j = 0; j < LETTER; j++){
76         if (pool[t + j]){
77             temp[h] = convert2(j);
78             dfs(pool[t + j], h + 1);
79         }
80     }
81 }

```

10.11 Trie

```
1  #include<stdio>
2  #include<string>
3  #define LETTER 26
4  struct Trie{
5      int num;
6      int next[LETTER];
7  }trie[500001];
8  int cnt;
9  void init(){
10     cnt = 0;
11     memset(trie, 0, sizeof(Trie));
12 }
13 inline int convert(char ch){ return ch - 'a'; }
14 inline char convert2(int value){ return value + 'a'; }
15 void insert(char *s)
16 {
17     int cur = 0;
18     for (int i = 0; s[i]; i++){
19         int &pos = trie[cur].next[convert(s[i])];
20         if (!pos){
21             pos = ++cnt;
22             memset(&trie[cnt], 0, sizeof(Trie));
23         }
24         cur = pos;
25     }
26     trie[cur].num++;
27 }
28 int search(char *s)
29 {
30     int cur = 0;
31     for (int i = 0; s[i]; i++){
32         cur = trie[cur].next[convert(s[i])];
33         if (!cur) return -1;
34     }
35     return trie[cur].num;
36 }
37 char temp[1001];
38 void dfs(int i, int h)
39 {
40     if (trie[i].num){
41         temp[h] = 0;
42         printf("%s %d\n", temp, trie[i].num);
43     }
44     for (int j = 0; j < LETTER; j++){
```



```
45     if (trie[i].next[j]){
46         temp[h] = convert2(j);
47         dfs(trie[i].next[j], h + 1);
48     }
49 }
50 }
```

11 Others

11.1 Attention

比赛：测试 PE 是否被判定为 WA，以及是否开了 O2 优化（详情咨询陈铮）

要点：

发票抬头写西安交通大学！！

留住所有发票！不能为收据，必须为机打发票！

去赛场报到时会拿到参赛费的发票，务必带回！

住宿不要超过三天

比赛不要乱吃！

香港：自行注意

请假条：

套模板之后交由徐宏喆老师或李玟老师签字，然后拿到计算机系办公室交由张华老师盖章，原件放在机房！

以及拿复印件请假。

11.2 operations

i 插入模式

esc 正常模式

:e xxx.cpp 编辑 xxx.cpp

:w 保存

:q 退出

:wq 保存并退出

ggVG 全选

y 复制

p 粘贴

u 撤销

"+y 复制到外部

"+p 从外部粘贴

正常模式下按 v 再按 i j k l 可选定指定区域

(vim 的剪切板和外部的剪切板貌似是分开的 ?)

(在 vimrc 正常工作的情况下)

F9 编译

F5 运行

(你们貌似都不用多屏 ?)

:split xxx.txt 横屏打开新文件 xxx.txt

:vsplit xxx.txt 竖屏打开新文件 xxx.txt

Ctrl+ww 移动到下一个窗口

11.3 Read

```
inline void read(int &x) {  
    char ch=getchar();  
    while(ch<'0' || ch>'9') ch=getchar();  
    x=0;  
    while(ch<='9' && ch>='0'){  
        x=x*10+ch-'0';  
        ch=getchar();  
    }  
}
```

11.4 Tags

| | | | | | |
|-----|----|-----|-----|-----|------|
| 二分 | 离线 | 倒跑 | 并查集 | DFS | BFS |
| 贪心 | | DP | 递推 | 莫队 | |
| 前缀和 | | 快速幂 | 倍增！ | | 差分数列 |

11.5 vimrc

```
map <F9> :! g++ % -o %< -g -lm -Wall && size %<.exe <CR>
map <F5> :! gdb %< <CR>
set cindent
set smartindent
set autoindent
set number
set ruler
set mouse=a
set bs=2
set tabstop=4
set softtabstop=4
set shiftwidth=4
set autoread
"set expandtab

syntax on
colo evening
```

11.6 Wrong

变量名打错！（数组开小）

忘了给变量或数组清零

程序逻辑错误（先 xx，再 xx）

n, m 打反