

# ACM/ICPC TEMPLATE

NKU -> HOT

October 20, 2012

## Contents

|       |                                   |    |
|-------|-----------------------------------|----|
| 1     | Introduction                      | 2  |
| 2     | Utility                           | 2  |
| 2.1   | Java Template                     | 2  |
| 2.2   | Java Multithread                  | 3  |
| 2.3   | Binary Search                     | 4  |
| 3     | Graph Theory                      | 4  |
| 3.1   | Prim - $O(N^2)$                   | 4  |
| 3.2   | Prim - $O(M \log N)$              | 5  |
| 3.3   | Kruskal - $O(M \log M)$           | 6  |
| 3.4   | Dijkstra - $O(N^2)$               | 8  |
| 3.5   | Dijkstra - $O(M \log N)$          | 9  |
| 3.6   | Dijkstra with heap                | 10 |
| 3.7   | Bellman-Ford                      | 12 |
| 3.8   | Shortest Path Faster Algorithm    | 13 |
| 3.9   | Network Flow - ISAP[NON-ORIGINAL] | 14 |
| 3.10  | Bipartite Graph Matching          | 16 |
| 3.11  | Minimum Cost Flow [TO BE TESTED!] | 17 |
| 3.12  | Kuhn-Munkras [NON-ORIGINAL]       | 18 |
| 3.13  | Cut Vertex and Edge               | 19 |
| 3.14  | Strongly Connected Components     | 20 |
| 4     | String Algorithm                  | 22 |
| 4.1   | ELF Hash                          | 22 |
| 4.2   | Aho-Corasick Automation           | 22 |
| 4.3   | Suffix array                      | 24 |
| 5     | Data Structure                    | 24 |
| 5.1   | Binary Indexed Tree               | 24 |
| 5.2   | Inversion                         | 25 |
| 5.3   | BigInt Multiply with FFT          | 26 |
| 6     | Computational Geometry            | 29 |
| 6.1   | Constants                         | 29 |
| 6.2   | Compare Function                  | 29 |
| 6.3   | 3D Point and Vector               | 29 |
| 6.4   | 3D Utility Functions              | 29 |
| 6.4.1 | 3D dot and 3D vector cross        | 29 |
| 6.4.2 | 3D line - line min distance       | 30 |
| 6.4.3 | Sphere coord                      | 30 |
| 6.5   | 2D Point and Vector               | 30 |
| 6.6   | 2D Utility Functions              | 30 |
| 6.6.1 | 2D dot and 2D scalar cross        | 30 |
| 6.6.2 | 2D triple point coline            | 31 |

|        |   |    |
|--------|---|----|
| 6.6.3  | 2D segment - segment overlap . . . . .            | 31 |
| 6.6.4  | 2D angle difference between two vectors . . . . . | 31 |
| 6.6.5  | 2D point - line distance . . . . .                | 31 |
| 6.6.6  | 2D point - circle tangent . . . . .               | 32 |
| 6.6.7  | 2D circle - circle tangent . . . . .              | 32 |
| 6.6.8  | 2D circle - polygon intersection area . . . . .   | 33 |
| 6.6.9  | 2D vector normal . . . . .                        | 34 |
| 6.6.10 | 2D vector rotation . . . . .                      | 34 |
| 6.7    | 2D Intersection . . . . .                         | 34 |
| 6.7.1  | ray - ray intersection . . . . .                  | 34 |
| 6.7.2  | line - circle intersection . . . . .              | 34 |
| 6.8    | Computational Geometry Topics . . . . .           | 35 |
| 6.8.1  | 2D convex hull . . . . .                          | 35 |
| 6.8.2  | 3D convec hull . . . . .                          | 35 |
| 6.8.3  | Max distance point pair . . . . .                 | 37 |
| 6.8.4  | closed point pair among two polygons . . . . .    | 37 |
| 6.8.5  | closed point pair . . . . .                       | 39 |
| 6.8.6  | closed circle pair . . . . .                      | 40 |
| 6.8.7  | circle hierarchy . . . . .                        | 42 |
| 7      | Math . . . . .                                    | 44 |
| 7.1    | Catalan Number . . . . .                          | 44 |

## 1 Introduction

NKU -> HOT ACM-ICPC template  
Thanks all past teammates and contributors!

## 2 Utility

### 2.1 Java Template

```

1 import java.util.*;
2 import java.io.*;
3
4 class Main {
5
6     void run() {
7         //Scanner in = new Scanner(System.in);
8         MyReader in = new MyReader();
9         String str;
10    }
11
12    public static void main(String args[]) {
13        new Main().run();
14    }
15
16    void debug(Object...x) {
17        System.out.println(Arrays.deepToString(x));
18    }
19 }
20
21 class MyReader {
22     BufferedReader br = new BufferedReader (
23         new InputStreamReader (System.in));
24     StringTokenizer in;
```

```

25     String next() {
26         try {
27             while (in == null || !in.hasMoreTokens()) {
28                 // Read a new line and split it into tokens
29                 in = new StringTokenizer(br.readLine());
30             }
31             // return next token
32             return in.nextToken();
33         } catch (Exception e) {
34             // EOF
35             return null;
36         }
37     }
38     // Transform the tokens into other types
39     int nextInt() {
40         return Integer.parseInt(next());
41     }
42 }

```

## 2.2 Java Multithread

BE CAREFUL: CALL START FOR EACH THREAD!

```

1  class Test extends Thread {
2      public static int ans;
3      public static int end;
4      public void run() {
5          int now = 0;
6          for (int i = 0; i < 400000000; i++) {
7              now = (now + i) % 9999997;
8          }
9          System.out.println(now);
10         new SubTask(0,0,100000000).start();
11         new SubTask(1,100000000,200000000).start();
12         new SubTask(2,200000000,300000000).start();
13         new SubTask(3,300000000,400000000).start();
14         for (;;) {
15             try {
16                 sleep(200);
17             } catch (Exception e) {}
18             if (end == 4) break;
19         }
20         System.out.println(ans);
21     }
22     public static void main(String[] args) {
23         new Test().start();
24     }
25 }
26
27 class SubTask extends Thread {
28     private int pos;
29     private int left;
30     private int right;
31     final static int mod = 9999997;
32
33     // init the input data
34     SubTask(int pos,int left,int right) {
35         this.pos = pos;

```

```

36         this.left = left;
37         this.right = right;
38     }
39
40     public void run() {
41         // solve the problem
42         int ans = 0;
43         for (int i = left; i < right; i++) {
44             ans = (ans + i) % mod;
45         }
46         // write the answer back
47         synchronized (this) {
48             Test.ans += ans;
49             Test.ans %= mod;
50             Test.end ++;
51         }
52     }
53 }

```

## 2.3 Binary Search

MAKE SURE check(x) is monotone in [L,R]

MAKE SURE check(L) == TRUE AND check(R) == FALSE FIRST!

```

1 while (l + 1 < r) {
2     int mid = (l + r) >> 1;
3     if (check(mid)) l = mid;
4     else r = mid;
5 }
6 return mid;

```

## 3 Graph Theory

### 3.1 Prim - $O(N^2)$

```

1 #include <iostream>
2 #include <cstdio>
3 using namespace std;
4
5 const int MAXN = 100;
6 const int EXP = 10;
7 const int INF = 1000000000;
8
9 int nn;
10 int map[MAXN+EXP][MAXN+EXP];
11
12 int sum;
13 bool inSet[MAXN+EXP];
14 int dist[MAXN+EXP];
15
16 void Prim(){
17     sum = 0;
18     for(int i = 1; i <= nn; i++) inSet[i] = 0, dist[i] = INF;
19     dist[1] = 0;
20     for(int i = 0; i < nn; i++){

```

```

21     int min = INF, idx = 0;
22     for(int j = 1; j <= nn; j++)
23         if(!inSet[j] && dist[j] < min)
24             min = dist[j], idx = j;
25     inSet[idx] = 1;
26     sum += min;
27     for(int j = 1; j <= nn; j++)
28         if(!inSet[j] && dist[j] > map[idx][j])
29             dist[j] = map[idx][j];
30 }
31 }
32
33 int main(){
34     while(scanf("%d\n",&nn) == 1 && nn){
35         for(int i = 1; i <= nn; i++)
36             for(int j = 1; j <= nn; j++)
37                 scanf("%d",&map[i][j]);
38         Prim();
39         printf("%d\n",sum);
40     }
41     return 0;
42 }

```

### 3.2 Prim- $O(M \log N)$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <queue>
4  using namespace std;
5
6  const int MAXN = 100;
7  const int MAXM = 10000;
8  const int EXP = 10;
9  const int INF = 1000000000;
10
11 int nn,mm;
12
13 int edges;
14 struct EDGE{
15     int n;
16     int v;
17     EDGE* nxt;
18 }pool[MAXM*2+EXP];
19 EDGE lnk[MAXN+EXP];
20
21 void addEdge(int _f, int _t, int _v){
22     pool[edges].n = _t;
23     pool[edges].v = _v;
24     pool[edges].nxt = lnk[_f].nxt;
25     lnk[_f].nxt = &pool[edges];
26     edges++;
27 }
28
29 struct NODE{
30     int n;
31     int dst;
32     NODE(int _n = 0, int _dst = 0){

```

```

33     n = _n;
34     dst = _dst;
35 }
36 };
37 bool operator <(NODE aa, NODE bb){
38     return aa.dst > bb.dst;
39 }
40
41 int sum;
42 bool inSet[MAXN+EXP];
43 int dist[MAXN+EXP];
44
45 void Prim_Prio(){
46     sum = 0;
47     for(int i = 1; i <= nn; i++) inSet[i] = 0, dist[i] = INF;
48     dist[1] = 0;
49     priority_queue <NODE> Q; Q.push(NODE(1,0));
50     while(Q.size()){
51         NODE now = Q.top(); Q.pop();
52         if(inSet[now.n]) continue;
53         inSet[now.n] = 1;
54         sum += now.dst;
55         for(EDGE* tmp = lnk[now.n].nxt; tmp; tmp = tmp->nxt){
56             if(!inSet[tmp->n] && tmp->v < dist[tmp->n]){
57                 dist[tmp->n] = tmp->v;
58                 Q.push(NODE(tmp->n,tmp->v));
59             }
60         }
61     }
62 }
63 }
64
65 int main(){
66     int cas; scanf("%d",&cas);
67     while(cas--){
68         scanf("%d%d", &nn, &mm);
69         edges = 0;
70         for(int i = 1; i <= nn; i++) lnk[i].nxt = 0;
71         for(int i = 1; i <= mm; i++){
72             int aa,bb,vv; scanf("%d%d%d", &aa, &bb, &vv);
73             addEdge(aa, bb, vv);
74         }
75         Prim_Prio();
76         printf("%d\n",sum);
77     }
78     return 0;
79 }

```

### 3.3 Kruskal -O(MlogM)

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 using namespace std;
5
6 const int MAXN = 100;
7 const int MAXM = 10000;

```

```

8  const int EXP = 10;
9  const int INF = 1000000000;
10
11  int nn,mm;
12
13  struct EDGE{
14      int f;
15      int t;
16      int v;
17  }pool[MAXM+EXP];
18
19  bool cmp(EDGE a, EDGE b){
20      return a.v < b.v;
21  }
22
23  int fa[MAXN+EXP];
24  int find(int x){
25      int r = x;
26      while(r != fa[r]) r = fa[r];
27      while(x != r){
28          int tmp = fa[x];
29          fa[x] = r;
30          x = tmp;
31      }
32      return r;
33  }
34
35  void uni(int aa, int bb){
36      int xx = find(aa);
37      int yy = find(bb);
38      if(xx != yy) fa[yy] = xx;
39  }
40
41  int sum;
42
43  void Kruskal(){
44      sum = 0;
45      sort(pool, pool+mm, cmp);
46      for(int i = 1; i <= nn; i++) fa[i] = i;
47      for(int i = 0; i < mm; i++){
48          int aa = find(pool[i].f);
49          int bb = find(pool[i].t);
50          if(aa == bb) continue;
51          sum += pool[i].v;
52          uni(aa, bb);
53      }
54  }
55
56
57  int main(){
58      int cas;   scanf("%d", &cas);
59      while(cas--){
60          scanf("%d%d", &nn, &mm);
61          for(int i = 0; i < mm; i++){
62              scanf("%d%d%d", &pool[i].f, &pool[i].t, &pool[i].v);
63          }
64          Kruskal();
65          printf("%d\n",sum);
66      }

```

```

66     return 0;
67 }

```

### 3.4 Dijkstra - $O(N^2)$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  using namespace std;
6
7  const int MAXN = 1000;
8  const int EXP = 10;
9  const int INF = 1000000000;
10
11 int nn;
12 int mm;
13
14 int map[MAXN][MAXN];
15
16 int dist[MAXN+EXP];
17 bool inSet[MAXN+EXP];
18
19 void init(){
20     for(int i = 0; i <= nn; i++){
21         for(int j = 0; j <= nn; j++){
22             map[i][j] = INF;
23         }
24     }
25
26 void Dijk(int s){
27     for(int i = 1; i <= nn; i++){
28         dist[i] = INF;
29         inSet[i] = 0;
30     }
31     dist[s] = 0;
32     for(int i = 1; i <= nn; i++){
33         int min = INF, idx = 0;
34         for(int j = 1; j <= nn; j++){
35             if(!inSet[j] && dist[j] < min){
36                 min = dist[j];
37                 idx = j;
38             }
39         }
40         inSet[idx] = 1;
41         for(int j = 1; j <= nn; j++){
42             if(!inSet[j] && dist[idx] + map[idx][j] < dist[j])
43                 dist[j] = dist[idx] + map[idx][j];
44         }
45     }
46 }
47
48 int main(){
49     int cas; scanf("%d", &cas);
50     while(cas--){
51         scanf("%d%d", &nn, &mm);
52         init();
53         for(int i = 1; i <= mm; i++){

```



```

53     int aa,bb,dd; scanf("%d%d%d", &aa, &bb, &dd);
54     if(map[aa][bb] > dd){
55         map[aa][bb] = map[bb][aa] = dd;
56     }
57 }
58 Dijk(1);
59 cout<<dist[nn]<<endl;
60 }
61 return 0;
62 }

```

### 3.5 Dijkstra - $O(M \log N)$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  using namespace std;
6
7  const int MAXN = 50000;
8  const int MAXM = 50000;
9  const int EXP = 10;
10 const int INF = 1000000000;
11
12 int edges;
13 struct EDGE{
14     int n;
15     int d;
16     EDGE *nxt;
17 }pool[MAXM*2+EXP];
18 EDGE lnk[MAXN+EXP];
19
20 void addEdge (int _f, int _t, int _d){
21     pool[edges].n = _t;
22     pool[edges].d= _d;
23     pool[edges].nxt = lnk[_f].nxt;
24     lnk[_f].nxt = &pool[edges];
25     edges++;
26 }
27
28 int nn;
29 int mm;
30
31 int dist[MAXN+EXP];
32 bool inSet[MAXN+EXP];
33
34 struct NODE{
35     int n;
36     int dst;
37     NODE(int _n = 0, int _dst = 0){
38         n = _n;
39         dst = _dst;
40     }
41 };
42
43 bool operator <(NODE aa, NODE bb){
44     return aa.dst > bb.dst;

```

```

45 }
46
47 void Dijk_Prio(int s){
48     for(int i = 1; i <= nn; i++){
49         dist[i] = INF;
50         inSet[i] = 0;
51     }
52     priority_queue <NODE> Q;
53     dist[s] = 0;
54     Q.push(NODE(s, dist[s]));
55     while(Q.size()){
56         NODE now = Q.top(); Q.pop();
57         if(inSet[now.n] == 1) continue;
58         inSet[now.n] = 1;
59         for(EDGE * tmp = lnk[now.n].nxt; tmp; tmp = tmp->nxt){
60             if(!inSet[tmp->n] && dist[now.n] + tmp->d < dist[tmp->n]){
61                 dist[tmp->n] = dist[now.n] + tmp->d;
62                 Q.push(NODE(tmp->n, dist[tmp->n]));
63             }
64         }
65     }
66 }
67
68 int main(){
69     int cas; scanf("%d", &cas);
70     while(cas--){
71         edges = 0;
72         scanf("%d%d", &nn, &mm);
73         for(int i = 1; i <= nn; i++) lnk[i].nxt = 0;
74         for(int i = 1; i <= mm; i++){
75             int aa,bb,dd; scanf("%d%d%d", &aa, &bb, &dd);
76             addEdge(aa, bb, dd);
77             addEdge(bb, aa, dd);
78         }
79         Dijk_Prio(1);
80         //cout<<dist[?]
81     }
82     return 0;
83 }

```

### 3.6 Dijkstra with heap

```

1  #include <cstdio>
2  #include <cstring>
3
4  using namespace std;
5
6  const int maxN=1010;
7  const int inf=2000000000;
8
9  class DJ_heap {
10 public:
11     int data[maxN];
12     int index[maxN];
13     int pos[maxN];
14     int tot;
15     void init (int n,int st) {

```

```

16     for (int i = 2; i <= n; i++) {
17         data[i] = inf;
18         int now = (i == st ? 1 : i);
19         index[i] = now;
20         pos[now] = i;
21     }
22     data[1] = 0;
23     index[1] = st;
24     pos[st] = 1;
25     tot = n;
26 }
27 void fix_down(int x) {
28     for (int son = x + x; son <= tot; x = son, son = x + x) {
29         if (son < tot && data[son+1] < data[son])
30             son++;
31         if (data[x] > data[son]) {
32             int tmp=data[x]; data[x]=data[son]; data[son]=tmp;
33             tmp=index[x]; index[x]=index[son]; index[son]=tmp;
34             pos[index[x]]=x;
35             pos[index[son]]=son;
36         }
37     }
38 }
39 void fix_up(int x) {
40     for (int fa = x>>1; x > 1; x = fa, fa = x>>1) {
41         if (data[fa] > data[x]) {
42             int tmp=data[fa]; data[fa]=data[x]; data[x]=tmp;
43             tmp=index[fa]; index[fa]=index[x]; index[x]=tmp;
44             pos[index[x]]=x;
45             pos[index[fa]]=fa;
46         }
47     }
48 }
49 void change(int x,int newdata) {
50     data[pos[x]]=newdata;
51     fix_up(pos[x]);
52 }
53 void pop(int &x,int &dist) {
54     x=index[1];
55     dist=data[1];
56     index[1]=index[tot];
57     data[1]=data[tot];
58     pos[x]=0;
59     pos[index[tot--]]=1;
60     fix_down(1);
61 }
62 bool empty() {
63     return tot==0;
64 }
65 };
66
67 int a[1010][2000];
68 int b[1010][2000];
69 int dist[1010];
70 bool visit[1010];
71
72 DJ_heap q;
73

```

```

74 int main() {
75     int n,m;
76     scanf("%d%d",&m,&n);
77     while (m--) {
78         int f,t,cost;
79         scanf("%d%d%d",&f,&t,&cost);
80         a[f][++a[f][0]]=t;
81         b[f][++b[f][0]]=cost;
82         a[t][++a[t][0]]=f;
83         b[t][++b[t][0]]=cost;
84     }
85     memset(dist,64,sizeof(dist));
86     q.init(n,n);
87     dist[n]=0;
88     while (!q.empty() && !visit[1]) {
89         int v,d;
90         q.pop(v,d);
91         for (int i=1;i<=a[v][0];i++)
92             if (!visit[a[v][i]] && dist[a[v][i]] > dist[v]+b[v][i]) {
93                 dist[a[v][i]] = dist[v] + b[v][i];
94                 q.change(a[v][i],dist[a[v][i]]);
95             }
96         visit[v]=1;
97     }
98     printf("%d\n",dist[1]);
99     return 0;
100 }

```

### 3.7 Bellman-Ford

```

1  #include <iostream>
2  #include <cstdio>
3
4  using namespace std;
5
6  const int MAXN = 1000;
7  const int MAXM = 2000;
8  const int EXP = 10;
9  const int INF = 1000000000;
10
11 int mm,nn;
12
13 int vf[MAXN+EXP],vt[MAXN+EXP],vc[MAXN+EXP]; 记录边 //
14
15 int dist[MAXN+EXP];
16
17 void init(){
18     scanf("%d%d",&nn,&mm);
19     for(int i = 0; i < mm; i++){
20         scanf("%d%d%d",vf+i,vt+i,vc+i);
21     }
22 }
23
24 void Bellman_Ford(int s){
25     for(int i = 1; i <= nn; i++) dist[i] = INF;
26     dist[s]=0;
27     for(int i = 0; i < nn-1; i++){

```

```

28     for(int i = 0; i < mm; i++){
29         if(dist[vf[i]] + vc[i] < dist[vt[i]]){
30             dist[vt[i]] = dist[vf[i]] + vc[i];
31         }
32         if(dist[vt[i]] + vc[i] < dist[vf[i]]){
33             dist[vf[i]] = dist[vt[i]] + vc[i];
34         }
35     }
36 }
37 }
38
39 int main(){
40     init();
41     Bellman_Ford(1);
42     printf("%d\n", dist[nn]);
43     return 0;
44 }

```

### 3.8 Shortest Path Faster Algorithm

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <queue>
5  using namespace std;
6
7  const int MAXN = 50000;
8  const int MAXM = 50000;
9  const int EXP = 10;
10 const int INF = 1000000000;
11
12 int edges;
13 struct EDGE{
14     int n;
15     int d;
16     EDGE *nxt;
17 }pool[MAXN*2+EXP];
18 EDGE lnk[MAXN+EXP];
19
20 void addEdge (int _f, int _t, int _d){
21     pool[edges].n = _t;
22     pool[edges].d= _d;
23     pool[edges].nxt = lnk[_f].nxt;
24     lnk[_f].nxt = &pool[edges];
25     edges++;
26 }
27
28 int nn;
29 int mm;
30
31 bool inQ[MAXN+EXP];
32 int dist[MAXN+EXP];
33
34 void spfa(int s){
35     for(int i = 0; i <= nn; i++){
36         inQ[i] = 0;
37         dist[i] = INF;

```

```

38     }
39     queue<int> Q; Q.push(s);
40     inQ[s] = 1; dist[s] = 0;
41     while(Q.size()){
42         int now = Q.front(); Q.pop();
43         inQ[now] = 0;
44         for(EDGE* tmp = lnk[now].nxt; tmp; tmp = tmp->nxt){
45             if(dist[now] + tmp->d < dist[tmp->n]){
46                 dist[tmp->n] = dist[now] + tmp->d;
47                 if(!inQ[tmp->n]) {
48                     Q.push(tmp->n);
49                     inQ[tmp->n] = 1;
50                 }
51             }
52         }
53     }
54 }
55
56 int main(){
57     int cas; scanf("%d", &cas);
58     while(cas--){
59         edges = 0;
60         scanf("%d%d", &nn, &mm);
61         for(int i = 1; i <= nn; i++) lnk[i].nxt = 0;
62         for(int i = 1; i <= mm; i++){
63             int aa,bb,dd; scanf("%d%d%d", &aa, &bb, &dd);
64             addEdge(aa, bb, dd);
65             addEdge(bb, aa, dd);
66         }
67         spfa(1);
68         //cout<<dist[?]
69     }
70     return 0;
71 }

```

### 3.9 Network Flow - ISAP[NON-ORIGINAL]

```

1  #include <cstring>
2  #include <cstdio>
3  #include <queue>
4  #include <algorithm>
5  #include <vector>
6
7  using namespace std;
8
9  const int MAXN = 210;
10 const int MAXM=500010;
11 const int inf = 2E9;
12
13 typedef struct {int v,next,val;} edge;
14 struct SAP {
15     edge e[MAXM];
16     int p[MAXN],eid;
17     inline void clear(){memset(p,-1,sizeof(p));eid=0;}
18     inline void insert1(int from,int to,int val) {
19         e[eid].v=to;
20         e[eid].val=val;

```

```

21     e[eid].next=p[from];
22     p[from]=eid++;
23     swap(from,to);
24     e[eid].v=to;
25     e[eid].val=0;
26     e[eid].next=p[from];
27     p[from]=eid++;
28 }
29 inline void insert2(int from,int to,int val) {
30     e[eid].v=to;
31     e[eid].val=val;
32     e[eid].next=p[from];
33     p[from]=eid++;
34     swap(from,to);
35     e[eid].v=to;
36     e[eid].val=val;
37     e[eid].next=p[from];
38     p[from]=eid++;
39 }
40 int n;
41 int h[MAXN];
42 int gap[MAXN];
43 int source,sink;
44 inline int dfs(int pos,int cost) {
45     if (pos==sink) {
46         return cost;
47     }
48     int j,minh=n-1,lv=cost,d;
49     for (j=p[pos];j!=-1;j=e[j].next) {
50         int v=e[j].v,val=e[j].val;
51         if (val>0) {
52             if (h[v]+1==h[pos]) {
53                 if (lv<e[j].val) d=lv;
54                 else d=e[j].val;
55                 d=dfs(v,d);
56                 e[j].val-=d;
57                 e[j^1].val+=d;
58                 lv-=d;
59                 if (h[source]>=n) return cost-lv;
60                 if (lv==0) break;
61             }
62             if (h[v]<minh) minh=h[v];
63         }
64     }
65     if (lv==cost) {
66         --gap[h[pos]];
67         if (gap[h[pos]]==0) h[source]=n;
68         h[pos]=minh+1;
69         ++gap[h[pos]];
70     }
71     return cost-lv;
72 }
73 int run() {
74     int ret=0;
75     memset(gap,0,sizeof(gap));
76     memset(h,0,sizeof(h));
77     gap[source]=n;
78     while (h[source]<n) ret+=dfs(source,inf);

```

```

79     return ret;
80 }
81 } solver;
82
83 int main() {
84     int N,M;
85     while (scanf("%d%d",&M,&N)!=EOF) {
86         solver.source = 1;
87         solver.sink = N;
88         solver.n = N;
89         solver.clear();
90         while (M--) {
91             int f,t,w;
92             scanf("%d%d%d",&f,&t,&w);
93             solver.insert1(f,t,w);
94         }
95         printf("%d\n",solver.run());
96     }
97     return 0;
98 }

```

### 3.10 Bipartite Graph Matching

```

1  #include <stdio>
2  #include <cstring>
3
4  bool adj[555][555];
5  bool visit[555];
6  int match[555];
7  int n;
8
9  bool dfs(int now) {
10     for (int i = 1; i <= n; i++) {
11         if (visit[i] == false && adj[now][i]) {
12             visit[i] = true;
13             int tt = match[i];
14             match[i] = now;
15             if (tt == -1 || dfs(tt)) return true;
16             match[i] = tt;
17         }
18     }
19     return false;
20 }
21
22 int main() {
23     int m;
24     scanf("%d%d",&n,&m);
25     for (int i = 0; i < m; i++) {
26         int f,t; scanf("%d%d",&f,&t);
27         adj[f][t] = true;
28     }
29     int ans = 0;
30     memset(match,0xff,sizeof(match));
31     for (int i = 1; i <= n; i++) {
32         memset(visit,0,sizeof(visit));
33         if (dfs(i)) ans ++;
34     }

```



```

35     printf("%d\n",ans);
36     return 0;
37 }

```

### 3.11 Minimun Cost Flow [TO BE TESTED!]

```

1  #include <iostream>
2  #include <queue>
3  #include <cstring>
4
5  using namespace std;
6
7  int n,m,ans,t,f;
8  int maxf[210][210],flow[210][210],cost[210][210];
9  int fa[210],dist[210];
10 bool inque[210];
11
12 inline int abs(int a) {return a > 0 ? a : -a ;}
13 void init() {
14     int a[210][2]={0},b[210][2]={0},s=0,sa=0,sb=0;
15     memset(maxf,0,sizeof(maxf));
16     memset(flow,0,sizeof(flow));
17     memset(cost,0,sizeof(cost));
18     for (int i=1;i<=n;i++){
19         for (int j=1;j<=m;j++) {
20             char tt;
21             cin>>tt;
22             if (tt=='H') {
23                 a[++sa][0]=i;
24                 a[sa][1]=j;
25             }
26             if (tt=='m') {
27                 b[++sb][0]=i;
28                 b[sb][1]=j;
29             }
30         }
31     }
32     s=sa;
33     for (int i = 1; i <= s; i++) {
34         for (int j = 1; j <= s; j++) {
35             cost[i][s+j] = abs(a[i][0]-b[j][0])+abs(a[i][1]-b[j][1]);
36             cost[s+j][i] = cost[i][s+j];
37             maxf[i][s+j] = 1;
38         }
39     }
40     for (int i = 1; i <= s; i++)
41         maxf[0][i]=maxf[s+i][s+s+1]=1;
42     n = t = s + s + 1;
43     f = 0;
44     ans = 0;
45 }
46
47 inline int value(int i,int j) {
48     return flow[j][i] > 0 ? -cost[i][j] : cost[i][j];
49 }
50
51 bool spfamark() {

```

```

52  memset(fa,0,sizeof(fa));
53  memset(inque,0,sizeof(inque));
54  memset(dist, 0x3f, sizeof(dist));
55  queue<int> q;
56  q.push(f); inque[f] = true; dist[f]=0;
57  while (!q.empty()) {
58      int now = q.front(); q.pop(); inque[now] = false;
59      for (int i = 0; i <= n;i++)
60          if ((maxf[now][i] - flow[now][i] > 0)
61              && dist[now] + value(now,i) < dist[i]){
62              dist[i] = dist[now] + value(now,i);
63              fa[i] = now;
64              if (!inque[i]) {
65                  inque[i]=1;
66                  q.push(i);
67              }
68          }
69      }
70      return dist[t] != 0x3f3f3f3f;
71 }
72
73
74 int main() {
75     while (cin >> n >> m && n && m) {
76         init();
77         while (spfamark()) {
78             for(int i = t; i != f; i = fa[i]) {
79                 ans+=value(fa[i],i);
80                 flow[fa[i]][i]++;
81                 flow[i][fa[i]]--;
82             }
83         }
84         cout << ans << endl;
85     }
86     return 0;
87 }

```

### 3.12 Kuhn-Munkras [NON-ORIGINAL]

refined from [http://blog.sina.com.cn/s/blog\\_6ec5c2d00100vt8d.html](http://blog.sina.com.cn/s/blog_6ec5c2d00100vt8d.html)

```

1  class KM_class {
2  private:
3      int match[maxm];
4      int lx[maxn];
5      int ly[maxm];
6      bool vis_x[maxn];
7      bool vis_y[maxm];
8      int slack;
9
10 public:
11     bool DFS(int u) {
12         vis_x[u] = true;
13         int tmp;
14         for(int v = 1; v <= M; v++) {
15             tmp = lx[u] + ly[v] - W[u][v];
16             if(tmp == 0) {
17                 if(!vis_y[v]) {

```

```

18         vis_y[v] = true;
19         if(match[v] == 0 || DFS(match[v]) ) {
20             match[v] = u;
21             return true;
22         }
23     }
24     } else {
25         slack = min(slack,tmp);
26     }
27 }
28 return false;
29 }
30
31 int KM() {
32     memset(match,0,sizeof(match));
33     memset(ly,0,sizeof(ly));
34     for(int u = 1; u <= N; u++) {
35         lx[u] = W[u][1];
36         for(int v = 2; v <= M; v++) {
37             lx[u] = max(lx[u],W[u][v]);
38         }
39     }
40
41     for(int u = 1; u <= N; u++) {
42         while(1) {
43             slack = INT_MAX;
44             memset(vis_x,0,sizeof(vis_x));
45             memset(vis_y,0,sizeof(vis_y));
46             if(DFS(u)) break;
47             for(int i = 1; i <= N; i++)
48                 if(vis_x[i])
49                     lx[i] -= slack;
50             for(int i = 1; i <= M; i++)
51                 if(vis_y[i])
52                     ly[i] += slack;
53         }
54     }
55     int sum = 0;
56     for(int v = 1; v <= M; v++) sum += W[match[v]][v];
57     return -sum;
58 }
59 } km;

```

### 3.13 Cut Vetrrix and Edge

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4
5 using namespace std;
6
7 bool a[110][110];
8 int visit[110];
9 int deep[110];
10 int back[110];
11 bool cut[110];
12 int n,ans;

```

```

13
14
15 void dfs(int k,int fa,int d) {
16     visit[k]=1;
17     back[k]=deep[k]=d;
18     int tot=0;
19     for (int i=1;i<=n;i++) {
20         if (a[k][i] && i!=fa && visit[i]==1)
21             back[k]=min(back[k],deep[i]);
22         if (a[k][i] && visit[i]==0) {
23             dfs(i,k,d+1);
24             tot++;
25             back[k]=min(back[k],back[i]);
26             if ((k==1 && tot>1) || (k!=1 && back[i]>=deep[k]))
27                 if (!cut[k]) {
28                     cut[k]=1;
29                     ans++;
30                 }
31             //if back[i]>deep[k] k,i is bridge;
32         }
33     }
34     visit[k]=2;
35 }
36
37 int main() {
38     while (1) {
39         scanf("%d",&n);
40         if (n==0)
41             break;
42         memset(a,0,sizeof(a));
43         memset(back,0,sizeof(back));
44         memset(cut,0,sizeof(cut));
45         memset(deep,0,sizeof(deep));
46         memset(visit,0,sizeof(visit));
47         ans=0;
48         int f;
49         while (scanf("%d",&f) && f>0) {
50             while (getchar()!=10) {
51                 int t;
52                 scanf("%d",&t);
53                 a[f][t]=a[t][f]=1;
54             }
55         }
56         dfs(1,0,0);
57         printf("%d\n",ans);
58     }
59     return 0;
60 }

```

### 3.14 Strongly Connected Components

```

1 #include <cstdio>
2 #include <cstring>
3 #include <stack>
4 #include <vector>
5
6 using namespace std;

```

```

7
8 vector<int> a[10010];
9 vector<int> b[10010];
10 stack<int> tt;
11 int fa[10010];
12 int d[10010];
13 int size[10010];
14 bool visit[10010];
15 int n,m;
16
17 void dfs(int k) {
18     visit[k]=1;
19     for (int i=0;i<a[k].size();i++)
20         if (!visit[a[k][i]])
21             dfs(a[k][i]);
22     tt.push(k);
23 }
24
25 void dfs(int k,int FA) {
26     fa[k]=FA;
27     size[FA]++;
28     visit[k]=1;
29     for (int i=0;i<b[k].size();i++)
30         if (!visit[b[k][i]])
31             dfs(b[k][i],FA);
32 }
33
34 int main() {
35     scanf("%d%d",&n,&m);
36     for (int i=0;i<m;i++) {
37         int f,t;
38         scanf("%d%d",&f,&t);
39         a[f].push_back(t);
40         b[t].push_back(f);
41     }
42     memset(visit,0,sizeof(visit));
43     for (int i=1;i<=n;i++)
44         if (!visit[i])
45             dfs(i);
46     memset(visit,0,sizeof(visit));
47     int s=0;
48     for (;!tt.empty();tt.pop())
49         if (!visit[tt.top()])
50             dfs(tt.top(),++s);
51     for (int i=1;i<=n;i++) {
52         int f=fa[i];
53         for (int j=0;j<b[i].size();j++) {
54             int t=fa[b[i][j]];
55             if (f!=t)
56                 d[t]++;
57         }
58     }
59     vector<int> ans;
60     for (int i=1;i<=s;i++)
61         if (d[i]==0)
62             ans.push_back(size[i]);
63     if (ans.size()==1)
64         printf("%d\n",ans[0]);

```

```

65     else
66         puts("0");
67     return 0;
68 }

```

## 4 String Algorithm

### 4.1 ELF Hash

```

1  int elfhash(char *key) {
2      unsigned int h = 0;
3      while(*key) {
4          h = (h << 4) + *key++;
5          unsigned int g=h&0Xf0000000L;
6          if (g) h ^= g >> 24;
7          h &= ~g;
8      }
9      return h%MOD;
10 }

```

### 4.2 Aho-Corasick Automation

```

1  #include <stdio>
2  #include <cstring>
3  #include <queue>
4
5  using std::queue;
6
7  void toInt(char s[]) {
8      for (int i = 0; s[i]; i++) {
9          if (s[i]=='A') s[i]='0';
10         if (s[i]=='G') s[i]='1';
11         if (s[i]=='T') s[i]='2';
12         if (s[i]=='C') s[i]='3';
13     }
14 }
15
16 struct trie{
17     trie *next[4];
18     trie *fail;
19     bool isend;
20 };
21
22 trie pool[1010];
23 trie *head;
24 trie *root;
25
26 void insert(char s[]) {
27     trie *now=root;
28     for (;;) {
29         if (s[0]==0) {
30             now->isend=1;
31             return;
32         }

```

```

33     int tt=s[0] - '0';
34     if (now->next[tt]==NULL)
35         now->next[tt]=++head;
36     now=now->next[tt];
37     s++;
38 }
39 }
40
41 void buildFaliure() {
42     queue<trie*> q;
43     for (int i=0;i<4;i++)
44         if (root->next[i]) {
45             root->next[i]->fail=root;
46             q.push(root->next[i]);
47         } else root->next[i]=root;
48     while (!q.empty()) {
49         trie *now=q.front(); q.pop();
50         for (int i=0;i<4;i++) {
51             trie *u=now->next[i];
52             if (u) {
53                 q.push(u);
54                 trie *v=now->fail;
55                 while (v->next[i]==NULL)
56                     v=v->fail;
57                 u->fail=v->next[i];
58             }
59         }
60         if (now->fail->isend) now->isend=1;
61     }
62 }
63
64 int dp[1010][1010];
65
66 trie* go(trie *now,char ch) {
67     ch -= '0';
68     trie *ans = now;
69     while (ans -> next[ch] == NULL)
70         ans = ans -> fail;
71     return ans -> next[ch];
72 }
73
74 int main() {
75     int ii=1;
76     for (;;) {
77         int n;
78         scanf("%d",&n);
79         if (n==0) break;
80         root=head=pool;
81         memset(dp,0x7f,sizeof(dp));
82         memset(pool,0,sizeof(pool));
83
84         static char buf[30];
85         for (int i=0;i<n;i++) {
86             scanf("%s",buf);
87             toInt(buf);
88             insert(buf);
89         }
90         buildFaliure();

```

```

91
92     static char word[1010];
93     scanf("%s",word);
94     toInt(word);
95     dp[0][0]=0;
96     n=head-pool;
97     int len;
98     for (len=0;word[len];len++) {
99         for (int j=0;j<n;j++)
100             if (dp[j][len]<=len) {
101                 for (char ch='0';ch<='3';ch++) {
102                     trie *tmp=go(pool+j,ch);
103                     if (tmp->isend) continue;
104                     int next=tmp-pool;
105                     int delta=0; if (ch!=word[len]) delta++;
106                     if (dp[next][len+1] > dp[j][len] + delta)
107                         dp[next][len+1] = dp[j][len] + delta;
108                 }
109             }
110     }
111     int ans=2000000000;
112     for (int j=0;j<n;j++)
113         if (dp[j][len]<ans) ans=dp[j][len];
114     if (ans>len) ans=-1;
115     printf("Case %d: %d\n",ii++,ans);
116 }
117 return 0;
118 }

```

### 4.3 Suffix array

```

1  int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
2  int cmp(int *r,int a,int b,int l)
3  {return r[a]==r[b]&&r[a+l]==r[b+l];}
4  void da(int *r,int *sa,int n,int m) {
5      int i,j,p,*x=wa,*y=wb,*t;
6      for(i=0;i<m;i++) ws[i]=0;
7      for(i=0;i<n;i++) ws[x[i]=r[i]]++;
8      for(i=1;i<m;i++) ws[i]+=ws[i-1];
9      for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
10     for(j=1,p=1;p<n;j*=2,m=p) {
11         for(p=0,i=n-j;i<n;i++) y[p++]=i;
12         for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
13         for(i=0;i<n;i++) wv[i]=x[y[i]];
14         for(i=0;i<m;i++) ws[i]=0;
15         for(i=0;i<n;i++) ws[wv[i]]++;
16         for(i=1;i<m;i++) ws[i]+=ws[i-1];
17         for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
18         for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
19             x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
20     }
21 }

```



## 5 Data Struct

### 5.1 Binary Indexed Tree

BECAREFUL WHILE I == 0 !!!

```
1 int sum(int k) {
2     int ans = 0;
3     for (int i = k; i > 0; i -= i & -i)
4         ans += a[i];
5     return ans;
6 }
7
8 void change(int k,int n,int delta) {
9     for (int i = k; i <= n; i += i & -i)
10        a[i] += delta;
11 }
```

### 5.2 Inversion

```
1 #include <cstdio>
2
3 int a[500010];
4 int t[500010];
5 long long ans;
6
7 void merge(int a[],int sizea,int b[],int sizeb) {
8     int nowa = 0;
9     int nowb = 0;
10    int s = 0;
11    while (nowa < sizea && nowb < sizeb) {
12        if (a[nowa]<=b[nowb])
13            t[s++]=a[nowa++];
14        else
15            if (a[nowa]>b[nowb]) {
16                t[s++] = b[nowb++];
17                ans += sizea - nowa;
18            }
19    }
20    while (nowa<sizea)
21        t[s++]=a[nowa++];
22    while (nowb<sizeb)
23        t[s++]=b[nowb++];
24 }
25
26 void sort(int a[],int size) {
27     if (size < 2)
28         return;
29     int lsize = size>>1;
30     int rsize = size-lsize;
31     sort(a, lsize);
32     sort(a + lsize, rsize);
33     merge(a, lsize, a+lsize, rsize);
34     for (int i = 0; i < size; i++)
35         a[i] = t[i];
36 }
37
```

```

38
39 int main() {
40     while (1) {
41         int n;
42         scanf("%d",&n);
43         if (!n)
44             break;
45         for (int i=0;i<n;i++)
46             scanf("%d",a+i);
47         ans = 0;
48         sort(a,n);
49         printf("%lld\n",ans);
50     }
51 }

```

### 5.3 BigInt Multiply with FFT

```

1  #include <stdio>
2  #include <cstring>
3  #include <cmath>
4
5  typedef long long Long;
6  const int MAXN=32768;
7  const double pi=acos(-1.0);
8  const Long MOD=100000;
9  const int TEN=5;
10
11 double ra[MAXN];
12 double ia[MAXN];
13 double rb[MAXN];
14 double ib[MAXN];
15 double rc[MAXN];
16 double ic[MAXN];
17 char a[MAXN];
18 char b[MAXN];
19 int slena;
20 int slenb;
21 int lena;
22 int lenb;
23 int n,logn;
24 Long ans[MAXN];
25
26 int rev(int x,int bit)
27 {
28     int ans=0;
29     for (int i=0;i<bit;i++)
30     {
31         ans<<=1;
32         if (x&1) ans|=1;
33         x>>=1;
34     }
35     return ans;
36 }
37
38 void fft(double ir[],double ii[],int size,int mark)
39 {
40     static double R[MAXN];

```

```

41     static double I[MAXN];
42     double delta=mark*2*pi;
43     for (int i=0;i<size;i++)
44     {
45         int tt=rev(i,logn);
46         R[tt]=ir[i];
47         I[tt]=ii[i];
48     }
49     for (int s=1;s<=logn;s++)
50     {
51         int m=1<<s;
52         double rwm=cos(delta/m);
53         double iwm=sin(delta/m);
54         for (int k=0;k<n;k+=m)
55         {
56             double rw=1;
57             double iw=0;
58             for (int j=0;j<m/2;j++)
59             {
60                 // t=w*A[k+j+m/2];
61                 double rt=rw*R[k+j+m/2]-iw*I[k+j+m/2];
62                 double it=rw*I[k+j+m/2]+iw*R[k+j+m/2];
63                 // u=A[k+j];
64                 double ru=R[k+j];
65                 double iu=I[k+j];
66
67                 // A[k+j]=u+t;
68                 R[k+j]=ru+rt;
69                 I[k+j]=iu+it;
70
71                 //A[k+j+m/2]=u-t;
72                 R[k+j+m/2]=ru-rt;
73                 I[k+j+m/2]=iu-it;
74
75                 double rnw=rw*rwm-iw*iwm;
76                 double inw=rw*iwm+iw*rwm;
77                 rw=rnw; iw=inw;
78             }
79         }
80     }
81     for (int i=0;i<size;i++)
82     {
83         ir[i]=R[i];
84         ii[i]=I[i];
85     }
86 }
87
88 double POW[10] = {1,1E1,1E2,1E3,1E4,1E5,1E6,1E7,1E8,1E9};
89
90 int next(char str[])
91 {
92     int len=0;
93     for (str[len]=getchar();str[len]!='0';str[len]=getchar())
94         len++;
95     str[len]=0;
96     return len;
97 }
98

```

```

99  int main()
100 {
101     int nn=0;
102     scanf("%d",&nn); getchar();
103     while (nn--)
104     {
105         memset(ra,0,n<<3);
106         memset(ia,0,n<<3);
107         memset(rb,0,n<<3);
108         memset(ib,0,n<<3);
109         memset(ans,0,n<<3);
110
111         slena=next(a);
112         int cnt=0; lena=0;
113         for (int j=slena-1;j>=0;j--)
114         {
115             ra[lena]=ra[lena]+(a[j]-'0')*POW[cnt++];
116             if (cnt==TEN) {len++; cnt=0;}
117         }
118         if (ra[len]>0.1) len++;
119
120         slenb=next(b);
121         cnt=0; lenb=0;
122         for (int j=slenb-1;j>=0;j--)
123         {
124             rb[lenb]=rb[lenb]+(b[j]-'0')*POW[cnt++];
125             if (cnt==TEN) {lenb++; cnt=0;}
126         }
127         if (rb[lenb]>0.1) lenb++;
128
129         n=1; logn=0;
130         while (n<lena || n<lenb) {n+=n;logn++;}
131         n+=n; logn++;
132
133         fft(ra,ia,n,1);
134         fft(rb,ib,n,1);
135         for (int i=0;i<n;i++)
136         {
137             rc[i]=ra[i]*rb[i]-ia[i]*ib[i];
138             ic[i]=ra[i]*ib[i]+rb[i]*ia[i];
139         }
140         fft(rc,ic,n,-1);
141         for (int i=0;i<n;i++)
142             ans[i]=(Long)(rc[i]/n+0.5);
143         for (int i=0;i<n-1;i++)
144         {
145             ans[i+1]+=ans[i]/MOD;
146             ans[i]%=MOD;
147         }
148         bool print=0;
149         for (int i=n-1;i>=0;i--)
150         {
151             if (!print && (ans[i]>0 || i==0))
152             {
153                 print=1;
154                 printf("%lld",ans[i]);
155             } else
156                 if (print)

```

```

157         printf("%05lld ",ans[i]);
158     }
159     putchar(10);
160 }
161 return 0;
162 }

```

## 6 Computational Geometry

### 6.1 Constants

```

1  const double eps=1e-12;
2  const double pi=acos(-1.0f);
3  const double INF=1.0/0.0f;
4  const double NEGINF=-1.0/0.0f;

```

### 6.2 Compare Function

```

1  inline bool equ0(const double& x){return fabs(x)<eps;}
2  inline bool equ(const double& x,const double& y){return fabs(x-y)<eps;
   ;}
3  inline bool ls(const double& x,const double& y){return x+eps<y;}
4  inline bool gr(const double& x,const double& y){return x-eps>y;}
5  inline bool greg(const double& x,const double& y){return x+eps>=y;}
6  inline bool lseq(const double& x,const double& y){return x-eps<=y;}

```

### 6.3 3D Point and Vector

```

1  struct Point{
2      double x,y,z;
3      explicit Point(const double& a=0,const double& b=0,const double& c
        =0):x(a),y(b),z(c){}
4      Point(const Point& p):x(p.x),y(p.y),z(p.z){}
5      Point operator-(const Point& p){return Point(-x,-y,-z);}
6      Point operator-(const Point& a) const{return Point(x-a.x,y-a.y,z-a.
        z);}
7      Point operator+(const Point& a) const{return Point(x+a.x,y+a.y,z+a.
        z);}
8      Point operator*(const double& s) const{return Point(s*x,s*y,s*z);}
9      double sqlen() const{return x*x+y*y+z*z;}
10     double len() const{return sqrt(sqlen());}
11     void norm(){double l=len();x/=l;y/=l;z/=l;}
12 };
13 typedef Point Vec;

```

### 6.4 3D Utility Functions

#### 6.4.1 3D dot and 3D vector cross

```

1 inline double dot(const Vec& u,const Vec& v)
2 {return u.x*v.x + u.y*v.y + u.z*v.z;}
3 inline Vec cross(const Vec& u,const Vec& v){
4     return Vec(u.y*v.z-u.z*v.y,u.z*v.x-u.x*v.z,u.x*v.y-u.y*v.x);
5 }

```

#### 6.4.2 3D line - line min distance

POJ 2852

```

1 inline bool line_mindist(const Point& p0,const Vec& d0,
2 const Point& p1,const Vec& d1,
3 double* t,double* k){
4     double a=dot(d0,d0),b=-dot(d0,d1),c=-dot(d1,d1),d=dot(d0,p1-p0),
5     e=dot(d1,p1-p0);
6     double det=a*c+b*b;
7     if(equ0(det)) return false;
8     *t=(d*c-b*e)/det;
9     *k=(a*e+b*d)/det;
10    return true;
11 }

```

#### 6.4.3 Sphere coord

```

1 inline double deg2grad(const double& d){return d*pi/180.0;}
2 inline Vec sphere_coord(const double& ele,const double& az){
3     double x,y,z,r;
4     z=sin(ele);
5     r=cos(ele);x=r*cos(az);y=r*sin(az);
6     return Vec(x,y,z);
7 }

```

### 6.5 2D Point and Vector

```

1 struct Point{
2     double x,y;
3     explicit Point(const double& a=0,const double& b=0):x(a),y(b){}
4     Point(const Point& p):x(p.x),y(p.y){}
5     Point& operator=(const Point& p){x=p.x;y=p.y;return *this;}
6     Point operator-(const Point& p) const{return Point(-x,-y);}
7     Point operator-(const Point& a) const{return Point(x-a.x,y-a.y);}
8     Point operator+(const Point& a) const{return Point(x+a.x,y+a.y);}
9     Point operator*(const double& s) const{return Point(s*x,s*y);}
10    double sqlen() const{return x*x+y*y;}
11    double len() const{return sqrt(sqlen());}
12    void norm(){double l=len();x/=l;y/=l;}
13 };
14 typedef Point Vec;

```

## 6.6 2D Utility Functions

### 6.6.1 2D dot and 2D scalar cross

```
1 inline double dot(const Vec& u,const Vec& v){return u.x*v.x + u.y*v.y  
   ;}  
2 inline double cross(const Vec& u,const Vec& v){return u.x*v.y - u.y*v  
   .x;}
```

### 6.6.2 2D triple point coline

```
1 inline bool coline(const Point& a,const Point& b,const Point& c){  
2     Vec u=b-a,v=c-a;  
3     return equ0(cross(u,v));  
4 }
```

### 6.6.3 2D segment - segment overlap

```
1 inline bool seg_overlap(const Point& a1,const Point& a2,const Point&  
   b1,const Point& b2){  
2     if(coline(a1,a2,b1) && coline(a1,a2,b2)){  
3         double k1,k2;  
4         Vec axis=a2-a1;  
5         Vec temp=b1-a1;  
6         if(equ0(axis.y)){  
7             k1=temp.x/axis.x;  
8             temp=b2-a1;  
9             k2=temp.x/axis.x;  
10        }  
11        else{  
12            k1=temp.y/axis.y;  
13            temp=b2-a1;  
14            k2=temp.y/axis.y;  
15        }  
16        if((gr(k1,1.0f) && gr(k2,1.0f)) || (ls(k1,0.0f) && ls(k2,0.0f)))  
            return false;  
17        else return true;  
18    }  
19    else return false;  
20 }
```

### 6.6.4 2D angle difference between two vectors

```
1 //CCW for positive  
2 inline double angle_diff(const Vec& from,const Vec& to){  
3     Vec nfrom=from;nfrom.norm();  
4     Vec nto=to;nto.norm();  
5     double d=dot(nfrom,nto),c=cross(nfrom,nto),a=acos(d);  
6     if(ls(c,0.0f)) return 2.0f*pi-a;else return a;  
7 }
```

### 6.6.5 2D point - line distance

```
1 inline double dist2line(const Point& p,const Point& o,const Vec& d){
2     Vec u=p-o,nd=d;nd.norm();
3     double proj=dot(u,nd);
4     return sqrt(u.sqrlen()-proj*proj);
5 }
```

### 6.6.6 2D point - circle tangent

POJ 1375

```
1 inline int point_circle_tan(const Point& p,const Point& C,const
2     double& r,Point* tps){
3     Vec v=C-p;double cdist=v.len();
4     if(!s(cdist,r)) return 0;
5     else if(equ(cdist,r)){tps[0]=p;return 1;}
6     double l=sqrt(cdist*cdist-r*r);v.norm();v=v*l;
7     double a=asin(r/cdist);
8     Vec u=vec_rotate(v,a);tps[0]=p+u;
9     u=vec_rotate(v,-a);tps[1]=p+u;
10    return 2;
};
```

### 6.6.7 2D circle - circle tangent

require circles no touch, no intersect  
POJ 2416

```
1 struct Seg{
2     Point p0,p1;
3     Seg(const Point& m=Point(),const Point& n=Point()):p0(m),p1(n){
4     }
5 };
6
7 inline void tan_2circle(const Point& C0,const double& R0,
8 const Point& C1,const double& R1,
9 Seg* segs){
10    Point c0=C0,c1=C1;double r0=R0,r1=R1;
11    if(gr(R1,R0)){swap(c0,c1);swap(r0,r1);}
12    Vec v=c1-c0;double cdist=v.len();
13
14    Point p=c0+v*(r0/(r1+r0));
15    Point t[2];
16    point_circle_tan(p,c0,r0,t);
17    Vec u=p-t[0];Point w;
18    w=t[0]+u*((r1+r0)/r0);
19    segs[0]=Seg(t[0],w);
20    u=p-t[1];w=t[1]+u*((r1+r0)/r0);
21    segs[1]=Seg(t[1],w);
22
23    if(equ(r0,r1)){
24        u=v;u.norm();u=u*r0;
25        Vec offset=vec_rotate(u,pi*0.5);
26        segs[2]=Seg(c0+offset,c1+offset);
27        offset=vec_rotate(u,-pi*0.5);
28        segs[3]=Seg(c0+offset,c1+offset);
29    }
```



```

29     }
30     else{
31         p=c0+v*(r0/(r0-r1));
32         Point k[2];
33         point_circle_tan(p,c1,r1,t);point_circle_tan(p,c0,r0,k);
34         segs[2]=Seg(t[0],k[0]);
35         segs[3]=Seg(t[1],k[1]);
36     }
37 }

```

## 6.6.8 2D circle - polygon intersection area

POJ 3675

schindlerlee, thanks

```

1 //CCW
2 inline double poly_area(const vector<Point>& p){
3     Point prev=p[0],cur;
4     double area=0;
5     for(int i=1;i<p.size();++i){
6         cur=p[i];
7         area+=0.5 * cross(prev,cur);
8         prev=cur;
9     }
10    return area;
11 }
12
13 double R;
14 Point C;
15 vector<Point> P;
16 int main(){
17     while(scanf("%lf",&R)==1){
18         P.clear();
19         int N;
20         scanf("%d",&N);
21         for(int i=0;i<N;++i){
22             double x,y;scanf("%lf%lf",&x,&y);
23             P.push_back(Point(x,y));
24         }
25         C=Point(0.0f,0.0f);
26         P.push_back(P[0]);
27         double area=0.0f;
28         double t,k;
29         for(int i=0;i<P.size()-1;++i){
30             Point s0=P[i],s1=P[i+1];
31             if(line_circle_intersect(s0,s1-s0,C,R,&t,&k)){
32                 Point p0=s0+(s1-s0)*t;
33                 Point p1=s0+(s1-s0)*k;
34                 if(lseq(k,0.0f) || greq(t,1.0f)){
35                     area+=0.5 * R*R * angle_diff(s0,s1);
36                 }
37                 else if(lseq(t,0.0f) && greq(k,1.0f)){
38                     area+=0.5 * cross(s0,s1);
39                 }
40                 else if(lseq(t,0.0f) && greq(k,0.0f) && lseq(k,1.0f)){
41                     area+=0.5 * cross(s0,p1);
42                     area+=0.5 * R*R * angle_diff(p1,s1);
43                 }

```

```

44     else if(greq(k,1.0f) && greq(t,0.0f) && lseq(t,1.0f)){
45         area+=0.5 * R*R * angle_diff(s0,p0);
46         area+=0.5 * cross(p0,s1);
47     }
48     else{
49         area+=0.5 * R*R * angle_diff(s0,p0);
50         area+=0.5 * cross(p0,p1);
51         area+=0.5 * R*R * angle_diff(p1,s1);
52     }
53 }
54 else area+=0.5f * R*R * angle_diff(s0,s1);
55 }
56 printf("%.2f\n",fabs(area));
57 }//while case
58 return 0;
59 }

```

### 6.6.9 2D vector normal

return two normalized vectors perp to the v  
(CCW,CW)

```

1  inline pair<Vec,Vec> vec_normal(const Vec& v){
2      Vec u=v;u.norm();
3      Vec CCW=Vec(-u.y,u.x),CW=Vec(u.y,-u.x);
4      return make_pair(CCW,CW);
5  }

```

### 6.6.10 2D vector rotation

CCW for positive angle

```

1  inline Vec vec_rotate(const Vec& v,const double& a){
2      double s=sin(a),c=cos(a);
3      return Vec(c*v.x-s*v.y,s*v.x+c*v.y);
4  }
5  inline Vec vec_rotate_left(const Vec& v){return Vec(-v.y,v.x);}
6  inline Vec vec_rotate_right(const Vec& v){return Vec(v.y,-v.x);}

```

## 6.7 2D Intersection

### 6.7.1 ray - ray intersection

```

1  inline bool ray_intersect(const Point& p,const Vec& u,
2  const Point& q,const Vec& v,
3  double* t,double* k){
4      double a=u.x,b=-v.x,c=u.y,d=-v.y;
5      double e=q.x-p.x,f=q.y-p.y;
6      double det=a*d-b*c;
7      if(equ0(det)) return false;
8      else{
9          *t=(e*d-b*f)/det;
10         *k=(a*f-e*c)/det;
11         return true;
12     }
13 }

```

### 6.7.2 line - circle intersection

```
1  inline bool line_circle_intersect(const Point& o,const Vec& d,  
2  const Point& C,const double& R,  
3  double* t,double* k){  
4      Vec delt=o-C;  
5      double a=d.sqlen(),b=2*dot(delt,d),c=delt.sqlen()-R*R;  
6      double x=b*b-4*a*c;  
7      if(!s(x,0.0f)) return false;  
8      x=sqrt(x);*t=(-b-x)/(2*a);*k=(-b+x)/(2*a);  
9      return true;  
10 }
```

## 6.8 Computational Geometry Topics

### 6.8.1 2D convex hull

$O(N \log N)$

monotone chain

vertex points only: modify the 'left' to strict, otherwise colinear points included.

CCW order

POJ 2187

```
1  const int MAXP=50000;  
2  Point P[MAXP];  
3  int N;  
4  
5  int H[MAXP],lower[MAXP];  
6  bool vis[MAXP];  
7  int top;  
8  int monotone_chain(){  
9      memset(vis,0,sizeof(vis));  
10     int utop=0,ltop=0;  
11     for(int i=N-1;i>=0;--i){  
12         if(utop<2) H[utop++]=i;  
13         else{  
14             while(utop>1 && cross(P[H[utop-1]]-P[H[utop-2]],P[i]-P[H[utop-1]]) <= 0) --utop;  
15             H[utop++]=i;  
16         }  
17     }  
18     for(int i=0;i<utop;++i) vis[H[i]]=true;  
19     for(int i=0;i<N;++i){  
20         if(ltop<2) lower[ltop++]=i;  
21         else{  
22             while(ltop>1 && cross(P[lower[ltop-1]]-P[lower[ltop-2]],P[i]-P[lower[ltop-1]]) <= 0) --ltop;  
23             lower[ltop++]=i;  
24         }  
25     }  
26     for(int i=0;i<ltop;++i) if(!vis[lower[i]]) H[utop++]=lower[i];  
27     return utop;  
28 }  
29 }
```

## 6.8.2 3D convec hull

O(N<sup>2</sup>)

incremental method

POJ 3528

schindlerlee, thanks.

```
1 //CCW
2 struct Face{
3     Point p0,p1,p2;
4     Face(const Point& a=Point(),const Point& b=Point(),const Point& c=
        Point()):
5         p0(a),p1(b),p2(c){
6     }
7 };
8 inline Vec face_normal(const Face& f){
9     Vec u=f.p1-f.p0;
10    Vec v=f.p2-f.p0;
11    Vec n=cross(u,v);
12    n.norm();
13    return n;
14 }
15 inline bool point_above_face(const Point& p,const Face& f){
16     Vec u=p-f.p0;
17     Vec n=face_normal(f);
18     return gr(dot(u,n),0.0f);
19 }
20
21
22 int edge[MAXP][MAXP];
23 int main(){
24     int N;
25     vector<Point> Points;
26     vector<Face> CH;
27     while( scanf("%d",&N)==1){
28         for( int i=0;i<N;++i){
29             double x,y,z;
30             scanf("%lf%lf%lf",&x,&y,&z);
31             Points.push_back( Point(x,y,z,i) );
32         }
33         //special case
34
35         CH.push_back( Face( Points[0], Points[1], Points[2] ) );
36         CH.push_back( Face( Points[2], Points[1], Points[0] ) );
37
38         for( int i=3;i<Points.size();++i){
39             Point cur=Points[i];
40             for( int j=0;j<CH.size();++j){
41                 Face f=CH[j];
42                 if( point_above_face( cur, f ) ){
43                     edge[f.p0.id][f.p1.id]=1;
44                     edge[f.p1.id][f.p2.id]=1;
45                     edge[f.p2.id][f.p0.id]=1;
46                 }
47                 else{
48                     edge[f.p0.id][f.p1.id]=-1;
49                     edge[f.p1.id][f.p2.id]=-1;
50                     edge[f.p2.id][f.p0.id]=-1;
51                 }
48             }
49         }
50     }
```

```

52     }
53     vector<Face> T;
54     for(int j=0;j<CH.size();j++){
55         Face f=CH[j];
56         if(edge[f.p0.id][f.p1.id]==1){
57             if(edge[f.p1.id][f.p0.id]==-1) T.push_back(Face(f.p0,f.p1,cur));
58             if(edge[f.p2.id][f.p1.id]==-1) T.push_back(Face(f.p1,f.p2,cur));
59             if(edge[f.p0.id][f.p2.id]==-1) T.push_back(Face(f.p2,f.p0,cur));
60         }
61         else{
62             T.push_back(f);
63         }
64     }
65     swap(CH,T);
66 }
67 double area=0.0f;
68 for(int i=0;i<CH.size();i++){
69     Face f=CH[i];
70     Vec crs=cross(f.p1-f.p0,f.p2-f.p0);
71     area+=0.5f * crs.len();
72 }
73 printf("%.3f\n",area);
74 }
75 return 0;
76 }

```

### 6.8.3 Max distance point pair

$O(N \log N)$

applied on the convex hull CCW ordered points

rotating calipers

no explicit angle calculation

POJ 2187; TJU 2847

```

1  inline int Next(int x){return (x+1)%top;}
2  int max_dist;
3  void rc(){
4      int s;
5      int p=0,q=Next(p);
6      while(area(P[H[p]],P[H[Next(p)]],P[H[Next(q)]]) >
7          area(P[H[p]],P[H[Next(p)]],P[H[q]])){
8          q=Next(q);
9      }
10     s=p;
11     do{
12         Vec d=P[H[p]]-P[H[q]];int temp=d.slen();
13         max_dist=max(max_dist,temp);
14         int np=p,nq=q;
15         if(area(P[H[p]],P[H[Next(p)]],P[H[Next(q)]]) >
16             area(P[H[p]],P[H[Next(p)]],P[H[q]])) nq=Next(q);
17         if(area(P[H[q]],P[H[Next(q)]],P[H[Next(p)]]) >
18             area(P[H[q]],P[H[Next(q)]],P[H[p]])) np=Next(p);
19         if(nq!=q) q=Next(q);
20         else p=Next(p);
21     } while(q!=Next(s));

```

22 }

#### 6.8.4 closed point pair among two polygons

$O(N \log N)$

rotating calipers

no explicit angle calculation

POJ 3608 utility functions

compare functor for monotone chain convex hull

```
1 struct comp{
2     bool operator()(const Point& a,const Point& b)const{
3         if(equ(a.x,b.x)) return ls(a.y,b.y);else return ls(a.x,b.x);
4     }
5 };
```

```
1 inline double area(const Point& a,const Point& b,const Point& c){return
    cross(b-a,c-a);}
2
3 const int MAXP=10000;
4 int M,N;
5 Point P[MAXP],Q[MAXP];
6
7 double mindist;
```

utility functions to step to the next vertex. used in rotating calipers

```
1 inline int nextp(int i){return (i+1)%M;}
2 inline int nextq(int i){return (i+1)%N;};
```

check if a point can perp project on a segment

```
1 inline bool point_over_seg(const Point& p,const Point& a,const Point& b
    ){
2     Vec u=b-a;double l=u.len();
3     u.norm();
4     double proj=dot(u,p-a);
5     double t=proj/l;
6     return ls(t,1.0f) && gr(t,0.0f);
7 }
```

check if two segments' perp projection overlap

```
1 inline bool seg_over_seg(const Point& a,const Point& b,
2                          const Point& c,const Point& d){
3     Vec u=b-a;
4     Vec r0=vec_rotate_left(u),r1=vec_rotate_left(u);
5     double t,k;
6     if(ray_intersect(a,r0,c,d-c,&t,&k)) if(ls(k,1) && gr(k,0)) return
7         true;
8     if(ray_intersect(b,r1,c,d-c,&t,&k)) if(ls(k,1) && gr(k,0)) return
9         true;
10    Vec v=d-c;
11    r0=vec_rotate_left(v);r1=vec_rotate_left(v);
12    if(ray_intersect(c,r0,a,b-a,&t,&k)) if(ls(k,1) && gr(k,0)) return
13        true;
14    if(ray_intersect(d,r1,a,b-a,&t,&k)) if(ls(k,1) && gr(k,0)) return
15        true;
16    return false;
```

13 }

return two parallel segments' distance

```
1 inline double dist_2seg(const Point& a,const Point& b,
2                       const Point& c,const Point& d){
3     return dist2line(c,a,b-a);
4 }
```

rotating calipers

```
1 void rc(){
2     int p,q;
3     int sp;
4     int cnt=0;
5     double f;
6     f=10001;
7     for(int i=0;i<M;++i) if(P[i].x<f){f=P[i].x;p=i;}
8     f=-10001;
9     for(int i=0;i<N;++i) if(Q[i].x>f){f=Q[i].x;q=i;}
10    while(gr(area(P[p],P[nextp(p)],Q[nextq(q)],area(P[p],P[nextp(p)],Q[q]
11        ))) q=nextq(q);
12    sp=p;
13    if(point_over_seg(Q[q],P[p],P[nextp(p)]))
14        mindist=min(mindist,dist2line(Q[q],P[p],P[nextp(p)]-P[p]));
15    else mindist=min(mindist,(P[p]-Q[q]).len());
16    do{
17        int np=p,nq=q;
18        if(gr(area(P[p],P[nextp(p)],Q[nextq(q)],area(P[p],P[nextp(p)],Q[q]
19            ))) nq=nextq(q);
20        if(gr(area(Q[q],Q[nextq(q)],P[nextp(p)],area(Q[q],Q[nextq(q)],P[p]
21            ))) np=nextp(p);
22        if(nq==q && np!=p){
23            if(point_over_seg(Q[q],P[p],P[nextp(p)]))
24                mindist=min(mindist,dist2line(Q[q],P[p],P[nextp(p)]-P[p]));
25            else mindist=min(mindist,(Q[q]-P[nextp(p)]).len());
26            p=nextp(p);
27        }
28        else if(np==p && nq!=q){
29            if(point_over_seg(P[p],Q[q],Q[nextq(q)]))
30                mindist=min(mindist,dist2line(P[p],Q[q],Q[nextq(q)]-Q[q]));
31            else mindist=min(mindist,(P[p]-Q[nextq(q)]).len());
32            q=nextq(q);
33        }
34        else if(np==p && nq==q){
35            if(seg_over_seg(P[p],P[nextp(p)],Q[q],Q[nextq(q)]))
36                mindist=min(mindist,dist_2seg(P[p],P[nextp(p)],Q[q],Q[nextq(q]
37                    )));
38            else{
39                //mindist=min(mindist,(P[nextp(p)]-Q[nextq(q)]).len());
40                mindist=min(mindist,(P[nextp(p)]-Q[q]).len());
41                //mindist=min(mindist,(Q[nextq(q)]-P[p]).len());
42            }
43            p=nextp(p);
44        }
45        if(p==sp) ++cnt;
46    }while(!(cnt>1&&p==nextp(sp)));
47 }
```

### 6.8.5 closed point pair

$O(N \log N)$

sweep line

POJ 3714; ZOJ 2107

compx, compy, compare functors used in sorted point sequence and BST respectively

```
1 const int MAXN=200002;
2
3 Point P[MAXN];
4 int N;
5 struct compx{
6     bool operator()(const Point& a,const Point& b)const{
7         if(a.x==b.x) return ls(a.y,b.y);
8         else return ls(a.x,b.x);
9     }
10 };
11 struct compy{
12     bool operator()(const Point& a,const Point& b)const{
13         if(a.y==b.y) return ls(a.x,b.x);
14         else return ls(a.y,b.y);
15     }
16 };
```

sweep line algorithm

```
1 double mindist;
2
3 Point eventp;
4 typedef set<Point,compy> Box;
5 typedef Box::iterator BoxIter;
6 void sweepline(){
7     sort(P,P+2*N,compx());
8     mindist=INF;
9     int l=0;
10    Box box;
11    for(int i=0;i<2*N;++i){
12        eventp=P[i];
13        double x=eventp.x-mindist,negy=eventp.y-mindist,psy=eventp.y+
            mindist;
14
15        while(l<i && ls(P[l].x,x)){
16            box.erase(P[l]);
17            ++l;
18        }
19
20        BoxIter a=box.lower_bound(Point(x,negy));
21        BoxIter b=box.lower_bound(Point(x,psy));
22        for(BoxIter iter=a;iter!=b;iter++){
23            Vec d=*iter - eventp;
24            if(iter->color != eventp.color) mindist=min(mindist,d.len());
25        }
26        box.insert(eventp);
27    }
28 }
```



### 6.8.6 closed circle pair

$O(N \log N)$

sweep line

HDU 3124

data structure and compare functors

```
1 struct Circle{
2     Point c;double r;
3     Circle(const Point& a=Point(),const double& b=0):c(a),r(b){
4     }
5 };
6 const int MAXN=50000;
7 Circle C[MAXN];
8 int N;
9 Point P[MAXN];
10
11 struct compx{
12     bool operator()(const Point& a,const Point& b)const{
13         if(equ(a.x,b.x)) return ls(a.y,b.y);
14         else return ls(a.x,b.x);
15     }
16 };
17 struct compy{
18     bool operator()(const Point& a,const Point& b)const{
19         if(equ(a.y,b.y)) return ls(a.x,b.x);
20         else return ls(a.y,b.y);
21     }
22 };
```

sweep line algorithm, use circle's top and bottom vertex as event point

```
1 double mindist;
2
3 Point eventp;
4 typedef set<Point,compy> Box;
5 typedef Box::iterator BoxIter;
6 void sweepline(){
7     sort(P,P+N,compx());
8     mindist=INF;
9     int l=0;
10    Box box;
11    for(int i=0;i<N;++i){
12        eventp=P[i];
13        Circle cir=C[eventp.c];
14        double x=eventp.x-mindist,negy=cir.c.y-cir.r-mindist,psy=cir.c.y+
            cir.r+mindist;
15
16        while(l<i && ls(C[P[l].c].c.x+C[P[l].c].r,x)){
17            Circle tmp=C[P[l].c];
18            box.erase(Point(tmp.c.x,tmp.c.y-tmp.r));
19            box.erase(Point(tmp.c.x,tmp.c.y+tmp.r));
20            ++l;
21        }
22
23        if(!box.empty()){
24            BoxIter begin=box.lower_bound(Point(NEGINF,negy));
25            BoxIter end=box.lower_bound(Point(NEGINF,psy));
26            for(BoxIter iter=begin;iter!=end;++iter){
27                Circle tmp=C[iter->c];
```

```

28         double d=(tmp.c-cir.c).len() - tmp.r - cir.r;
29         mindist=min(mindist,d);
30     }
31 }
32 box.insert(Point(cir.c.x, cir.c.y+cir.r, eventp.c));
33 box.insert(Point(cir.c.x, cir.c.y-cir.r, eventp.c));
34 }
35 }

```

main function, save circle's top and bottom vertex

```

1 int main(){
2     int T; scanf("%d",&T);
3     while(T-->0){
4         scanf("%d",&N);
5         for(int i=0; i<N; ++i){
6             int x,y,r;
7             scanf("%d%d%d",&x,&y,&r);
8             C[i].c.x=x; C[i].c.y=y; C[i].r=r;
9             P[i]=Point(C[i].c.x-C[i].r, C[i].c.y, i);
10        }
11        sweepline();
12        printf("%.6f\n", mindist);
13    }
14    return 0;
15 }

```

### 6.8.7 circle hierarchy

$O(N \log N)$

sweep line

event point is the most left and right vertex of a circle

implicit store the interval cuted by the current sweep line.

POJ 2932 data structures and compare functors

```

1 const int MAXC=40000;
2 const int MAXN=2*MAXC;
3 struct Circle{
4     Point c; double r;
5     Circle(const Point& a=Point(), const double& b=0):c(a),r(b){}
6 };
7 //a extra INF radius circle
8 Circle circle[MAXC+1];
9 Point P[MAXN];
10 int N;
11 Point eventp;
12
13 struct compx{
14     bool operator()(const Point& a, const Point& b) const{
15         if(equ(a.x,b.x)) return ls(a.y,b.y); else return ls(a.x,b.x);
16     }
17 };
18
19 struct Interval{
20     int p;
21     int i,j;
22     bool lowi, lowj;

```

```

23   Interval(int par=0,int a=0,bool a0=false ,int b=0,bool b0=false):
24       p(par),i(a),j(b),lowi(a0),lowj(b0){}
25   };
26   inline pair<double,double> get_itv(const Circle& c,const double& x){
27       double d=fabs(c.c.x-x);
28       double del=c.r*c.r - d*d;
29       if(lseq(del,0.0f)) del=0.0f;
30       del=sqrt(del);
31       return make_pair(c.c.y-del,c.c.y+del);
32   }
33   struct compcircle{
34       bool operator()(const Interval& a,const Interval& b)const{
35           pair<double,double> itv0=get_itv(circle[a.i],eventp.x);
36           pair<double,double> itv1=get_itv(circle[a.j],eventp.x);
37           double x0,x1;
38           if(a.lowi) x0=itv0.first;else x0=itv0.second;
39           if(a.lowj) x1=itv1.first;else x1=itv1.second;
40
41           itv0=get_itv(circle[b.i],eventp.x);
42           itv1=get_itv(circle[b.j],eventp.x);
43           double y0,y1;
44           if(b.lowi) y0=itv0.first;else y0=itv0.second;
45           if(b.lowj) y1=itv1.first;else y1=itv1.second;
46
47           if(equ(x0,y0)) return ls(x1,y1);
48           else return ls(x0,y0);
49       }
50   };
51
52   //-----union set-----
53   int parent[MAXC+1];
54   inline void make_set(){for(int i=0;i<=N;++i) parent[i]=i;}
55   //attach y to x
56   inline void link(int x,int y){parent[y]=x;}

```

sweep line  
left and right point as event point  
store following in the BST:(Interval structure)  
-- parent  
-- which circle contribute the two end points  
-- if these end points is bottom vertex of the interval

```

1   typedef set<Interval,compcircle> Itvl;
2   typedef Itvl::iterator Iter;
3   void debug(Itvl itvl){
4       for(Iter k=itvl.begin();k!=itvl.end();++k){
5           printf("parent: %d lower: %d upper: %d %d\n",k->p,k->i,k->lowi,
6               k->j,k->lowj);
7       }
8   }
9   void sweepline(){
10       make_set();
10      sort(P,P+2*N,compx());
11      Itvl itvl;
12      itvl.insert(Interval(0,0,true,0,false));
13
14      for(int i=0;i<2*N;++i){
15          eventp=P[i];
16

```

```

17     if(eventp.in){
18         Iter lower=itvl.lower_bound(Interval(0,eventp.i,true,eventp.i,
19             false));
20         —lower;
21         link(lower—>p,eventp.i);
22         Interval tmp=*lower;
23         itvl.erase(lower);
24         itvl.insert(Interval(tmp.p,tmp.i,tmp.lowi,eventp.i,true));
25         itvl.insert(Interval(eventp.i,eventp.i,true,eventp.i,false));
26         itvl.insert(Interval(tmp.p,eventp.i,false,tmp.j,tmp.lowj));
27     }
28     else{
29         Iter d=itvl.find(Interval(0,eventp.i,true,eventp.i,false));
30         Iter l=d;—l;
31         Iter u=d;++u;
32         Interval low=*l,up=*u;
33         itvl.erase(d);itvl.erase(l);itvl.erase(u);
34         itvl.insert(Interval(low.p,low.i,low.lowi,up.j,up.lowj));
35     }
36 }

```

as for POJ 2932 don't ask for the total hierarchy, so this is a special sweep line algorithm to speed up.

```

1 void sweepline(){
2     make_set();
3     sort(P,P+2*N,compx());
4     Itvl itvl;
5     itvl.insert(Interval(0,0,true,0,false));
6
7     for(int i=0;i<2*N;++i){
8         eventp=P[i];
9
10        if(eventp.in){
11            Iter lower=itvl.lower_bound(Interval(0,eventp.i,true,eventp.i,
12                false));
13            —lower;
14            link(lower—>p,eventp.i);
15            if(lower—>p == 0){
16                Interval tmp=*lower;
17                itvl.erase(lower);
18                itvl.insert(Interval(tmp.p,tmp.i,tmp.lowi,eventp.i,true));
19                itvl.insert(Interval(eventp.i,eventp.i,true,eventp.i,false));
20                itvl.insert(Interval(tmp.p,eventp.i,false,tmp.j,tmp.lowj));
21            }
22        }
23        else{
24            Iter d=itvl.find(Interval(0,eventp.i,true,eventp.i,false));
25            if(d==itvl.end()) continue;
26            Iter l=d;—l;
27            Iter u=d;++u;
28            Interval low=*l,up=*u;
29            itvl.erase(d);itvl.erase(l);itvl.erase(u);
30            itvl.insert(Interval(low.p,low.i,low.lowi,up.j,up.lowj));
31        }
32    }
33 }

```

## 7 Math

### 7.1 Catalan Number