hash

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
ull seed = 1e9 + 7;
const int N = 1e6 + 7;
ull fac[N], ha[N];
ull getSTR(int l, int r){
    return ha[r] - ha[l - 1] * fac[(r - l + 1)];
//  if(l == 0)  return ha[r];
//  else return
}
int n;
bool jud(int i){
    int flag = 1;
    ull jud = getSTR(1, i);
    for(int j = i + 1; j <= n; j+= i){
        if(getSTR(j, j + i - 1) != jud){
            flag = 0;
            break;
        }
    }
    if(flag)    return true;
    return false;
}
signed main(){
    int mx = 0;
    fac[0] = 1;
    for(int i = 1; i < N; i++)  fac[i] = fac[i - 1] * seed;
    int m;  cin >> m;
    set<ull>st;
    for(int i = 1; i <= m; i++){
        string s;   cin >> s;
        int n = s.size();
        ull now = 0;
        for(int i = 0; i < n; i++){
            now = now * seed + s[i];
        }
        st.insert(now);
    }
    int ans = st.size();
    cout << ans <<"\n";
}
```

dijk

```
struct edge {
  int v, w;
};

struct node {
  int dis, u;

  bool operator>(const node& a) const { return dis > a.dis; }
};

vector<edge> e[maxn];
int dis[maxn], vis[maxn];
priority_queue<node, vector<node>, greater<node> > q;

void dijkstra(int n, int s) {
  memset(dis, 0x3f, (n + 1) * sizeof(int));
  dis[s] = 0;
  q.push({0, s});
  while (!q.empty()) {
    int u = q.top().u;
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto ed : e[u]) {
      int v = ed.v, w = ed.w;
      if (dis[v] > dis[u] + w) {
        dis[v] = dis[u] + w;
        q.push({dis[v], v});
      }
    }
  }
}
```

树状数组

```
const int N = 1e5 + 7;
ll Btree[N];
int lowbit(int x){
    return x & -x;
}
ll getsum(int x){
    int ans = 0;
    while(x > 0){
        ans += Btree[x];
        x -= lowbit(x);
    }
    return ans;
}
void add(int x, ll k){
    while(x <= n){
        Btree[x] += k;
        x += lowbit(x);
    }
}
```

并查集

```
#include<bits/stdc++.h>
using namespace std;
#define pb emplace_back
const int mod = 1e9 + 7;
const int N = 1e5 + 7;
int fa[N];
int find(int x){
    if(x == fa[x])  return x;
    return fa[x] = find(fa[x]);
}
void unite(int x, int y){
    fa[find(x)] = find(y);
}
signed main(){
    int n, m;   cin >> n >> m;
    for(int i = 1; i <= n; i++){
        fa[i] = i;
    }
    for(int i = 1; i <= m; i++){
        int j, x, y;    cin >> j >> x >> y;
        if(j == 1)  unite(x, y);
        else{
            if(find(x) == find(y)){
                cout <<"Y\n";
            }else{
                cout <<"N\n";
            }
        }
    }
}
```

二叉树

```
//创建二叉树和三种遍历,输入序列如 1 5 8 0 0 0 6 0 0
#include <stdio.h>

struct Node{
    int data;
    Node* left;
    Node* right;
};

void create(Node* &T){
    int x;
    scanf("%d",&x);
    if(x==0){
        T = NULL;
        return;
    }
    T = new Node;
    T->data = x;
    T->left = T->right = NULL;
    create(T->left);
    create(T->right);
}

void preOrder(Node * T){
    if(T==NULL)
        return;
    printf("%d ",T->data);
    preOrder(T->left);
    preOrder(T->right);
}

void midOrder(Node * T){
    if(T==NULL)
        return;
    midOrder(T->left);
    printf("%d ",T->data);
    midOrder(T->right);
}

void postOrder(Node * T){
    if(T==NULL)
        return;

    postOrder(T->left);
    postOrder(T->right);
    printf("%d ",T->data);
}


int main(){
    Node* root = new Node;
    create(root);
    preOrder(root);
    printf("\n");
    midOrder(root);
    printf("\n");
    postOrder(root);

    return 0;
}
```

堆

```
void swap(int &x,int &y){int t=x;x=y;y=t;}//交换函数
int heap[N];//定义一个数组来存堆
int siz;//堆的大小
void push(int x){//要插入的数
    heap[++siz]=x;
    now=siz;
    //插入到堆底
    while(now){//还没到根节点，还能交换
        ll nxt=now>>1;//找到它的父亲
        if(heap[nxt]>heap[now])swap(heap[nxt],heap[now]);//父亲比它大，那就交换
        else break;//如果比它父亲小，那就代表着插入完成了
        now=nxt;//交换
    }
    return;
}
```

```
void pop(){
    swap(heap[siz],heap[1]);siz--;//交换堆顶和堆底，然后直接弹掉堆底
    int now=1;
    while((now<<1)<=siz){//对该节点进行向下交换的操作
        int nxt=now<<1;//找出当前节点的左儿子
        if(nxt+1<=siz&&heap[nxt+1]<heap[nxt])nxt++;//看看是要左儿子还是右儿子跟它换
        if(heap[nxt]<heap[now])swap(heap[now],heap[nxt]);//如果不符合堆性质就换
        else break;//否则就完成了
        now=nxt;//往下一层继续向下交换
    }
}
```