CodeEaters (IIT Patna)

TEAM NOTEBOOK (ICPC Kharagpur and Amritapuri)

Property of 3 idiots:-
Diksha Bansal
Vatsal Singhal
Chandan Kumar

Strongly Connected Components (Kasuraja's Algo):

```
void fillOrder(int v, bool visited[], stack<int> &Stack)
{
   visited[v] = true;
   list<int>::iterator i;
   for(i = adj[v].begin(); i != adj[v].end(); ++i)
      if(!visited[*i])
         fillOrder(*i, visited, Stack);
   Stack.push(v);
}
void printSCCs()
{
   stack<int> Stack;
   bool *visited = new bool[V];
   for(int i = 0; i < V; i++)
      visited[i] = false;
   // Fill vertices in stack according to their finishing times
   for(int i = 0; i < V; i++)
      if(visited[i] == false)
         fillOrder(i, visited, Stack);
   Graph gr = getTranspose();
   for(int i = 0; i < V; i++)
      visited[i] = false;
   while (Stack.empty() == false)
   {
      // Pop a vertex from stack
      int v = Stack.top();
      Stack.pop();
      if (visited[v] == false)
      {
         gr.DFSUtil(v, visited);
         cout << endl;
      }
   }}
```

Articulation points and Bridges:

v : vector used to store adjacency list
visited : boolean array to keep track of nodes visited
disc : int array to store discovered time of vertex
low is int array to which stores, for every vertex v, the discovery time of the earliest discovered vertex to which v or any vertices in the subtree rooted at v is having a back edge. initialized by INFINITY.
parent : int array used to store parent of each node.
is : bool array if ith vertex is an articulation point.
time : used to keep track of discovered time.
ans : vector of pair<int ,int> used to store bridges.

```
void dfs(ll x,  ll time) {
   visited[x] = true;
   disc[x] = low[x] = time+1;
   ll child = 0;
   fr(i,v[x].size()) {
      ll a = v[x][i];
      if(a==parent[x]) continue;
      if(visited[a]) low[x] = min(low[x] , disc[a] );
      else {
         child++;
         parent[a] = x;
         dfs(a,time+1);
         low[x] = min(low[x], low[a]);
         if(parent[x]==-1 && child>1)
            is[x] = true,num++;
         else if(parent[x]!=-1 && low[a]>=disc[x])
            is[x] = true,num++;
         if(low[a]>disc[x])
            ans.pb(mp(x,a));
      }} }
```

0-1 BSF:

You have a graph G with V vertices and E edges. The graph is a weighted graph but the weights can only be 0 or 1. Write an efficient code to calculate shortest path from a given source.

```
for all v in vertices:
    dist[v] = inf
dist[source] = 0;
deque d
d.push_front(source)
while d.empty() == false:
    vertex = get front element and pop as in BFS.
    for all edges e of form (vertex , u):
        if travelling e relaxes distance to u:
            relax dist[u]
            if e.weight = 1:
                d.push_back(u)
            else:
                d.push_front(u)
```

Euler path/circuit:

Euler path in undirected graph:
Graph is connected and all vertices have even degree except or 2 have odd degrees.
Euler Circuit in undirected graph:
All vertices have even degree and graph is connected.
Euler circuit in directed graph:
All vertices are a part of a single strongly connected component and indegree and outdegree of all vertices is same,

Bit Manipulation:

1. To multiply by 2^x : S = S<<x
2. To divide by 2^x : S = S>>x
3. To set jth bit : S|=(1<<j)
4. To check jth bit : T = S &(1<<j) (If T=0 not set else set)
5. To turn off jth bit : S&=~(1<<j)
6. To flip jth bit : S^=(1<<j)
7. To get value of LSB: T = (S &(-S)) (Gives 2^position)
8. To turn on all bits S = (1<<n) - 1
in a set of size n:

Techniques:

1. For counting problems, try counting number of incorrect ways instead of correct ways.
2. Prune Infeasible/Inferior Search Space Early
3. Utilize Symmetries
4. Try solving the problem backwards
5.Binary Search the answer
6. Meet in the middle (Solve left half, Solve right half, combine)
7. Greedy
8. DP

Hierholzer’s algorithm for directed graph:

```
void printCircuit(vector< vector<int> > adj)
{
   unordered_map<int,int> edge_count;

   for (int i=0; i<adj.size(); i++)
   {
      edge_count[i] = adj[i].size();
   }

   if (!adj.size())
      return;
   stack<int> curr_path;
   vector<int> circuit;
   curr_path.push(0);
   int curr_v = 0;

   while (!curr_path.empty())
   {
      if (edge_count[curr_v])
      {
         curr_path.push(curr_v);
         int next_v = adj[curr_v].back();
         edge_count[curr_v]--;
         adj[curr_v].pop_back();
         curr_v = next_v;
      }
      else
      {
         circuit.push_back(curr_v);
         curr_v = curr_path.top();
         curr_path.pop();
      }
   }
```

9. Analyse complexity carefully
10. Reduce the problem to some standard problem
11. Add m when doing modular arithmetic.
12. Carefully analyse reasoning behind adding small details in the Q.
13. Use exponential search in case of unbounded search.

STL DS:

stack<type> name
empty(),size(),pop(),top(),push(x)
queue<type> name
empty(),size(),pop(),front(),back(),push(x)
priority_queue <type> name
empty(),size(),pop(),top(),push(x)
deque<type> name
pop_front(),pop_back(),push_front(),push_back(),size(),at(index),front(),back()
set/multiset/map/multimap<type>name
begin(),end(),size(),empty(),insert(val),erase(itr or val),find(val), lower_bound(val),upper_bound(val)
(lower bound includes val, upper bound does not)
pair<type,type> name (first and second)

STL Algorithms:

1.sort(first_iterator, last_iterator) – To sort the given vector.
2. reverse(first_iterator, last_iterator) – To reverse a vector.
3. *max_element (first_iterator, last_iterator) – To find the maximum element of a vector.
4. *min_element (first_iterator, last_iterator) – To find the minimum element of a vector.
5. accumulate(first_iterator, last_iterator, initial value of sum) – Does the summation of vector elements
6. binary_search(first_iterator, last_iterator, x) – Tests whether x exists in sorted vector or not.
7.lower_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value not less than 'x'.

```
    for (int i=circuit.size()-1; i>=0; i--)
    {
        cout << circuit[i];
        if (i)
           cout<<" -> ";
    }
}
```

Bipartite graph: Coloring possible with 2 colors.

Ford-Fulkerson (Edmond Karp) max flow Algorithm:

O(EV^3)

```
bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return (visited[t] == true);
```

8.upper_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

9.count(first_iterator, last_iterator,x) – To count the occurrences of x in vector.
10.next_permutation(first_iterator, last_iterator) – This modified the vector to its next permutation.
11.prev_permutation(first_iterator, last_iterator) – This modified the vector to its previous permutation
12. random_shuffle(arr.begin(), arr.end());
13. ios_base::sync_with_stdio(false);
    cin.tie(NULL);

Number Theory:

1. To calculate sum of factors of a number, we can find the number of prime factors and their exponents. N = ae1 * be2 * ce3 …
Then sum = (1 + a + a^2….)(1 + b + b^2 ..)...
Number of factors=(a+1)*(b+1)...
2.Every even integer greater than 2 can be expressed as the sum of 2 primes.
3. For rootn prime method, check for 2, 3 then:
for (i=5; i*i<=n; i=i+6) n%i and n%(i+2)
4. Number of divisors will be prime only if N=p^x where p is prime.
5. Kth prime factor= store smallest factor in seive and repeatedly divide with it to get the answer.
6. fib(n+m)=fib(n)fib(m+1)+fib(n-1)fib(m)
7. A number is Fibonacci if and only if one or both of (5*n2 + 4) or (5*n2 – 4) is a perfect square
8. every positive Every positive integer can be written uniquely as a sum of distinct non-neighbouring Fibonacci numbers.
9. Matrix multiplication
mul[i][j] += a[i][k]*b[k][j];
10. Root n under mod p exists only if
    n^((p-1)/2) % p = 1

```
}
int fordFulkerson(int graph[V][V], int s, int t)
{
   int u, v;
   int rGraph[V][V];
   for (u = 0; u < V; u++)
      for (v = 0; v < V; v++)
          rGraph[u][v] = graph[u][v];

   int parent[V];

   int max_flow = 0;
   while (bfs(rGraph, s, t, parent))
   {
      int path_flow = INT_MAX;
      for (v=t; v!=s; v=parent[v])
      {
         u = parent[v];
         path_flow = min(path_flow, rGraph[u][v]);
      }
      for (v=t; v != s; v=parent[v])
      {
         u = parent[v];
         rGraph[u][v] -= path_flow;
         rGraph[v][u] += path_flow;
      }
      max_flow += path_flow;
   }
   return max_flow;
}
```

Dinic’s Algorithm: **O(VE^2)**

```
const int MAXN = ...;
const int INF = 1000000000;
```

11.divisibility by 4: last 2 digits divisible by 4

12.divisibility by 8: last 3 digits divisible by 8

13. Divisibility by 3,9: sum of digs divisible by 3,9

14. Divisibility by 11: alternate (+ve,-ve) digit sum is divisible by 11

15. Divisibility by 12: divisible by 3 and 4

16. Divisibility by 13: alternating sum in blocks of 3 (L to R) div 13

17. Integral solution of ax+by=c exists if gcd(a,b) divides c

Probability:

P(all events) = P(E1) * P(E2) * ... * P(En)

P(at least one event) = 1 - P(E1') * P(E2') * ... * P(En')

$P(A \cap B) = P(A) + P(B) - P(A \cup B)$

Probability of A if B has happened:

$P(A|B) = P(A \cap B) / P(B)$

expected value is the sum of: [(each of the possible outcomes) × (the probability of the outcome occurring)].

Var(X) = E(X^2) – m^2

Seive of Eratostones:

```
vector<ll> prime;
void SieveOfEratosthenes(ll n)
{
```

```
int n, c[MAXN][MAXN], f[MAXN][MAXN], s, t, d[MAXN], ptr[MAXN],
q[MAXN];

bool bfs() {
        int qh=0, qt=0;
        q[qt++] = s;
        memset (d, -1, n * sizeof d[0]);
        d[s] = 0;
        while (qh < qt) {
                int v = q[qh++];
                for (int to=0; to<n; ++to)
                if (d[to] == -1 && f[v][to] < c[v][to]){
                                q[qt++] = to;
                                d[to] = d[v] + 1;
                        }}
        return d[t] != -1;
}
 int dfs (int v, int flow) {
        if (!flow)  return 0;
        if (v == t)  return flow;
        for (int & to=ptr[v]; to<n; ++to) {
                if (d[to] != d[v] + 1)  continue;
                int pushed = dfs (to, min (flow, c[v][to] - f[v][to]));
                if (pushed) {
                        f[v][to] += pushed;
                        f[to][v] -= pushed;
                        return pushed;
                }
        }
        return 0;
}
int dinic()
```

```
   bool prim[n+1];
   memset(prim, true, sizeof(prim));
   prime.pb(2);
   for(ll i=4; i<=n; i+=2) prim[i] = false;
   for(ll i=3; i<=n; i+=2){
        if(prim[i] ){
            prime.pb(i);
            for(ll j=2*i; j<=n; j+=i) prim[i] = false;
        } }}
```

Extended Euclid’s Algorithm:

```
LL gcde(LL a,LL b,LL *x,LL *y)
{
  if (a == 0)
  {
     *x = 0, *y = 1;
     return b;
  }
  LL x1, y1;
  LL gcd = gcde(b%a, a, &x1, &y1);
  *x = y1 - (b/a) * x1;
  *y = x1;
  return gcd;
}
```

To find inverse of a wrt m:
gcde(a,m,&x,&y);
x is the inverse of a.

Segmented Sieve for primes

```
void segsieve(LL l,LL r)
{
  LL limit = floor(sqrt(r))+1;
```

```
{
        int flow = 0;
        for (;;) {
                if (!bfs())  break;
                memset (ptr, 0, n * sizeof ptr[0]);
                while (int pushed = dfs (s, INF))
                        flow += pushed;
        }
        return flow;
}
```

Maximum Bipartite Matching:

O(M*N*N)

```
bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[])
{
   // Try every job one by one
   for (int v = 0; v < N; v++)
   {
      // If applicant u is interested in job v and v is
      // not visited
      if (bpGraph[u][v] && !seen[v])
      {
         seen[v] = true; // Mark v as visited
         // If job 'v' is not assigned to an applicant OR
         // previously assigned applicant for job v (which is matchR[v])
         // has an alternate job available.
         // Since v is marked as visited in the above line, matchR[v]
         // in the following recursive call will not get job 'v' again
         if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen, matchR))
         {
            matchR[v] = u;
            return true;
         }
      }
   }
vector<LL> prime;
sieve(limit, prime);
   limit=r-l+1;
   bool mark[limit+1];
   memset(mark, true, sizeof(mark));
  //True= is prime
   for (int i = 0; i < prime.size(); i++)
   {
      int loLim = floor(l/prime[i]) * prime[i];
      if (loLim < l)
         loLim += prime[i];

      for (int j=loLim; j<=r; j+=prime[i])
         mark[j-l] = false;
   }
}
```

Modular power

```
LL Mpow(LL x, unsigned LL y, LL m)
{
  LL res = 1;
  x = x % m;
  while (y > 0)
  {
     if (y & 1)
        res = (res*x) % m;
     y = y>>1; // y = y/2
     x = (x*x) % m; }
  Return res;}
```

Matrix Exponentiation

```
LL power(LL F[3][3], LL n)
{
   LL M[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}};
   if (n==1)
      return F[0][0] + F[0][1];
```

```
   return false;
}

int maxBPM(bool bpGraph[M][N])
{
// The value of matchR[i] is the applicant number
// assigned to job i
   int matchR[N];
   memset(matchR, -1, sizeof(matchR));

   int result = 0; // Count of jobs assigned to applicants
   for (int u = 0; u < M; u++)
   {
      // Mark all jobs as not seen for next applicant.
      bool seen[N];
      memset(seen, 0, sizeof(seen));

      // Find if the applicant 'u' can get a job
      if (bpm(bpGraph, u, seen, matchR))
         result++;
   }
   return result;
}
```

Geometry:

$$area = \frac{s^2 n}{4 \tan\left(\frac{180}{n}\right)}$$

1.Area of a regular polygon(equal sides)

2. Angle between (m1, b1) and (m2, b2):

arctan ((m2 − m1) / (m1 · m2 + 1))

3. Triangle: Area = a · b · sin γ / 2

```
   power(F, n/2);
   multiply(F, F);
   if (n%2 != 0)
      multiply(F, M);
   return F[0][0] + F[0][1] ;
}

LL findNthTerm(LL n)
{
   LL F[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}} ;
   return power(F, n-2);
}
```

Euler’s totient:

Number of integers coprime to n less than n

```
LL phi(LL n)
{
   LL result = n;
   for (LL p=2; p*p<=n; ++p)
   {
      if (n % p == 0)
      {
         while (n % p == 0)
            n /= p;
         result -= result / p;
      }
   }
   if (n > 1)
      result -= result / n;
   return result;
}
```

Largest power of p that divides n!

```
// Returns largest power of p that divides n!
int largestPower(int n, int p)
{
```

• Area = | x1 · y2 + x2 · y3 + x3 · y1 − y1 · x2 − y2 · x3 − y3 · x1 | / 2

• Heron’s formula:

Let s = (a + b + c) / 2; then Area = s · (s − a) · (s − b) · (s − c)

4. Circle: (x − xc)^2+ (y − yc)^2= r^2

5.Polygon area (vertex coordinates):

| x1 · y2 + x2 · y3 + ... + xn · y1 − y1 · x2 − y2 · x3 − ... − yn · x1 | / 2

Orientation:

```
LL orientation(PoLL p1, PoLL p2, PoLL p3)
{
   LL val = (p2.y - p1.y) * (p3.x - p2.x) -
             (p2.x - p1.x) * (p3.y - p2.y);

   if (val == 0) return 0;  // colinear

   return (val > 0)? 1: 2; // clock or counterclock wise
}
```

Line intersection:

```
bool onSegment(PoLL p, PoLL q, PoLL r)
{
   if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
      q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
     return true;
   return false;
}
bool doIntersect(PoLL p1, PoLL q1, PoLL p2, PoLL q2)
{
   LL o1 = orientation(p1, q1, p2);
   LL o2 = orientation(p1, q1, q2);
   LL o3 = orientation(p2, q2, p1);
   LL o4 = orientation(p2, q2, q1);
```

```
    // Initialize result
    int x = 0;

    // Calculate x = n/p + n/(p^2) + n/(p^3) + ....
    while (n)
    {
        n /= p;
        x += n;
    }
    return x;
}
```

nCr (with lucas Theorem):

```
LL ncrp(LL n, LL r, LL p)
{
  LL C[r+1];
  memset(C, 0, sizeof(C));
  C[0] = 1;
  for (LL i = 1; i <= n; i++)
  {
     for ( LL j = min(i, r); j > 0; j--)
        C[j] = (C[j] + C[j-1])%p;
  }
  return C[r];
}
LL ncrpl(LL n,LL r, LL p)
{
 if (r==0)
   return 1;
 int ni = n%p, ri = r%p;
 return (ncrpl(n/p, r/p, p) *
        ncrp(ni, ri, p)) % p;
}
```

Chinese Remainder Theorem

```
    if (o1 != o2 && o3 != o4)
        return true;
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false;}
```

Circle intersection area:

```
int areaOfIntersection(x0, y0, r0, x1, y1, r1){
var rr0 = r0*r0;
var rr1 = r1*r1;
var c = Math.sqrt((x1-x0)*(x1- x0) +(y1-y0)*(y1- y0));
var phi =(Math.acos((rr0+(c*c)-rr1) /(2*r0*c)))*2;
var theta =(Math.acos((rr1+(c*c)-rr0) /(2*r1*c)))*2;
var area1 = 0.5*theta*rr1 - 0.5*rr1*Math.sin(theta);
var area2 = 0.5*phi*rr0 - 0.5*rr0*Math.sin(phi);
return area1 + area2;
}
```

Convex Hull:

```
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
          (p1.y - p2.y)*(p1.y - p2.y);
}
```

```
LL crt(LL num[], LL rem[], LL k)
{
  LL prod = 1;
  for (int i = 0; i < k; i++)
     prod *= num[i];
  LL result = 0;
  for (int i = 0; i < k; i++)
  {
     LL pp = prod / num[i];
     LL inv,y;
     gcde(pp,num[i],&inv,&y);
     result += rem[i] * inv * pp;
  }
  return result % prod;
}
```

For combining wrt a large number, use it 2 numbers at a time.

Wilson’s theorem

((p-1)!)%p=-1

Inclusion-Exclusion:

(A U B)= add 1 at a time, subtract 2 at a time ……

Number of solutions to a linear eqn:

```
LL countSol(LL coeff[], LL start, LL end, LL rhs)
{
   // Base case
   if (rhs == 0)
     return 1;

   LL result = 0;  // Initialize count of solutions

   // One by subtract all smaller or equal coefficiants and recur
   for (LL i=start; i<=end; i++)
```

```
int compare(const void *vp1, const void *vp2)
{
  Point *p1 = (Point *)vp1;
  Point *p2 = (Point *)vp2;
  int o = orientation(p0, *p1, *p2);
  if (o == 0)
    return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;
  return (o == 2)? -1: 1;
}

void convexHull(Point points[], int n)
{
  int ymin = points[0].y, min = 0;
  for (int i = 1; i < n; i++)
  {
    int y = points[i].y;
    if ((y < ymin) || (ymin == y &&
        points[i].x < points[min].x))
       ymin = points[i].y, min = i;
  }
  swap(points[0], points[min]);
  p0 = points[0];
  qsort(&points[1], n-1, sizeof(Point), compare);
  int m = 1;
  for (int i=1; i<n; i++)
  {
     // Keep removing i while angle of i and i+1 is same
     while (i < n-1 && orientation(p0, points[i],
                                   points[i+1]) == 0)
        i++;
     points[m] = points[i];
     m++;
  }
  if (m < 3) return;
  stack<Point> S;
  S.push(points[0]);
```

```
    if (coeff[i] <= rhs)
      result += countSol(coeff, i, end, rhs-coeff[i]);
  return result;
}
```

Sum of GP:

```
long long gp(LL r, LL p,LL m){
if(p==0)
return 1;
if(p==1)
return 1;
LL ans=0;
if(p%2==1){
ans=Mpow(r,p-1,m);
ans=(ans+((1+r)*gp(Mpow(r,2,m),(p-1)/2,m))%m)%m;
}
else{
  ans=((1+r)*gp(Mpow(r,2,m),p/2,m))%m;
}
return ans;
}
```

Ternary Search (max of unimodal function):

```
double ts(double start, double end)
{
  double l = start, r = end;

  for(int i=0; i<200; i++) {
    double l1 = (l*2+r)/3;
    double l2 = (l+2*r)/3;
    //cout<<l1<<" "<<l2<<endl;
    if(func(l1) > func(l2)) r = l2; else l = l1;
    }
  return func(r);
}
```

```
  S.push(points[1]);
  S.push(points[2]);
  for (int i = 3; i < m; i++)
  {
    while (orientation(nextToTop(S), S.top(), points[i]) != 2)
      S.pop();
    S.push(points[i]);
  }
  while (!S.empty())
  {
     Point p = S.top();
     cout << "(" << p.x << ", " << p.y <<")" << endl;
     S.pop();
  }
}
```

Point in a polygon:

```
bool isInside(Point polygon[], int n, Point p)
{
  if (n < 3)  return false;
  Point extreme = {INF, p.y};
  int count = 0, i = 0;
  do
  {
     int next = (i+1)%n;
     if (doIntersect(polygon[i], polygon[next], p, extreme))
     {
        if (orientation(polygon[i], p, polygon[next]) == 0)
          return onSegment(polygon[i], p, polygon[next]);

        count++;
     }
     i = next;
  } while (i != 0);
  return count&1;  // Same as (count%2 == 1)
```

Data Structures:

Iterative trie:

```
int trie[MAX_N * 30][3], nxt;
void trie_init(int n) {
   int nn = (n+2)*30;
   for(int i=0; i<nn; i++)
      trie[i][0] = trie[i][1] = trie[i][2] = -1;
   nxt = 1;
}

void trie_insert(int v, int x) {
   int cur = 0;
   for(int i=29; i>=0; i--) {
      int bit = v>>i & 1;
      if(trie[cur][bit]==-1)
         trie[cur][bit] = nxt++;
      cur = trie[cur][bit];
      trie[cur][2] = max(trie[cur][2], x);
   }
}

int trie_getmax(int v, int m) {
   int cur = 0, mx = -1;
   for(int i=29; i>=0; i--) {
      int bit = v>>i & 1;
      if(m>>i & 1)
         cur = trie[cur][!bit];
      else {
         int lt = trie[cur][!bit];
         if(lt!=-1) mx = max(mx, trie[lt][2]);
         cur = trie[cur][bit];
      }
      if(cur==-1) break;
   }
}
```

Game Theory:

1. If nim-sum is non-zero, player starting first wins.
2. Mex: smallest non-negative number not present in a set.
3. Grundy=0 means game lost.
4. Grundy=mex of all possible next states.
5. Sprague-Grundy theorem:

If a game consists of sub games (nim with multiple piles)
Calculate grundy number of each sub game (each pile)
Take xor of all grundy numbers:
If non-zero, player starting first wins.

Pattern Matching:

Suffix Arrays:

```
struct suffix
{
   int index; // To store original index
   int rank[2]; // To store ranks and next rank pair
};
int cmp(struct suffix a, struct suffix b)
{
   return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank[1] ?1: 0):
            (a.rank[0] < b.rank[0] ?1: 0);
}
int *buildSuffixArray(char *txt, int n)
{
   struct suffix suffixes[n];
   for (int i = 0; i < n; i++)
   {
      suffixes[i].index = i;
      suffixes[i].rank[0] = txt[i] - 'a';
      suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] - 'a'): -1;
```

```
  if(cur!=-1) mx = max(mx, trie[cur][2]);
  return mx;
}
```

Iterative segment tree:

```
void build() {
 for (LL i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];}

void modify(LL p, LL value) {  // set value at position p
 for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];}

LL query(LL l, LL r) {  // sum on LLerval [l, r)
 LL res = 0;
 for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
  if (l&1) res += t[l++];
  if (r&1) res += t[--r];
 }
 return res;
}
```

Lazy Segment tree

```
LL lconstruct(LL *a,LL *st,LL ss,LL se,LL si)
{
  if(ss==se)
  {
    st[si]=a[ss];
    return st[si];
  }
  LL mid=ss+(se-ss)/2;
  st[si]=(lconstruct(a,st,ss,mid,si*2+1)+lconstruct(a,st,mid+1,se,si*2+2));
  return st[si];
}

LL lgs(LL *st,LL l,LL r,LL ss,LL se,LL si,LL *lazy)
{
  if(lazy[si])
```

```
}
sort(suffixes, suffixes+n, cmp);
int ind[n];
for (int k = 4; k < 2*n; k = k*2)
{
  int rank = 0;
  int prev_rank = suffixes[0].rank[0];
  suffixes[0].rank[0] = rank;
  ind[suffixes[0].index] = 0;
  for (int i = 1; i < n; i++)
  {
    if (suffixes[i].rank[0] == prev_rank &&
        suffixes[i].rank[1] == suffixes[i-1].rank[1])
    {
      prev_rank = suffixes[i].rank[0];
      suffixes[i].rank[0] = rank;
    }
    else
    {
      prev_rank = suffixes[i].rank[0];
      suffixes[i].rank[0] = ++rank;
    }
    ind[suffixes[i].index] = i;
  }
  for (int i = 0; i < n; i++)
  {
    int nextindex = suffixes[i].index + k/2;
    suffixes[i].rank[1] = (nextindex < n)?
                  suffixes[ind[nextindex]].rank[0]: -1;
  }
  sort(suffixes, suffixes+n, cmp);
}

 // Store indexes of all sorted suffixes in the suffix array
int *suffixArr = new int[n];
for (int i = 0; i < n; i++)
  suffixArr[i] = suffixes[i].index;
```

```
    //same as update
    if(ss>r||se<l||ss>se)
    return 0;
    if(l<=ss&&r>=se)
    {
    return st[si];
    }
    LL mid=ss+(se-ss)/2;
    return (lgs(st,l,r,ss,mid,si*2+1,lazy)+lgs(st,l,r,mid+1,se,si*2+2,lazy));
}


void lupdate(LL *st,LL ss,LL se,LL ql,LL qr,LL diff,LL si,LL *lazy)
{
    if(lazy[si])
    {
        st[si]=(st[si]+(se-ss+1)*lazy[si]);
        if(ss!=se)
        {
            lazy[si*2+1]=(lazy[si*2+1]+lazy[si]);
            lazy[si*2+2]=(lazy[si*2+2]+lazy[si]);
        }
        lazy[si]=0;
    }
    if(ss>se||qr<ss||ql>se)
    return;
    if(ss>=ql&&se<=qr)
    {
        st[si]=(st[si]+(se-ss+1)*diff);
        if(ss!=se)
        {
            lazy[si*2+1]=(lazy[si*2+1]+diff);
            lazy[si*2+2]=(lazy[si*2+2]+diff);
        }
        return;
    }
    if(ss!=se)
```

```
    return  suffixArr;
}

void search(char *pat, char *txt, int *suffArr, int n)
{
    int m = strlen(pat);
    int l = 0, r = n-1;
    while (l <= r)
    {
        int mid = l + (r - l)/2;
        int res = strncmp(pat, txt+suffArr[mid], m);
        if (res == 0)
        {
            cout << "Pattern found at index " << suffArr[mid];
            return;
        }
        if (res < 0) r = mid - 1;
        else l = mid + 1;
    }
    cout << "Pattern not found";
}
```

KMP Algorithm(STL):

```
std::size_t found = a.find(b, 0);
while(found != std::string::npos) {
  std::cout << "found!" << '\n';
  found = a.find(b, found+1);
}
```

KMP Algorithm(STL):

KMP b stores the string(pattern)
we need to find it occurrences in string a.
and vector v stores occurrences of b in a

```
   {
      LL mid=ss+(se-ss)/2;
      lupdate(st,ss,mid,ql,qr,diff,si*2+1,lazy);
      lupdate(st,mid+1,se,ql,qr,diff,si*2+2,lazy);
   }
   st[si]=(st[2*si+1]+st[2*si+2]);
}
```

Policy based DS:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> pbds;
insert(val),erase(),order_of_key(),find_by_order()
```

Union-Find:

```
LL find(struct subset subsets[], LL i)
{
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}
void Union(struct subset subsets[], LL x, LL y)
{
   LL xroot = find(subsets, x);
   LL yroot = find(subsets, y);
   // Attach smaller rank tree under root of high rank tree
   if (subsets[xroot].rank < subsets[yroot].rank)
      subsets[xroot].parent = yroot;
   else if (subsets[xroot].rank > subsets[yroot].rank)
      subsets[yroot].parent = xroot;
   else
   {
      subsets[yroot].parent = xroot;
      subsets[xroot].rank++;
```

```
void kmp(string a, string b){
   vector<ll> v;
   ll n = a.length() ,  m = b.length();
   /*  Compute temporary array pre[m] to maintain
   size of suffix which is same as prefix */
   ll pre[m] , i=1, j=0;
   pre[0] = 0;
   while(i<m) {
     if(b[i]==b[j])
       pre[i] = j+1, i++, j++;
     else if(b[i]!=b[j]){
       if(j==0) pre[i]=0, i++;
       else j = pre[j-1];
     }}
   i=0, j=0;
   /* Search for pattern in text.  */
   while(i<n) {
     if(a[i]==b[j]){
       i++, j++;
       if(j==m){
         v.pb(i+1-m);
         j = pre[j-1];
       }}
     else{
       if(j==0) i++;
       else j =pre[j-1];
     }}}
   }
}
```

Graph Theory

Dijkstra’s Algorithm:

```
void Dijkstra(LL src,LL V)
{
   set< pair<LL, LL> > setds;
   vector<LL> dist(V, INF);
   setds.insert(make_pair(0, src));
   dist[src] = 0;
   while (!setds.empty())
   {
      pair<int, int> tmp = *(setds.begin());
      setds.erase(setds.begin());
      int u = tmp.second;
      vector< pair<int, int> >::iterator i;
      for (i = adj[u].begin(); i != adj[u].end(); ++i)
      {
         int v = (*i).first;
         int weight = (*i).second;
         if (dist[v] > dist[u] + weight)
         {
            if (dist[v] != INF)
               setds.erase(setds.find(make_pair(dist[v], v)));
            dist[v] = dist[u] + weight;
            setds.insert(make_pair(dist[v], v));
         }
     }
    }
}
```

Standard DP

LCS:

```
void lcs( char *X, char *Y, LL m, LL n )
{
  LL L[m+1][n+1];
  for (LL i=0; i<=m; i++)
  {
    for (LL j=0; j<=n; j++)
    {
     if (i == 0 || j == 0)
       L[i][j] = 0;
     else if (X[i-1] == Y[j-1])
       L[i][j] = L[i-1][j-1] + 1;
     else
       L[i][j] = max(L[i-1][j], L[i][j-1]);
    }
  }
  // Following code is used to prLL LCS
  LL index = L[m][n];
  char lcs[index+1];
  lcs[index] = '\0'; // Set the terminating character
  LL i = m, j = n;
  while (i > 0 && j > 0)
  {
    if (X[i-1] == Y[j-1])
    {
       lcs[index-1] = X[i-1]; // Put current character in result
       i--; j--; index--;     // reduce values of i, j and index
    }
    else if (L[i-1][j] > L[i][j-1])
      i--;
    else
      j--;
```

Floyd Warshall(All pair)

```
for (k = 0; k < V; k++)
        for (i = 0; i < V; i++)
            for (j = 0; j < V; j++)
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
```

Bellman-Ford(for negative edges):

```
void BellmanFord(struct Graph* graph, LL src)
{
    LL V = graph->V;
    LL E = graph->E;
    LL dist[V];
    for (LL i = 0; i < V; i++)
        dist[i]   = INT_MAX;
    dist[src] = 0;
    for (LL i = 1; i <= V-1; i++)
    {
        for (LL j = 0; j < E; j++)
        {
            LL u = graph->edge[j].src;
            LL v = graph->edge[j].dest;
            LL weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }//to check for negative weight cycle, repeat above
}   // if shorter path is found, cycle exists
```

Prim's Algorithm for MST

```
void primMST()
```

```
  }
  cout << "LCS of " << X << " and " << Y << " is " << lcs;
}
```

Max contiguous subarray sum (Kadane's Algo):

```
LL maxSubArraySum(LL a[], LL size)
{
  LL max_so_far = a[0];
  LL curr_max = a[0];

  for (LL i = 1; i < size; i++)
  {
       curr_max = max(a[i], curr_max+a[i]);
       max_so_far = max(max_so_far, curr_max);
  }
  return max_so_far;
}
```

LIS in nlogn:

```
LL CeilIndex(std::vector<LL> &v, LL l, LL r, LL key) {
    while (r-l > 1) {
    LL m = l + (r-l)/2;
    if (v[m] >= key)
        r = m;
    else
        l = m;
    }
    return r;
}

LL LongestIncreasingSubsequenceLength(std::vector<LL> &v) {
    if (v.size() == 0)
        return 0;
```

```
{
   priority_queue<pair<LL,LL>,greater<pair<LL,LL>>> pq;
   LL src = 0;
   vector<LL> key(V, INF);
   vector<LL> parent(V, -1);
   vector<bool> inMST(V, false);
   pq.push(make_pair(0, src));
   key[src] = 0;
   while (!pq.empty())
   {
      LL u = pq.top().second;
      pq.pop();
      inMST[u] = true;  // Include vertex in MST
      list< pair<LL, LL> >::iterator i;
      for (i = adj[u].begin(); i != adj[u].end(); ++i)
      {
         LL v = (*i).first;
         LL weight = (*i).second;
         if (inMST[v] == false && key[v] > weight)
         {
            key[v] = weight;
            pq.push(make_pair(key[v], v));
            parent[v] = u;
         }
      }}}
```

LCA:

Pre-processing: O(nlogn) , Query: O(logn)

```
vector <int> tree[MAXN];
int depth[MAXN];
int parent[MAXN][level];
// pre-compute  depth for each node and their first parent(2^0th parent)
void dfs(int cur, int prev){
   depth[cur] = depth[prev] + 1;
   parent[cur][0] = prev;
```

```
   std::vector<LL> tail(v.size(), 0);
   LL length = 1; // always poLLs empty slot in tail

   tail[0] = v[0];
   for (size_t i = 1; i < v.size(); i++) {
      if (v[i] < tail[0])
         tail[0] = v[i];
      else if (v[i] > tail[length-1])
         tail[length++] = v[i];
      else
         tail[CeilIndex(tail, -1, length-1, v[i])] = v[i];
   }

   return length;
}
```

Coin Change Problem:

```
int count( int S[], int m, int n )
{
   int table[n+1];
   memset(table, 0, sizeof(table));

   // Base case (If given value is 0)
   table[0] = 1;
   for(int i=0; i<m; i++)
      for(int j=S[i]; j<=n; j++)
         table[j] += table[j-S[i]];

   return table[n];
}
```

Rod Cutting Problem:

```
LL cutRod(LL price[], LL n)
{
  LL val[n+1];
```

```
    for (int i=0; i<tree[cur].size(); i++) {
        if (tree[cur][i] != prev)
            dfs(tree[cur][i], cur);
    }
}
void precomputeSparseMatrix(int n){
    for (int i=1; i<level; i++){
        for (int node = 1; node <= n; node++){
            if (parent[node][i-1] != -1)
          parent[node][i]=parent[parent[node][i-1]][i-1];
        } }}
int lca(int u, int v){
    if (depth[v] < depth[u]) swap(u, v);
    int diff = depth[v] - depth[u];
    for (int i=0; i<level; i++)
        if ((diff>>i)&1)
            v = parent[v][i];
    if (u == v) return u;
    for (int i=level-1; i>=0; i--)
        if (parent[u][i] != parent[v][i]){
            u = parent[u][i];
            v = parent[v][i];
        }
    return parent[u][0];
}
```

Topological Sort:

```
void topologicalSortUtil(LL v, bool visited[],
                                stack<LL> &Stack)
{
    visited[v] = true;
    list<LL>::iterator i;
```

```
  val[0] = 0;
  LL i, j;

  // Build the table val[] in bottom up manner and return the last entry
  // from the table
  for (i = 1; i<=n; i++)
  {
     LL max_val = INT_MIN;
     for (j = 0; j < i; j++)
       max_val = max(max_val, price[j] + val[i-j-1]);
     val[i] = max_val;
  }

  return val[n];}
```

Sum Of Subset:

```
bool isSubsetSum(LL set[], LL n, LL sum)
{
   bool subset[n+1][sum+1];
   for (LL i = 0; i <= n; i++)
     subset[i][0] = true;
   for (LL i = 1; i <= sum; i++)
     subset[0][i] = false;
    for (LL i = 1; i <= n; i++)
    {
      for (LL j = 1; j <= sum; j++)
      {
        if(j<set[i-1])
        subset[i][j] = subset[i-1][j];
        if (j >= set[i-1])
          subset[i][j] = subset[i-1][j] ||
                                 subset[i - 1][j-set[i-1]];
      }
    }
    return subset[n][sum];
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            topologicalSortUtil(*i, visited, Stack);
    Stack.push(v);
}

void topologicalSort()
{
    stack<LL> Stack;
    bool *visited = new bool[V];
    for (LL i = 0; i < V; i++)
        visited[i] = false;
    for (LL i = 0; i < V; i++)
      if (visited[i] == false)
        topologicalSortUtil(i, visited, Stack);


    while (Stack.empty() == false)
    {
        cout << Stack.top() << " ";
        Stack.pop();
    }
}
```

Manacher's Algorithm:

return longest palindromic substring in O(n).

```
string manacher(string s){
    ll len = s.length();
    string ne = "@";

    fr(i,len)
```

```
}
```

Catalan numbers:

1, 1, 2, 5, 14, 42, 132, 429, 1430,........

C(n) =(1/(n+1)) * choose(2n, n);

C(n+1) = Summation(i = 0 to n) [C(i) * C(n-i)]

0/1 Knapsack:

```
LL knapSack(LL W, LL wt[], LL val[], LL n)
{
   LL i, w;
   LL K[n+1][W+1];
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }
   return K[n][W];
}
```

Egg Drop Problem:

```
LL eggDrop(LL n, LL k)
{
    LL eggFloor[n+1][k+1];
    LL res;
    LL i, j, x;
    for (i = 1; i <= n; i++)
```

```
        ne+= "#"+s[i] ;
    ne += "#$";

    len = ne.size();

    ll p[len+1] = {0}, c=0,r=0;
    fre(i,len-2){
        ll imirror = 2*c-i;
        if(r>i)  p[i] = min(r-i, p[imirror]);
        while(ne[i+1+p[i]]==ne[i-1-p[i]])     p[i]++;
        if(i+p[i]>r)   c=i, r = i+p[i];
    }
    ll mlen = 0, cind = 0;
    fre(i,len-2) {
        if(p[i]>mlen) mlen = p[i], cind = i;
    }
    return s.substr((cind-mlen-1)/2, mlen);
}
```

Z Algorithm:

O(c.length() + s.length())
String c need to be find out in string s;
z[i] stores the maximum length of substring starting from ith position which is prefix of a.
We need to find how many times z[i] = c.length()
a = c+'&' + s where & is character that is not present in either of the strings.

```
void zalgo(string s, string c ){
     string a = c+"#"+ s;
      ll n = a.length();
      ll z[n+1], l=0,r=0,k ;
      z[0] = 0;
```

```
    {
        eggFloor[i][1] = 1;
        eggFloor[i][0] = 0;
    }
    // We always need j trials for one egg and j floors.
    for (j = 1; j <= k; j++)
        eggFloor[1][j] = j;
    for (i = 2; i <= n; i++)
    {
        for (j = 2; j <= k; j++)
        {
            eggFloor[i][j] = INT_MAX;
            for (x = 1; x <= j; x++)
            {
                res = 1 + max(eggFloor[i-1][x-1], eggFloor[i][j-x]);
                if (res < eggFloor[i][j])
                    eggFloor[i][j] = res;
            }
        }
    }
    return eggFloor[n][k];
}
```

Cap Assignment (bit-mask):

```
long long int countWaysUtil(int mask, int i)
{
    if (mask == allmask) return 1;
    if (i > 100) return 0;
    if (dp[mask][i] != -1) return dp[mask][i];
    long long int ways = countWaysUtil(mask, i+1);
    int size = capList[i].size();
    for (int j = 0; j < size; j++)
    {
        if (mask & (1 << capList[i][j])) continue;
        else ways += countWaysUtil(mask | (1 << capList[i][j]), i+1);
        ways %= MOD;
fre(i,n-1){
  if(i>r){
    l = r = i;
    while(r<n && a[r]==a[r-l]) r++;
    z[i] = r-l;
    r--;
  }
  else {
    k = i-l;
    if(z[k]< r-i+1)  z[i] = z[k];
    else{
      l =  i;
      while(r<n && a[r]==a[r-l] )r++;
       z[i] = r-l;
       r--;
    }} }
ll m = c.length(), ans=0;
fre(i,n-1)
{
  if(z[i]== m)
    ans++;
}}
```

```
  }
  return dp[mask][i] = ways;
}
```

Points to Remember before submitting:-

1. Use mod

2. Check overflows- array bound

3. Don’t sort vector if empty

4. Don't pop stack etc if empty

FINAL WORDS TO REMEMBER:-

You find that you have peace of mind and can enjoy yourself, get more sleep, and rest when you know that it was a one hundred percent effort that you gave — win or lose. When the game is over I just want to look at myself in the mirror, win or lose, and know I gave it everything I had.Success is not final, failure is not fatal: it is the courage to continue that counts.Never lose hope.Whatever be the situation. I know our competitors are Red rated or what not, but let’s fight them graciously till last and learn and gather wonderful experiences for times to come. SO GEAR UP and GET! SET! Go!!!!