# Little Piplup

Contest Teamnote
UCPC 2019 ver

# Little Piplup

Gratus907(Wonseok Shin), Coffeetea(Jinje Han), DhDroid(Donghyun Son)

## Contents

**1 Settings** **3**
1.1 C++ . . . . . . 3

**2 Data Structures** **3**
2.1 Segment Tree . . . . . . 3
2.2 Disjoint Set Union (Union - Find) . . . . 4

**3 Mathematics** **4**
3.1 Useful Mathematical Formula . . . . . . 4
3.2 Number of Integer Partition . . . . . . 4
3.3 Binomial Coefficient . . . . . . 4
3.4 Extended Euclidean Algorithm . . . . . . 4
3.5 Fast Modulo Exponentiation . . . . . . 4
3.6 Miller-Rabin Primality Testing . . . . . . 5
3.7 Pollard-Rho Factorization . . . . . . 5
3.8 Euler Totient . . . . . . 6
3.9 Modular Multiplicative Inverse . . . . . . 6

**4 Geometry** **6**
4.1 CCW . . . . . . 6
4.2 Point in polygon . . . . . . 6
4.3 Closest Pair Problem . . . . . . 7
4.4 Convex Hull (Graham Scan) . . . . . . 7
4.5 Intersection of Line Segment . . . . . . 8

**5 Graphs** **9**
5.1 Topological Sorting . . . . . . 9
5.2 Lowest Common Ancestor . . . . . . 9
5.3 MST Kruskal Algorithm . . . . . . 10
5.4 MST Prim Algorithm . . . . . . 10
5.5 Dinic's Algorithm . . . . . . 11

**6 Shortest Path** **12**
6.1 Dijkstra . . . . . . 12
6.2 Bellman Ford . . . . . . 12
6.3 SPFA Algorithm . . . . . . 13
6.4 Floyd-Warshall . . . . . . 13

**7 Dynamic** **13**
7.1 Longest Increasing Subsequence . . . . . . 13
7.2 Largest Sum Subarray . . . . . . 14
7.3 0-1 Knapsack . . . . . . 14
7.4 Longest Common Subsequence . . . . . . 14

**8 String** **14**
8.1 KMP Algorithm . . . . . . 14
8.2 Manacher's Algorithm . . . . . . 15
8.3 Trie . . . . . . 15

**9 Miscellaneous** **16**
9.1 Useful Bitwise Functions in C++ . . . . . . 16
9.2 List of Useful Numbers . . . . . . 16

**10 Checkpoints** **17**
10.1 Debugging . . . . . . 17
10.2 Thinking . . . . . . 17

# 1 Settings

## 1.1 C++

# 2 Data Structures

## 2.1 Segment Tree

To deal with queries on intervals, we use segment tree.

```
int arr[SIZE];
int tree[TREE_SIZE];
int makeTree(int left,int right,int node)
{
    if (left == right)
        return tree[node] = arr[left];
    int mid = (left + right) / 2;
    tree[node] += makeTree(left, mid, node * 2);
    tree[node] += makeTree(mid + 1,right, node * 2 +1);
    return tree[node];
}
```

Updating segment tree

```
void update(int left,int right,int node, int change_node ,int diff)
{
    if (!(left <= change_node &&change_node <= right))
        return; //No effect on such nodes.
    tree[node] += diff; // This part must be changed with tree function.
    if (left != right)
    {
        int mid = (left + right) / 2;
        update(left, mid, node * 2, change_node, diff);
        update(mid +1,right, node * 2 +1, change_node, diff);
    }
}
```

Answering queries with segment tree

```
/*
Our Search range : start to end
Node has range left to right
We may answer query in O(log n) time.
*/
int Query(int node, int left, int right, int start, int end)
{
    if (right < start || end < left)
        return 0; //Node is out of range
    if (start <= left && right <= end)
        return tree[node]; //If node is completely in range
    int mid = (left + right) / 2;
    return Query(node * 2, left, mid, start, end)
    +Query(node*2+1,mid+1,right,start,end);
}
```

Answering range minimum queries with segment tree

```
struct Range_Minimum_Tree
{
    int n;
    vector<int> segtree;

    Range_Minimum_Tree(const vector<int> &data)
    {
        n = data.size();
        segtree.resize(4 * n);
        initialize(data, 0, n - 1, 1);
    }

    int initialize(const vector<int> &data, int l, int r, int node)
    {
        if (l == r)
            return segtree[node] = data[l];
        int mid = (l + r) / 2;
        int lmin = initialize(data, l, mid, node * 2);
        int rmin = initialize(data, mid + 1, r, node * 2 + 1);
        return segtree[node] = min(lmin, rmin);
    }

    int minq(int l, int r, int node, int nodeleft, int noderight)
    {
        if (r < nodeleft || noderight < l)
            return INT_MAX;
        if (l <= nodeleft && noderight <= r)
            return segtree[node];
        int mid = (nodeleft + noderight) / 2;
        return min(minq(l,r,node*2,nodeleft,mid),
        minq(l,r,node*2+1,mid+1,noderight));
    }
};
```

## 2.2 Disjoint Set Union (Union - Find)

```
// Original Author : Ashishgup
struct Disjoint_Set_Union
{
    int connected;
    int parent[V], size[V];

    void init(int n)
    {
        for(int i=1;i<=n;i++)
        {
            parent[i]=i;
            size[i]=1;
        }
        connected=n;
    }
    int Find(int k)
    {
        while(k!=parent[k])
        {
            parent[k]=parent[parent[k]];
            k=parent[k];
        }
        return k;
    }
    int getSize(int k)
    {
        return size[Find(k)];
    }
    void unite(int x, int y)
    {
        int u=Find(x), v=Find(y);
        if(u==v)
            return;
        if(size[u]>size[v])
            swap(par1, par2);
        size[v]+=size[u];
        size[u] = 0;
        parent[u] = parent[v];
    }
} dsu;
```

# 3 Mathematics

## 3.1 Useful Mathematical Formula

- Catalan Number : Number of valid parantheses strings with $n$ pairs

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

- Nim Game : Remember - XOR of all piles.

## 3.2 Number of Integer Partition

```
def partitions(n):
    parts = [1]+[0]*n
    for t in range(1, n+1):
        for i, x in enumerate(range(t, n+1)):
            parts[x] += parts[i]
    return parts[n]
```

## 3.3 Binomial Coefficient

Fast-to-Type Binomial coefficient

## 3.4 Extended Euclidean Algorithm

```
int Extended_Euclid(int a, int b, int *x, int *y)
{
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int EEd = Extended_Euclid(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;
    return EEd;
}
```

## 3.5 Fast Modulo Exponentiation

Calculating $x^y \bmod p$ in $\mathcal{O}(\log y)$ time.

```
/*
Fast Modulo Exponentiation algorithm
Runs on O(log y) time,
calculate x^y mod p
*/

ll modpow(ll x, ll y, ll p)
{
    ll res = 1;
    x = x % p;
    while (y > 0)
    {
        if (y & 1)
            res = (res*x) % p;
        y = y>>1;
        x = (x*x) % p;
    }
    return res;
}
```

## 3.6 Miller-Rabin Primality Testing

Base values of $a$ chosen so that results are tested to be correct up to $10^14$.

```
bool MRwitness(ll n, ll s, ll d, ll a)
{
    ll x = modpow(a, d, n);
    ll y = -1;

    while (s)
    {
        y = (x * x) % n;
        if (y == 1 && x != 1 && x != n-1)
            return false;
        x = y;
        s--;
    }
    return (y==1);
}

bool Miller_Rabin(ll n)
{
    if (n<2)
        return false;
    if (n == 2 || n == 3 || n == 5 || n == 7 ||
     n == 11 || n == 13 || n == 17)
        return true;
    if (n%2 == 0 || n%3 == 0 || n%5 == 0)
        return false;
    ll d = (n-1) / 2;
    ll s = 1;
    while (d%2==0)
    {
        d /= 2;
        s++;
    }
    int candidate[7] = {2,3,5,7,11,13,17};
    bool result = true;
    for (auto i : candidate)
    {
        result = result & MRwitness(n,s,d,i);
        if (!result)
            break;
    }
    return result;
}
```

## 3.7 Pollard-Rho Factorization

```
ll PollardRho(ll n)
{
    srand (time(NULL));
    if (n==1)
        return n;
    if (n % 2 == 0)
        return 2;
    ll x = (rand()%(n-2))+2;
    ll y = x;
    ll c = (rand()%(n-1))+1;
    ll d = 1;
    while (d==1)
    {
        x = (modpow(x, 2, n) + c + n)%n;
        y = (modpow(y, 2, n) + c + n)%n;
        y = (modpow(y, 2, n) + c + n)%n;
        d = gcd(abs(x-y), n);
        if (d==n)
            return PollardRho(n);
    }
    return d;
}
```

### 3.8 Euler Totient

Calculating number of integers below $n$ which is coprime with $n$.

```
#define ll long long
ll euler_phi(ll n)
{
    ll p=2;
    ll ephi = n;
    while(p*p<=n)
    {
        if (n%p == 0)
            ephi = ephi/p * (p-1);
        while(n%p==0)
            n/=p;
        p++;
    }
    if (n!=1)
    {
        ephi /= n;
        ephi *= (n-1);
    }
    return ephi;
}
```

### 3.9 Modular Multiplicative Inverse

```
ll modinv(ll x, ll p)
{
    return modpow(x,p-2,p);
}
```

# 4 Geometry

## 4.1 CCW

```
//Is 3 points Counterclockwise? 1 : -1
//0 : on same line
int CCW(Point a, Point b, Point c)
{
    int op = a.x*b.y + b.x*c.y + c.x*a.y;
    op -= (a.y*b.x + b.y*c.x + c.y*a.x);
    if (op > 0)
        return 1;
    else if (op == 0)
        return 0;
    else
        return -1;
}
```

## 4.2 Point in polygon

Returns boolean, if point is in the polygon (represented as vector of points).

```
// point in polygon test
inline double is_left(Point p0, Point p1, Point p2)
{
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
bool is_in_polygon(Point p, vector<Point>& poly)
{
    int wn = 0;
    for (int i = 0; i < poly.size(); ++i)
    {
        int ni = (i + 1 == poly.size()) ? 0 : i + 1;
        if (poly[i].y <= p.y)
        {
            if (poly[ni].y > p.y)
                if (is_left(poly[i], poly[ni], p) > 0)
                    ++wn;
        }
        else
        {
            if (poly[ni].y <= p.y)
                if (is_left(poly[i], poly[ni], p) < 0)
                    --wn;
        }
    }
    return wn != 0;
}
```

## 4.3 Closest Pair Problem

Requires : Points must be sorted with x-axis.
Runs in $\mathcal{O}(n \log^2 n)$

```
int dist (Point &p, Point &q)
{
    return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y);
}

bool compare(Point &p, Point &q)
{
    return (p.x < q.x);
}

bool ycompare(Point &p, Point &q)
{
    return (p.y<q.y);
}
Point pts[101010];

int closest_pair(Point p[], int n)
{
    //printf("%p call %d\n",p,n);
    if (n==2)
    {
        return dist(p[0], p[1]);
    }
    if (n==3)
    {
        return min(dist(p[0],p[1]),
        min(dist(p[1],p[2]),dist(p[0],p[2])));
    }
    Point mid[n];
    int line = (p[n/2 - 1].x + p[n/2].x) / 2;
    int d = min(closest_pair(p, n/2), closest_pair(p + n/2, n - n/2));
    int pp = 0;
    for (int i = 0; i < n; i++)
    {
        int t = line - p[i].x;
        if (t*t < d)
        {
            mid[pp] = p[i];
            pp++;
        }
    }
    sort(mid,mid+pp,ycompare);
    for (int i = 0; i < pp - 1; i++)
        for (int j = i + 1; j < pp && mid[j].y - mid[i].y < d; j++)
            d = min(d, dist(mid[i], mid[j]));
    return d;
}
```

## 4.4 Convex Hull (Graham Scan)

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

int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

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

int orientation(Point p, Point q, Point r) // Basically CCW
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);
    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

int compare(const void *vp1, const void *vp2)
{
   Point *p1 = (Point *)vp1;
   Point *p2 = (Point *)vp2;

   // Find orientation
   int o = orientation(p0, *p1, *p2);
   if (o == 0)
     return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

   return (o == 2)? -1: 1;
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
   // Find the bottommost point
   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;
   }

   // Place the bottom-most point at first position
   swap(points[0], points[min]);

   // Sort n-1 points with respect to the first point.
   p0 = points[0];
   qsort(&points[1], n-1, sizeof(Point), compare);

   // If two or more points make same angle with p0,
   // Remove all but the one that is farthest from p0
   int m = 1;
   for (int i=1; i<n; i++)
   {
       while (i < n-1 && orientation(p0, points[i],
                                    points[i+1]) == 0)
          i++;
       points[m] = points[i];
       m++;
   }

   // If modified array of points has less than 3 points,
   // convex hull is not possible
   if (m < 3) return;

   // Create an empty stack and push first three points
   // to it.
   stack <Point> S;
   S.push(points[0]);
   S.push(points[1]);
   S.push(points[2]);

   // Process remaining n-3 points
   for (int i = 3; i < m; i++)
   {
      while (orientation(nextToTop(S), S.top(), points[i]) != 2)
         S.pop();
      S.push(points[i]);
   }

   // Now stack has the output points, print contents of stack
   while (!S.empty())
   {
       Point p = S.top();
       cout << "(" << p.x << ", " << p.y <<")" << endl;
       S.pop();
   }
}
```

### 4.5 Intersection of Line Segment

```
//jason9319.tistory.com/358. modified
int isIntersect(Point a, Point b, Point c, Point d)
{
    int ab = ccw(a, b, c)*ccw(a, b, d);
    int cd = ccw(c, d, a)*ccw(c, d, b);
    if (ab == 0 && cd == 0)
    {
        if (a > b)swap(a, b);
        if (c > d)swap(c, d);
        return (c <= b&&a <= d);
    }
    return (ab <= 0 && cd <= 0);
}
```

# 5 Graphs

## 5.1 Topological Sorting

Topological sorting with dfs

```
vector <int> graph[V];
bool visited[V];
vector <int> sorted;

void dfs(int root)
{
    visited[root] = 1;
    for (auto it:graph[root])
    {
        if (!visited[it])
            dfs(it);
    }
    sorted.push_back(root);
}

int main()
{
    int n, m;
    scanf("%d%d",&n,&m);
    for (int i = 0; i<m; i++)
    {
        int small,big;
        scanf("%d%d",&small,&big);
        graph[small].push_back(big);
    }
    for (int i = 1; i<=n; i++)
        if (!visited[i])
            dfs(i);
    reverse(sorted.begin(),sorted.end()); // must reverse!
}
```

## 5.2 Lowest Common Ancestor

LCA Algorithm by sparse table

```
//modified from jason9319's original code.
int n, m;
bool visited[101010];
int par[101010][21];
int d[101010];
```

```
vector <int> graph[101010];
void dfs(int here,int depth) // run dfs(root,0)
{
    visited[here] = true;
    d[here] = depth;
    for (int there : graph[here])
    {
        if (visited[there])
            continue;
        par[there][0] = here;
        dfs(there, depth + 1);
    }
}

void precomputation()
{
    for (int i = 1; i<21; i++)
    {
        for (int j = 1; j<=n; j++)
        {
            par[j][i] = par[par[j][i-1]][i-1];
        }
    }
}

int lca(int x, int y)
{
    if (d[x]>d[y])
        swap(x,y);
    for (int i = 20; i>=0; i--)
    {
        if (d[y]-d[x] >= (1<<i))
        {
            y = par[y][i];
        }
    }
    if (x==y)
        return x;
    for (int i = 20; i>=0; i--)
    {
        if (par[x][i] != par[y][i])
        {
            x = par[x][i];
            y = par[y][i];
        }
    }
    return par[x][0];
}
```

### 5.3 MST Kruskal Algorithm

Based on Union-Find implementation
$\mathcal{O}(E \log E)$ if path-compressed Union Find.

```
int Kruskal()
{
    int mstlen = 0;
    sort(edgelist.begin(),edgelist.end());
    for (auto it:edgelist)
    {
        if (Find(it.s)==Find(it.e)) // Cycle Detection
            continue;
        else
        {
            Union(it.s,it.e);
            mstlen += it.w;
        }
    }
    return mstlen;
}
```

### 5.4 MST Prim Algorithm

```
vector <pii> Tree[101010];
// Note that we use {weight, destination} pair here.
// This is to use priority_queue!
bool visit[101010];
priority_queue <pii, vector<pii>, greater<pii>> pq;
void add(int i)
{
    visit[i] = true;
    for (auto it:Tree[i])
        pq.push(it);
}

int Prim(int start)
```

```
{
    int mstlen = 0;
    add(start);
    while(!pq.empty())
    {
        int cur = pq.top().second;
        int weight = pq.top().first;
        pq.pop();
        if (visit[cur])
            continue;
        else
        {
            mstlen+=weight;
            add(cur);
        }
    }
    return mstlen;
}
```

### 5.5 Dinic's Algorithm

```
//Original Author : https://plzrun.tistory.com/

int r[V][V]; // flow capacity
bool chk[V][V]; // edge existence
int level[V];
vector<int> v[V];
queue<int> q;

bool bfs(int src, int sink)
{
    memset(level,-1,sizeof(level));
    level[src]=0;
    q.push(src);
    while(!q.empty())
    {
        int x = q.front();
        q.pop();
        for(int y: v[x])
        {
            if(r[x][y]>0 && level[y]<0) {
                level[y]=level[x]+1;
                q.push(y);
            }
        }
    }
    return level[sink]>=0;
}

int work[V];

int dfs(int x, int sink, int f)
{
    if(x==sink) return f;
    for(int &i=work[x]; i<v[x].size(); i++)
    {
        int y=v[x][i];
        if(level[y]>level[x] && r[x][y]>0)
        {
            int t = dfs(y,sink,min(f,r[x][y]));
            if(t>0)
            {
                r[x][y]-=t;
                r[y][x]+=t;
                return t;
            }
        }
    }
    return 0;
}

int dinic(int src, int sink)
{
    int flow=0;
    while(bfs(src,sink))
    {
        int f=0;
        memset(work,0,sizeof(work));
        while((f=dfs(src,sink,INT_MAX))>0)
            flow+=f;
    }
    return flow;
}
```

# 6 Shortest Path

## 6.1 Dijkstra

$\mathcal{O}(E \log V)$ Single-Start-Shortest-Path.
Not working for graph with minus weight.

```
const int INF = 987654321;
const int MX = V+something;
struct Edgeout
{
    int dest, w;
    bool operator<(const Edgeout &p) const
    {
        return w > p.w;
    }
};

vector <Edgeout> edgelist[MX];
int V, E, start;
int dist[MX];

bool relax(Edgeout edge, int u)
{
    bool flag = 0;
    int v = edge.dest, w = edge.w;
    if (dist[v] > dist[u] + w && (dist[u]!=INF))
    {
        flag = true;
        dist[v] = dist[u]+w;
    }
    return flag;
}

int dijkstra()
{
    fill(dist,dist+MX,INF);
    dist[start] = 0;
    priority_queue<Edgeout> pq;
    pq.push({start,0});
    while(!pq.empty())
    {
        Edgeout x = pq.top();
        int v = x.dest, w = x.w;
        pq.pop();
        if (w>dist[v])
            continue;
        for (auto ed : edgelist[v])
            if (relax(ed,v))
                pq.push({ed.dest,dist[ed.dest]});
    }
}
```

## 6.2 Bellman Ford

$\mathcal{O}(EV)$ Single-Start-Shortest-Path.
Not working for graph with minus cycle $\rightarrow$ must detect.

```
struct Edge
{
    int u, v, w;
};

vector <Edge> edgelist;
int V, E;
int dist[V+1];

bool relax_all_edge()
{
    bool flag = false;
    for (auto it:edgelist)
    {
        int u = it.u, v = it.v, w = it.w;
        if (dist[v] > dist[u] + w && (dist[u]!=INF))
        {
            flag = true;
            dist[v] = dist[u]+w;
        }
    }
    return flag;
}

int bellman_ford()
{
    fill(dist,dist+V+2,INF);
    dist[1] = 0;
    for (int i = 0; i<V-1; i++)
    {
        relax_all_edge();
    }
    if (relax_all_edge())
        return -1;
    else
        return 0;
}
```

### 6.3 SPFA Algorithm

Average $\mathcal{O}(E)$, worst $\mathcal{O}(VE)$ time. Average-case improvement of Bellman Ford by using an additional queue. → 데이터를 누가 짰을지를 생각해 보면 그냥 이런거 집어치우는게 맞을듯. ICPC Preliminary에서나 쓰자.

### 6.4 Floyd-Warshall

Works on adjacency matrix, in $\mathcal{O}(V^3)$.

```
int d[120][120];
int n;
void Floyd_Warshall()
{
    for (int i = 1; i<=n; i++)
        for (int j = 1; j<=n; j++)
            for (int k = 1; k<=n; k++)
                d[j][k] = MIN(d[j][k],d[j][i]+d[i][k]);
}
```

# 7 Dynamic

### 7.1 Longest Increasing Subsequence

Find LIS in $\mathcal{O}(n \log n)$ time.

```
vector <int> sequence;
vector <int> L;
int lis_len;
int position[BIG];
int lis[BIG];
int lis_pushed[BIG];
int n;
void FindLIS(vector <int> &seq)
{
    L.push_back(seq[0]);
    position[0] = 0;
    for (int i = 1; i<n; i++)
    {
        int u = L.size();
        if (seq[i] > L[u-1])
        {
            position[i] = u;
            L.push_back(seq[i]);
        }
        else
        {
            int pos = lower_bound(L.begin(),L.end(),seq[i])-L.begin();
            L[pos] = seq[i];
            position[i] = pos;
        }
    }
    lis_len=L.size();
    int lookingfor = lis_len-1;
    for (int i = n-1; i>=0; i--)
    {
        if (lis_pushed[position[i]]==0 && lookingfor == position[i])
        {
            lis[position[i]] = seq[i];
            lis_pushed[position[i]]=1;
            lookingfor--;
        }
    }
}
```

## 7.2 Largest Sum Subarray

Computes sum of largest sum subarray in $\mathcal{O}(N)$

```
void consecsum(int n)
{
    dp[0] = number[0];
    for (int i = 1; i<n; i++)
        dp[i] = MAX(dp[i-1]+number[i],number[i]);
}

int maxsum(int n)
{
    consecsum(n);
    int max_sum=-INF;
    for (int i = 0; i<n; i++)
        dp[i] > max_sum ? max_sum = dp[i] : 0;
    return max_sum;
}
```

## 7.3 0-1 Knapsack

```
int dp[N][W];
int weight[N];
int value[N];
void knapsack()
{
    for (int i = 1; i<=n; i++)
    {
        for (int j = 0; j<=W; j++)
            dp[i][j] = dp[i-1][j];
        for (int j = weight[i]; j<=W; j++)
            dp[i][j] = max(dp[i][j], dp[i-1][j-weight[i]]+value[i]);
    }
}
```

## 7.4 Longest Common Subsequence

```
//input : two const char*
//output : their LCS, in c++ std::string type
string lcsf(const char *X,const char *Y)
{
    int m = (int)strlen(X);
    int n = (int)strlen(Y);
    int L[m+1][n+1];
    for (int i=0; i<=m; i++)
    {
        for (int 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]);
        }
    }
    int index = L[m][n];

    char lcsstring[index+1];
    lcsstring[index] = 0;

    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (X[i-1] == Y[j-1])
        {
            lcsstring[index-1] = X[i-1];
            i--; j--; index--;
        }
        else if (L[i-1][j] > L[i][j-1])
            i--;
        else
            j--;
    }
    string lcsstr = lcsstring;
    return lcsstr;
}
```

# 8 String

## 8.1 KMP Algorithm

```
// Original Author : bowbowbow (bowbowbow.tistory.com)
vector<int> getPi(string p)
{
    int m = (int)p.size(), j=0;
    vector<int> pi(m, 0);
    for(int i = 1; i< m ; i++)
    {
        while(j > 0 && p[i] !=  p[j])
            j = pi[j-1];
        if(p[i] == p[j])
            pi[i] = ++j;
    }
    return pi;
}
vector<int> kmp(string s, string p)
{
    vector<int> ans;
    auto pi = getPi(p);
    int n = (int)s.size(), m = (int)p.size(), j =0;
    for(int i = 0 ; i < n ; i++)
    {
        while(j>0 && s[i] != p[j])
            j = pi[j-1];
        if(s[i] == p[j])
        {
            if(j==m-1)
            {
                ans.push_back(i-m+1);
                j = pi[j];
            }
            else
                j++;
        }
    }
    return ans;
}
```

## 8.2 Manacher's Algorithm

```
//original Author : Myungwoo (blog.myungwoo.kr)
int N,A[MAXN];
char S[MAXN];

void Manachers()
{
    int r = 0, p = 0;
    for (int i=1;i<=N;i++)
    {
        if (i <= r)
            A[i] = min(A[2*p-i],r-i);
        else
            A[i] = 0;
        while (i-A[i]-1 > 0 && i+A[i]+1 <= N
        && S[i-A[i]-1] == S[i+A[i]+1])
            A[i]++;
        if (r < i+A[i])
            r = i+A[i], p = i;
    }
}
```

## 8.3 Trie

```
struct Trie_Node
{
    Trie_Node * child[26];
    Trie_Node()
    {
        fill(child,child+26,nullptr);
        this->end = 0;
    }
    bool end;
    void insert(const char* key)
    {
        if (*key == 0)
            end = true;
        else
        {
            int cur = *key-'a';
            if (child[cur]==NULL)
                child[cur] = new Trie_Node();
            child[cur]->insert(key+1);
        }
    }
    bool find(char *key)
    {
        if (*key == 0)
            return false;
        if (end)
            return true;
        int cur = *key - 'a';
        return child[cur]->find(key+1);
    }
};
```

# 9 Miscellaneous

## 9.1 Useful Bitwise Functions in C++

```
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_clzll(ll x);// number of leading zero
int __builtin_ctzll(ll x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
int __builtin_popcountll(ll x);// number of 1-bits in x

lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);

// compute next perm. ex) 00111, 01011, 01101, 01110, 10011, 10101..
ll next_perm(ll v)
{
  ll t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
```

## 9.2 List of Useful Numbers

```
< 10^k    prime   # of prime         < 10^k             prime
--------------------------------------------------------------
1             7            4         10            9999999967
2            97           25         11           99999999977
3           997          168         12          999999999989
4          9973         1229         13         9999999999971
5         99991         9592         14        99999999999973
6        999983        78498         15       999999999999989
7       9999991       664579         16      9999999999999937
8      99999989      5761455         17     99999999999999997
9     999999937     50847534         18    999999999999999989
```

# 10 Checkpoints

## 10.1 Debugging

- $10^5 * 10^5 \Rightarrow$ INTEGER OVERFLOW.
- If unsure with overflow, use
  `#define int long long` and stop caring.
- 행렬과 기하의 $i, j$ 인덱스 조심. 헷갈리면 쓰면서 가기.
- output이 특정 수열/OX 형태 : 작은 예제를 Exhasutive Search. 모르는 무언가를 알기 위해서는 데이터가 필요하다.

## 10.2 Thinking

- 모든 경우를 다 할 수 없나? 왜 안 되지? 시간 복잡도 잘 생각해 보기. 정해의 Target Complexity를 먼저 생각하고 주요 알고리즘들의 Complexity로 짜맞추기.
  예를들어, 쿼리가 30만개 들어온다면 한 쿼리를 적어도 $\log n$ 에 처리할 방법이 아무튼 있다는 뜻.
- 단조함수이며, 충분히 빠르게 검증가능한가 : Binary Search.
- 차원이 높은 문제 : 차원 내려서 생각하기. $3 \rightarrow 2$.
- 이 문제가 사실 그래프 관련 문제는 아닐까?
  - 만약 그렇다면, '간선' 과 '정점' 은 각각..?
- 이 문제에 Overlapping Subproblem이 보이나? $\rightarrow$ Dynamic Programming 을 적용.
- 답의 상한이 Reasonable 하게 작은가?
- 답이 단순한 수열이면, Berlekamp-Massey (본선에서) or OEIS (예선)
- 마지막 생각 : 조금 추하지만 해싱이나 Random `bitset` 을 이용한 $n^2/64$ 같은걸로 뚫을 수 있나?