Problem: http://www.codechef.com/CDCRFT14/problems/BIT

Counting Bits Editorial:

Simple problem. For each integer from 1 to N count set bits.
Complexity: N log (N)

Setter's solution:

```
#include<bits/stdc++.h>
#define sd(n) scanf("%d",&n)
int foo(int p)
{
    int ret=0;
    while(p)ret+=p&1,p/=2;
    return ret;
}
int main()
{
    int t;
    sd(t);
    while(t--)
    {
        int n,ans=0,i;
        sd(n);
        for(i=1; i<=n; i++)
            ans+=foo(i);
        printf("%d\n",ans);
    }
    return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/ATOM

Atoms Editorial:

Given initial atoms, limit on atoms and multiplication rate, find the time at which reaction has to be stopped.

Simple simulation. Most submissions failed due to overflow and using log functions.
Complexity: log(M-N) to base K.

Setter's solution:

```
#include<bits/stdc++.h>
using namespace std;
#define sd(n) scanf("%d",&n)
int main()
{
    int t;
    sd(t);
    while(t--)
    {
        LL n,k,m,cur,ans=0;
        cin >> n >> k >> m;
        cur=n;
        while(1)
        {
            if((double)cur*(double)k>(double)m)break;
            cur=cur*k;
            ans++;
        }
        cout << ans << endl;
    }
    return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/BALLS

Naughty Editorial:

This problem can be solved by keeping a track of how many times a number has occurred on top/bottom. Iterate through all possible numbers that we have and suppose that this number occurs on top, then find total number of moves for this number to show up on at least half of the total balls. The minimal number through all this will be the answer. To find the minimal number of turns to make we need to know two numbers: the number of balls with current color on the top and the number of balls with the current number on bottom (but not at the same time). Let it be integers a and b. Let $m=(n+1)/2$ — the minimal number count of current number to win the game. Then if $a+b<m$ it is impossible won the game using current number at all. If $a \geq m$ then the answer is 0, otherwise the answer is m-a.

Solution

```
#include<bits/stdc++.h>
#define SZ(x) ((int)(x).size())
using namespace std;
typedef pair<int,int> PII;
vector<PII> mp(100005,make_pair(0,0));
int main()
{
  int n,m,a,b;
  scanf("%d",&n); m=(n+1)/2;

  for ( int i=0; i<n; i++ ) {
     scanf("%d%d",&a,&b);
     mp[a].first++;
     if ( a!=b ) mp[b].second++;
  }
  int ans=n+1;
  for(int i=0;i<100001;i++)
  {
     int c1=mp[i].first;
     int c2=mp[i].second;
     if ( c1+c2<m )
     continue;
     ans=min(ans,max(m-c1,0));
  }
  if(ans>n)
     printf("Impossible\n");
  else
     printf("%d\n",ans);
  return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/INGREDIE

Ingredients Editorial

Problem boils down to:
Given a simple, connected, undirected weighted graph, find one path starting from first vertex given as starting point of first kid and another from the second vertex given as starting point of second kid such that together the two paths cover all the

vertices given as store locations and total sum of all edges traversed is minimized.

A simple approach involves breaking down the questions into three parts.

First part involves finding shortest paths to all vertices from starting vertices and the store location vertices.
One may use dijkstra at the given starting points and all the store locations but to save coding effort Floyd Warshall is a better idea. Constraints allow both.

Second part of the question required you to decide which kid should visit which vertex.
Because of small constraints, one may do a brute force and try all possibilities which are $2^{\text{no. of store locations}}$ = 256 iterations in the extreme case.

The third part of the questions requires you to decide in what order each kid should visit the vertices allotted to him.
Again, taking advantage of small constraints, we may try all permutations for both kids within the loop which does the second part.
A common mistake was forgetting to sort the store locations allotted before starting to check all permutations.
Trying all permutations in the worst case would be 8! = 40320 iterations
Second and third part together = 256*40320 = 10321920 which will work in time.

Setter's solution:

```
#define FOR(i,a,b) for(int i=(a); i<(b); i++)
#define INF 0x3f3f3f3f

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    int a,b,c;
    int graph[101][101];
    FOR(i,0,101)
    {
      FOR(j,0,101)
      {
           if(i==j)
                graph[i][j]=0;
           else
                graph[i][j] = INF;
      }
```

```
}
FOR(i,0,m)
{
  scanf("%d%d%d", &a, &b, &c);
  graph[a][b] = c;
  graph[b][a] = c;
}
FOR(i,0,n)
  graph[i][i]=0;
int s;
vector <int > stores;
scanf("%d", &s);
FOR(i,0,s)
{
  scanf("%d", &a);
  stores.push_back(a);
}
int start1, start2;
scanf("%d%d", &start1, &start2);
sort(stores.begin(), stores.end());
FOR(k,0,n)
  FOR(i,0,n)
       FOR(j,0,n)
            if(graph[i][k] + graph[k][j] < graph[i][j])
                 graph[i][j] = graph[i][k]+graph[k][j];
vector <int > person1, person2;
LL ans = INF, dist1, dist2;
for(int mask=0; mask<(1<<s); mask++)
{
  person1.clear();
  person2.clear();
  person1.push_back(start1);
  person2.push_back(start2);
  FOR(j,0,s)
  {
       if(!((1<<j) & mask))
       {
            person1.push_back(stores[j]);
       }
       else person2.push_back(stores[j]);
  }
  LL temp1=INF, temp2 = INF;
  do
```

```
        {
             dist1=0;
             FOR(iter1, 1, person1.size())
             {
                  dist1 += graph[person1[iter1-1]][person1[iter1]];
             }
             temp1 = min(temp1, dist1);
        }while(next_permutation(person1.begin()+1, person1.end()));
        do
        {
             dist2=0;
             FOR(iter2, 1, person2.size())
                  dist2 += graph[person2[iter2-1]][person2[iter2]];
             temp2 = min(temp2, dist2);
        }while(next_permutation(person2.begin()+1, person2.end()));
        ans = min(ans, temp1+temp2);
      }
      printf("%lld\n", ans);
      return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/DURIND

During Editorial

Let us call the “key” string as pattern and the keyholes string as “text”.
Problem boils down to:
Find all indices where the pattern is a substring of text. Now count the number of ways of placing it so that there is no overlapping and you may use any number of pattern strings.

First part of the problem can be easily done by making a failure table and using KMP algorithm:
http://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm

Second part of the problem can be done using dynamic programming.
A recursive solution would have given a stack overflow unless done bottom-up.
Here the state of the dp would have been the current index of the array containing

all the indices where the pattern is a substring.
At each point you can decide to:
1. Place the key.
   In this case you do a binary search to find the next compatible index.
2. Not place the key.
   In this case you can simply go to the next index.
This would have been a O(nlogn) solution.

A better solution is to use an iterative O(n) dynamic programming solution.
Here you just have to iterate over all indices and add the solution for all the compatible solutions which constitute the overlapping subproblems.
The recursive dynamic program could also have been done in a similar fashion but bottom up.

Tester's solution:

```
#define getSize(x) ((int) (x).size())
#define FORN(i, n) for(LL i = 0; i < n; i++)

vector<int> lps;
vector<int> match;
string pattern;
int dp[1000005];
void computeLPS()
{
    int k=0; // Length of the previous longest prefix suffix
    int M=getSize(pattern);
    lps.resize(M);
    lps[0] = 0;

    int i=1;
    while(i<M)
    {
      if(pattern[i]==pattern[k])
      {
           k++;
           lps[i] = k;
           i++;
      }
      else
      {
           if(k!=0)
                k = lps[k-1]; // not incrementing i here as we have
not found the lps value for it
           else
                lps[i++] = 0;
      }
    }
}

void KMPSearch(string text)
{
    computeLPS();
    int N=getSize(text),M=getSize(pattern);

    match.clear(); //will store the indices where the pattern matches
the text
    int i=0,j=0;
    while(i<N)
    {
      if(pattern[j]==text[i])
      {
           j++;
           i++;
      }
      if(j==M)
      {
           match.push_back(i-j);
           j = lps[j-1];
      }
      else if(pattern[j]!=text[i])
      {
           if(j!=0)
                j = lps[j-1];
           else
                i = i+1;
      }
    }
}


int main()
{
      string b;
      cin >> pattern >> b;
      KMPSearch(b);
    vector< pll > v;
      LL l=0,t=1,tt=0;
      FORN(i,SZ(match))
    {
      while(l<SZ(v) && match[i]-v[l].F>=SZ(pattern))
      {
           t=(t+v[l++].S)%MOD;
      }
      v.PB(MP(match[i],t));tt=(tt+t)%MOD;
    }
    cout << tt << endl;
    return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/SUBBXOR

Subxor editorial:

Complexity: O (N log (N))

Given an array A of size N, we need to output how many subarrays have XOR less than K.

Let array be A_1,A_2....A_N.

Let us denote XOR of subarray A_i to A_j as XOR(i,j).

a^a=0 Eqn.....1

a^0=a Eqn....2

Using Eqn. 1,2 we can prove

XOR(i,j)=XOR(1,i-1)^XOR(1,j) Eqn....3

For each index i=1 to N, we can count how many subarrays ending at i^{th} position satisfy the given condition.

We can use a data structure called trie to solve this problem. So supposing we have "insert" and "query" functions, our code will be:

```
ans=0
p=0
for i=1 to N:
    q=p^A[i]
    ans += query(q,k)
    insert(q)
    p=q
```

insert(q) inserts the integer q into the structure.

query(q,k) returns how many integers already exist into structure which when taken xor with q return an integer less than k.

Now, What is a trie? Read here:
http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=usingTries

So, we can form a trie here where each node will have at max. two children. Left child means the subtree where current bit is 0 and right child corresponds to bit 1.

For example, if we insert 5 (101) into the trie. We have this structure:

```
    NODE
      \
       NODE
      /
     NODE
      \
       NODE
```

Now if we insert 4(100) into this structure we get:

```
    NODE
       \
       NODE
       /
    NODE
   /    \
NODE    NODE
```

We will have to also keep two values at each node denoting how many leafs exist if we go to left or right from that node.

For query(q,k): Read the code.

Setter's solution:

```
#include<bits/stc++.h>
#define sd(n) scanf("%d",&n)
typedef long long LL;
typedef struct ll
{
    int leftc;
    int rightc;
    struct ll * left;
    struct ll * right;
}node;
node * insert(node * root, unsigned int n, int level)
{
    if(level==-1)return root;
    unsigned int x=((n>>level)&1);
    if(x)
    {
        root->rightc++;
        if(root->right==NULL)
        {
            root->right=(node *)malloc(sizeof(node));
            root->right->leftc=root->right->rightc=0;
        }
        root->right=insert(root->right,n,level-1);
    }
    else
    {
        root->leftc++;
        if(root->left==NULL)
        {
            root->left=(node *)malloc(sizeof(node));
            root->left->leftc=root->left->rightc=0;
        }
        root->left=insert(root->left,n,level-1);
    }
    return root;
}
```

```
unsigned int query( node * root, unsigned int n, int level, unsigned
int k)
{
    if(level==-1 || root==NULL)return 0;
    unsigned int p=((n>>level)&1);
    unsigned int q=((k>>level)&1);
    if(q)
    {
        if(p==0)return root->leftc+query(root->right,n,level-1,k);
        else return root->rightc+query(root->left,n,level-1,k);
    }
    else
    {
        if(p==0)
            return query(root->left,n,level-1,k);
        else
            return query(root->right,n,level-1,k);
    }
}
int main()
{
    int t;
    sd(t);
    while(t--)
    {
        node * root;
        root=(node *)malloc(sizeof(node));
        root->left=root->right=NULL;
        root->leftc=root->rightc=0;
        root=insert(root,0,20);
        unsigned int n,i,j,p=0,x,q,k;
        long long int ans=0;
        scanf("%u%u",&n,&k);
        for(i=0; i<n; i++)
        {
            scanf("%u",&x);
            q=p^x;
            ans+=(LL)(query(root,q,20,k));
            root=insert(root,q,20);
            p=q;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/DRUNKG

Problem Description:

Alice and Bob are playing a game. They have a pile of N coins. The rules of the game are:

1. Alice moves first. Both players alternate turns.
2. In a move a player first decides on the number of coins to be picked up from the pile. The possible options are 1, 4, 9, 16. The number of coins in the pile should be greater than or equal to the number of coins decided by the player.
3. Even if the the player decides to pick up some number of coins from (1, 4, 9, 16), he/she might not end up picking exactly those coins from the pile. If the player has decided to pick up X coins from the pile.
    a. The number of coins currently in the pile should be greater than or equal to X.
    b. The player might pick up X - 1, X, X + 1 coins from the pile finally with the relative probability of C1: C2: C3. This means that upon deciding to pick up X coins, the player actually picks up X - 1 coins with the probability C1 / (C1 + C2 + C3) and so on.
    c. It it happens that actually picking up X + 1 coins from the pile is not possible as there are only X coins remaining in the pile, then the relative probability will become C1: C2. Similarly, if it happens that the person cannot really pick up X - 1 coins as it becomes 0, the relative probability distribution between the remaining options becomes C2: C3
    d. In each turn in a valid move a person should always end up picking at least 1 coin. So, in case where there is only one coin remaining in the pile, the current player will pick that 1 coin with all probability.
4. The player who is unable to make a move loses the game.
5. Both player's intentions of winning or losing are given in the input. Both Alice and Bob can individually either want to win the game or lose.
6. We have to return the probability of Bob getting whatever he wants, winning or losing.

Solution Details:

The most important point in this problem is that, in a particular turn, a player will always choose the option which maximizes the probability of whatever that player wants. So, let us define a function solve (n, turn)

solve (n, turn): If there are n coins in the pile, it returns the probability of winning for the player whose turn it is.
So, depending on whether Bob wants to lose or not, we call this function for Bob and get his probability of winning in this situation. There are broadly 2 types of cases which arrive in computing the solve function:

1. If the current player wants to win, they will try to minimize the second player's solve function for the remaining coins for all possible moves and choose the best move accordingly.

2. If the current player wants to lose, they will try to maximize the second player's solve function for the remaining coins for all possible moves and choose the best move accordingly.

The base case is when there are 1 or 0 coins remaining.

Tester's Solution:

```
#include<bits/stdc++.h>
using namespace std;

double dp[1<<20][2];
int vis[1<<20][2];
int moves[] = {1,4,9,16};
int dir[2];
int c[3];

double rec(int cur, int turn) {
 double &ret = dp[cur][turn];
 if(vis[cur][turn] == 0) {
   vis[cur][turn] = 1;
   if(cur == 0) {
     if(dir[turn] == 0) {
       ret = 1.0;
     } else {
       ret = 0.0;
     }
   } else {
     ret = 0.0;
     for(int i=0;i<4;i++) {
       int total = 0;
       int cmove = moves[i];
```

```
        if(cmove > cur) continue;
        double cret = 0.0;
        for(int j=-1;j<=1;j++) {
          if(cmove+j <= cur && cmove + j > 0) {
            total += c[j+1];
            cret += c[j+1]*(rec(cur - (cmove+j), 1-turn));
          }
        }
        if(total == 0) {
          continue;
        }
        cret /= total;
        if(dir[0] == dir[1]) {
          ret = max(ret, 1.0 - cret);
        } else {
          ret = max(ret, cret);
        }
      }
    }
  }
 return ret;
}

int main() {
 int t;
 scanf("%d",&t);
 while(t--) {
   int n;
   scanf("%d",&n);
   scanf("%d%d%d",&c[0],&c[1],&c[2]);
   scanf("%d%d",&dir[1],&dir[0]);
   assert(n >= 1 && n <= 1000000);
   for(int i=0;i<=n;i++)
     vis[i][0] = vis[i][1] = 0;
   for(int i=0;i<=n;i++) {
     rec(i,0);
     rec(i,1);
   }
   if(dir[0] == dir[1]) {
     printf("%.4lf\n",(1.0 - rec(n,1)));
   } else {
     printf("%.4lf\n",rec(n,1));
   }
 }
 return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/GIVEAWAY

The problem directly asks to find, given an array, you have queries and updates of the following form.
Query(x,y,z) = Find the number of elements in the array between index x and y which are less than z.
Update(x,y) = Change element at position x to y.

The following is an online solution to do the above.
Say the array is of length N.
Consider a segment tree like data structure, where the root node represents interval from 1..N.
Its left child represents 1..N/2 and right child represents N/2+1..N and so on.
say we want to find Query(x,y,z).
We know the set of intervals which will belong to the range (x,y).
There will be O(log N) such intervals(segment tree).
Now, if for all these intervals, we can find the rank of element z in all intervals, the required answer is the sum of these ranks.
To find the ranks, we can just keep the elements belonging to the respective intervals in the segment tree in sorted order. Then, we can binary search in each of the logN intervals to get the respective ranks.
The above will work when there are no updates. But since there are updates, we need a ranked data structure which also supports updates.
Hence, we will replace a sorted array at each node with an AVL tree. This will allow you to find the rank of an element in O(log N) and update elements in O(log N).
Hence, to update an element, we need to update O(log N) intervals(from root to leaf of segment tree) and each update, at each node, will take O(log N). Hence the complexity per update is O(log^2 N).
To query an element, we need to binary search in O(log N) intervals, hence one query will take O(log^2 N) time.
Hence the time complexity is O(Q log^2 N).
To store the segment tree, we need to store on each node the AVL tree for that interval, hence at each level of the segment tree, each element appears once. So, the total space complexity is O(N log N).

Problem: http://www.codechef.com/CDCRFT14/problems/NQUEEN

Queen Editorial

Prereq:
Lowest Common Ancestor in a tree
Articulation Points
Bridges

Yes this question was not for beginners. :D

After all the story, the problem boils down to this:
Given a graph, There are two type of queries:

1. Given two nodes in a graph, say A and B. Tell if there still exists a path between A and B, even if the edge between C and D is removed from the graph.
2. Given two nodes in a graph, say A and B. Tell if there still exists a path between A and B, even if the node C is completely removed from the graph.

Step1 : Do a DFS on the graph, and compute low[x] for every node ‘x’. low[x] is the upper most level, level[y] such that there is an edge from the subtree of ‘x’ to ‘y’. (Read about articulation points)
Step2 :
For query type1:
if lca(A,B) exists on a level below or equal to the level of max(level[C],level[D]) then they can not act as a bridge between A and B now as the lca(A,B) lies above level of max(level[C],level[D]).
let us assume without loss of generality that C is below D in the tree. Now find ancestor of A which is at the level of C, and call it X. if low[X] == level[X] and parent[X] == D and X==C then the edge CD was a bridge.
For query type2:
it is the more general case of query 1, and do some boundary changes on query1 and you will be able to solve the query of type2.

Here is Gennady’s solution for the problem:

```
const int N = 400010;

map <string, int> mp;
vector <int> g[N];
bool ADD;
int n;
int get(string w) {
  // capitalize?
  if (mp.find(w) == mp.end()) {
    if (ADD) {
      g[n].clear();
      mp[w] = n++;
    }
    else return -1;
  }
  return mp[w];
}

int md[N], de[N], comp[N], tin[N], tout[N];
int pv[N], pr[N][21];
int T, nc;

void dfs(int v) {
  tin[v] = ++T;
  comp[v] = nc;
  int sz = g[v].size();
  md[v] = de[v];
  for (int j = 0; j < sz; j++) {
    int u = g[v][j];
    if (u == pv[v]) {
      continue;
    }
    if (comp[u] == -1) {
      pv[u] = v;
      de[u] = de[v] + 1;
      dfs(u);
      if (md[u] < md[v]) {
        md[v] = md[u];
      }
    } else
    if (de[u] < md[v]) {
      md[v] = de[u];
    }
  }
  tout[v] = ++T;
}

bool anc(int x, int y) {
  return (tin[x] <= tin[y] && tout[y] <= tout[x]);
```

```
}

int main() {
  int m;
  cin >> m;
  cin >> m;
  mp.clear();
  n = 0;
  ADD = true;
  for (int i = 0; i < m; i++) {
    string w1, w2;
    cin >> w1 >> w2;
    int u1 = get(w1);
    int u2 = get(w2);
    g[u1].push_back(u2);
    g[u2].push_back(u1);
  }
  ADD = false;
  for (int i = 0; i < n; i++) comp[i] = -1;
  nc = 0;
  T = 0;
  for (int i = 0; i < n; i++)
    if (comp[i] == -1) {
      pv[i] = i;
      de[i] = 0;
      dfs(i);
      nc++;
    }
  for (int i = 0; i < n; i++) pr[i][0] = pv[i];
  for (int j = 1; j < 18; j++)
    for (int i = 0; i < n; i++) pr[i][j] = pr[pr[i][j - 1]][j - 1];
  int tt;
  cin >> tt;
  while (tt--) {
    int com;
    cin >> com;
    string wa, wb, wc, wd;
    int ua, ub, uc, ud;
    bool result = false;
    if (com == 1) {
      cin >> wa >> wb >> wc >> wd;
      int ua = get(wa);
      int ub = get(wb);
```

```
        int uc = get(wc);
        int ud = get(wd);
        if (wa == wb) result = true; else
        if (ua == -1 || ub == -1) result = false; else
        if (comp[ua] != comp[ub]) result = false; else
        if (comp[ua] != comp[uc]) result = true; else {
          if (de[uc] < de[ud]) {
            swap(uc, ud);
          }
          if (pv[uc] != ud) result = true; else
          if (anc(uc, ua) == anc(uc, ub)) result = true; else
          if (md[uc] != de[uc]) result = true;
          else result = false;
        }
      } else {
        cin >> wa >> wb >> wc;
        int ua = get(wa);
        int ub = get(wb);
        int uc = get(wc);
        if (wa == wb) result = true; else
        if (ua == -1 || ub == -1) result = false; else
        if (comp[ua] != comp[ub]) result = false; else
        if (uc == -1) result = true; else
        if (uc == ua || uc == ub) result = false; else
        if (comp[ua] != comp[uc]) result = true; else {
          if (!anc(uc, ua) && !anc(uc, ub)) result = true; else {
            if (anc(uc, ua)) {
              for (int j = 17; j >= 0; j--)
                if (de[pr[ua][j]] > de[uc]) ua = pr[ua][j];
            }
            if (anc(uc, ub)) {
              for (int j = 17; j >= 0; j--)
                if (de[pr[ub][j]] > de[uc]) ub = pr[ub][j];
            }
            if (ua == ub) result = true; else {
              if (pv[ua] == uc && md[ua] >= de[uc]) result = false;
else
              if (pv[ub] == uc && md[ub] >= de[uc]) result = false;
              else result = true;
            }
          }
        }
      }
```

```
    puts(result ? "STABLE" : "NOT STABLE");
  }
  return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/COPRIME

Coprime Editorial:

Problem boils down to:
Given a bipartite graph with directed edges, do matching to maximise the edges, but each vertex should either have all incoming or all outgoing edges (not both).

We can solve this problem by making a new bipartite graph, where, vertices on left side in new graph denote the directed edges from left to right in original graph. Vertices on right side in new graph denote the directed edges from right to left in original graph.

In the new graph, edges will be between those vertices which contradict each other. Two vertices contradict each other when both cannot exist in the final matching of the original graph. We need to remove minimum vertices in the new graph such that all contradictions (ie. edges in new graph) are removed.

This is same a minimum vertex cover, which is same as maximum bipartite matching.

Setter's solution:

```
#include<bits/stdc++.h>
#define pb push_back(n)
#define mp make_pair(n)
#define sd(n) scanf("%d",&n)
int gcd(int a, int b)
{
    if(b==0)return a;
    return gcd(b,a%b);
}
bool match(int u,
        const vector<vector<int> > & connections,
        vector<bool> & visited,
        vector<int> & matchL,
        vector<int> & matchR) {
    int M = connections[u].size();
    for (int j = 0; j < M; j++) {
        int v = connections[u][j];
        if (!visited[v]) {
            visited[v] = true;
            if (matchR[v] < 0 || match(matchR[v], connections,
visited, matchL, matchR)) {
                matchL[u] = v;
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}
vector < pair < int , int> > ed1,ed2;
int foo(int N) {
    vector<vector<int> > connections = vector<vector<int> >(N,
vector<int>());

    for (int i = 0; i < (int)ed1.size(); i++)
    {
        for (int j = 0; j < (int)ed2.size(); j++)
        {
            if(ed1[i].F==ed2[j].S || ed1[i].S==ed2[j].F)
                connections[i].push_back(j);
        }
    }

    vector<int> matchL = vector<int>(N, -1);
    vector<int> matchR = vector<int>(N, -1);

    int t = 0;
    for (int i = 0; i < N; i++) {
        vector<bool> visited(N, false);
        if (match(i, connections, visited, matchL, matchR))
            t++;
    }

    return t;
```

```
}
int main()
{
    int n,i,j,ans,ar[500],arr[500];
    sd(n);
    for(i=0; i<n; i++)
        sd(ar[i]);
    for(i=0;i<n; i++)
        sd(arr[i]);
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            if(arr[j]>ar[i] && gcd(ar[i],arr[j])!=1)
                ed1.pb(mp(i,j));
    }
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            if(ar[j]>arr[i] && gcd(arr[i],ar[j])!=1)
                ed2.pb(mp(i,j));
    }
    cout << (int)ed1.size() + (int)ed2.size() -
foo(max((int)ed1.size(),(int)ed2.size())) <<endl;
    return 0;
}
```

Problem: http://www.codechef.com/CDCRFT14/problems/CIRCLES

Circles Editorial:

Finding circumcenter of a circle given 3 pts on that circle ->
Let three points be A,B,C

Now, make two vectors AB(joining A and B) and AC(similarly)

ABP -> perpendicular bisectors of AB

ACP -> perpendicular bisectors of AC

Circumcenter would be meeting point of ABP and ACP

Now translate the axis to center of first circle.
Rotate the axis in such a manner that first circle comes in xy-plane.

Now get the plane of second circle in the form ax+by+cz=d;

we can get the line of intersection of this plane with xy plane(plane of first circle) by putting z=0;
so now intersecting line becomes ax+by=d;
now first circle is centered at origin in xy plane;
so equation of first circle will be x^2 + y^2 = r^2 ;
now we have to find the intersection of this circle and the intersecting line.

We will be getting a quadratic equation on solving this->

1) imaginary roots -> line doesn't intersect the first circle[Not possible]
    two rings can't entangle because plane of second ring doesn’t intersect first(ring in xy plane) ring.
2) equal roots -> first circle intersect plane of second circle at only one point[Not Possible}
    we have been given both rings don't touch each other, therefore if first ring only touches the second plane, its not possible to entangle.
3) distinct roots ->
    Let a,b are the two points that came from solving the equations of first circle and second plane.
    For entangling two rings -> one point(a,b) should lie inside the second ring and other should lie outside the second ring.

Setter’s solution:

```
#include<bits/stdc++.h>

using namespace std;
#define SZ(V) (LL)V.size()
#define FORN(i, n) for(LL i = 0; i < n; i++)
#define FORAB(i, a, b) for(LL i = a; i <= b; i++)
```

```
#define eps 1e-8
#define PB push_back

typedef long long LL;
typedef pair<LL,LL> pll;
typedef vector<double> vd;
bool flag;

void normalize(vd &a){// normalizing a vector
     double d;
     FORN(i,3) d+=a[i]*a[i];
     if(d>=eps) FORN(i,3) a[i]/=sqrt(d);
}
vd cross(vd a,vd b){// calculating cross product of two vectors
     vd c(3);
     FORN(i,3) c[0+i]=a[(1+i)%3]*b[(2+i)%3] - a[(2+i)%3]*b[(1+i)%3];
     normalize(c);
     return c;
}
vd  circum_center(vd  a,vd  b,vd  c){//  calculating  circumcenter  of
circle given three points a,b,c
     vd ab(3),ac(3);
     FORN(i,3) ab[i]=b[i]-a[i];
     FORN(i,3) ac[i]=c[i]-a[i];
     vd n=cross(ab,ac);
     vd abp=cross(n,ab);//slope of perpendicular bisector of AB
     vd acp=cross(n,ac);//slope of perpendicular bisector of AC

     vd abm(3),acm(3);
     FORN(i,3)  abm[i]=(a[i]+b[i])/2.0;//point  lying  on  perpendicular
bisector of AB
     FORN(i,3)  acm[i]=(a[i]+c[i])/2.0;//point  lying  on  perpendicular
bisector of AC
     //  we  have  two  perpendicular  bisectors  and  center  would  be  the
intersecting point of these bisectors
     // calculating center of circle
     double x=-190.0,y=-190.0;
     FORN(i,3){
          if(abs(abp[(0+i)%3]*acp[(1+i)%3]                                    -
abp[(1+i)%3]*acp[(0+i)%3]) >eps){
               x                                                      =(
(acp[(0+i)%3]*abm[(1+i)%3]-acp[(1+i)%3]*abm[(0+i)%3])                 -
(acp[(0+i)%3]*acm[(1+i)%3]-acp[(1+i)%3]*acm[(0+i)%3])
)/(abp[(0+i)%3]*acp[(1+i)%3] - abp[(1+i)%3]*acp[(0+i)%3]) ;
          }
          if(abs(abp[(0+i)%3]*acp[(1+i)%3]                        -
abp[(1+i)%3]*acp[(0+i)%3]) >eps){
               y=                                                  (
(abm[(1+i)%3]*abp[(0+i)%3]-abm[(0+i)%3]*abp[(1+i)%3])             -
(acm[(1+i)%3]*abp[(0+i)%3]-acm[(0+i)%3]*abp[(1+i)%3])
)/(abp[(0+i)%3]*acp[(1+i)%3] - abp[(1+i)%3]*acp[(0+i)%3]) ;
          }
     }
     vd center(3);
     FORN(i,3) center[i] = (abp[i]*x) + abm[i];
     return center;
}
bool check_null(vd a){ // checking it magnitude of vector is zero
     return (abs(a[0])<eps && abs(a[1])<eps && abs(a[2])<eps);
}
void rotate(vd center,vd a,vd b,vd c,vd &nx,vd &ny,vd &nz){
     vd ca(3),cb(3);
     if(!check_null(cross(a,b))){
          FORN(i,3) ca[i]=a[i]-center[i];
          FORN(i,3) cb[i]=b[i]-center[i];
     }
     else{
          FORN(i,3) ca[i]=b[i]-center[i];
          FORN(i,3) cb[i]=c[i]-center[i];
     }
     normalize(ca);
     nx=ca;// taking vector from origin to a point on first circle as
new x-axis
     nz=cross(nx,cb);// take new z axis to be the normal to plane of
first circle
     ny=cross(nz,nx);// new y axis can be calculated by fact that  nx
X ny = nz;  where X be cross product
     return;
}
double dot(vd a,vd b){ // dot product of two vectors
     double ans=0;
     FORN(i,3) ans+=a[i]*b[i];
     return ans;
}
void update(vd &a,vd x,vd y,vd z){ // calculating new coordinates by
taking projection of a vector to new axis
```

```
      vd b=a;
      a[0]=dot(b,x);
      a[1]=dot(b,y);
      a[2]=dot(b,z);
      return;
}
bool check(vd center,vd p,vd a,vd b){// checking if one of a,b lies
inside the second circle and other outside
      double radius=0.0;
      FORN(i,3) radius+=(p[i]-center[i])*(p[i]-center[i]);
      double da=0.0,db=0.0;
      FORN(i,3) da+=(a[i]-center[i])*(a[i]-center[i]);
      FORN(i,3) db+=(b[i]-center[i])*(b[i]-center[i]);
      return ((min(da,db)<radius) && (max(da,db)>radius)) ;
}
int main()
{
      LL test,sum,a[3];
      cin >> test;
      while(test--){
            vd a1(3),a2(3),a3(3),b1(3),b2(3),b3(3);
            FORN(i,3) cin >> a1[i];
            FORN(i,3) cin >> a2[i];
            FORN(i,3) cin >> a3[i];
            FORN(i,3) cin >> b1[i];
            FORN(i,3) cin >> b2[i];
            FORN(i,3) cin >> b3[i];
            vd c1=circum_center(a1,a2,a3);
            //    translating the axis to c1
            FORN(i,3) a1[i]=a1[i]-c1[i];
            FORN(i,3) a2[i]=a2[i]-c1[i];
            FORN(i,3) a3[i]=a3[i]-c1[i];
            FORN(i,3) b1[i]=b1[i]-c1[i];
            FORN(i,3) b2[i]=b2[i]-c1[i];
            FORN(i,3) b3[i]=b3[i]-c1[i];
            vd nx,ny,nz;
            FORN(i,3) c1[i]=0.0;
            rotate(c1,a1,a2,a3,nx,ny,nz);
            // rotating axis so that first circle would end up on xy
axis
            // new x,y,z axis would be nx,ny,nz respectively
            update(a1,nx,ny,nz);
            update(a2,nx,ny,nz);
```

```
          update(a3,nx,ny,nz);
          update(b1,nx,ny,nz);
          update(b2,nx,ny,nz);
          update(b3,nx,ny,nz);
          /* calculating equation of plane of second circle
               ax+by+cz=d;
               a->constants[0]; b-> constants[1]; c-> constants[2];
d->d;
          */
          vd constants(3);
          FORN(i,3){
               constants[i]      =      (      (b2[(1+i)%3]      -
b1[(1+i)%3])*(b3[(2+i)%3]-b1[(2+i)%3])      -      (b3[(1+i)%3]      -
b1[(1+i)%3])*(b2[(2+i)%3]-b1[(2+i)%3]) );
          }
          double d=0.0,radius=0.0;
          FORN(i,3) d+=b1[i]*constants[i];
          FORN(i,3) radius+=a1[i]*a1[i];
          radius=sqrt(radius);
          // equation of first circle will become x^2 + y^2 = r^2
          // solving both equations for plane(of second circle) and
the first circle itself
          flag=true;
          if(abs(constants[0])<eps){
               if(abs(constants[1])<eps) flag=false;
               else{
                    double y=d/constants[1],k=(radius*radius - y*y);
                    if(abs(k) <eps ) flag=false;
                    else{
                         vd a(3,0.0),b(3,0.0);
                         a[1]=b[1]=y;
                         a[0]=sqrt(k);
                         b[0]=-sqrt(k);
                         // checking if these circles are entangled
                         //  a,b  are  intersecting  points  of  first
circle and the plane
                         if(!check(circum_center(b1,b2,b3),b1,a,b))
flag=false;
                    }
               }
          }
          else{
               double a,b,c;
```

```
                a=            (constants[0]*constants[0])           +
(constants[1]*constants[1]);
                b= -2.0*constants[1]*d;
                c= d*d - constants[0]*constants[0]*radius*radius;
                double k=b*b -4*a*c;
                if(k<eps) flag=false;
                else{
                     vd x(3,0.0),y(3,0.0);
                     x[1] = (-b + sqrt(k))/(2*a);
                     y[1] = (-b - sqrt(k))/(2*a);
                     x[0] = (d-b*x[1])/a;
                     y[0] = (d-b*y[1])/a;
                     // checking if these circles are entangled
                     //  x,y  are  intersecting  points  of  first  circle
and the plane
                     if(!check(circum_center(b1,b2,b3),b1,x,y))
flag=false;
                }
           }
           cout << (flag ? "YES" : "NO") << endl;
      }
      return 0;
}
```