1 Brute Forcee

```
/*Brute Force*/
#define MAXN 1<<18+5 //雙倍空間
/*折半枚舉 與 二進制枚舉*/
int main() {
    int n, m, i, temp;
    ll mod, mod_max = 0;
    vector<ll> arr, ans(MAXN,0), ans2(MAXN,0);
    cin >> n >> m;
    for(i=0;i<n;i++){
        cin >> temp;
        arr.push_back(temp%m);
    }

    //折半枚舉
    for(int i=0;i<(1<<(n/2));i++){ //2^(n/2)
        for(int j=0;j<n/2;j++){
            if(i>>j&1) //二進制枚舉(選或不選)
                ans[i] = (ans[i] + arr[j]) % m; //前半
                    枚舉
        }
    }
    for(int i=0;i<(1<<(n-n/2));i++){ //2^(n-n/2)
        for(int j=0;j<(n-n/2);j++){
            if(i>>j&1) ans2[i] = (ans2[i] + arr[n/2+j])
                % m; //後半枚舉
        }
    }

    //二分維護
    temp = 1<<(n-n/2);
    sort(ans2.begin(), ans2.begin() + temp);
    for(auto i:ans){
        mod_max = max(mod_max, i + *(upper_bound(ans2.
            begin(), ans2.begin() + temp, m-1-i)-1));
        //mod最大為m-1，配對另一半最優解
    }
    cout << mod_max <<"\n";

    return 0;
}
```

2 Data Structure

2.1 SQRT

```
// build  O(n)
// update O(√n)
// query  O(√n)
//分塊結構
//假設要求區間總和
struct blk{
    vector<int> local;    //每塊的全部元素
    int global;           //儲存每塊的總和
    int tag;              //儲存整塊一起更新的值
    blk(){                //初始化
        local.clear();    //清空區間元素
        tag = global = 0; //將區間總和先設為0
    }
};
vector<blk> b;
void build(){
    int len=sqrt(n),num=(n+len-1)/len;
    for(int i=0;i<n;i++){    //第i個元素分在第 i/len 塊
        cin>>x;
        //存入區間中
        b[i/len].local.push_back(x);
        //更新區間總和
        b[i/len].global += x;
    }
}
void update(int ql,int qr,int v){
    int blk_l=ql/len,blk_r=qr/len,ret=0;
    if(blk_l == blk_r){
        //如果都在同一塊直接一個一個跑過去就好
        for(int i=ql;i<=qr;i++)
            b[blk_l].local[i%len]+=v;
        b[blk_l].global+=(qr-ql+1)*v;
        return;
    }
    for(int i=ql;i<(blk_l+1)*len;i++){  //最左的那一塊
        b[blk_l].local[i%len]+=v;
        b[blk_l].global+=v;
    }
    for(int i=blk_l+1;i<blk_r;i++){  //中間每塊
        b[i].tag+=v;
        b[i].global+=v*len;
    }
    for(int i=blk_r*len;i<=qr;i++){  //最右的那一塊
        b[blk_r].local[i%len]+=v;
        b[blk_r].global+=v;
    }
}
int query(int ql,int qr){
    int blk_l=ql/len,blk_r=qr/len,ret=0;
    if(blk_l == blk_r){
        //如果都在同一塊直接一個一個跑過去就好
        for(int i=ql;i<=qr;i++)
            ret+=b[blk_l].local[i%len]+b[blk_l].tag;
        return ret;
    }
    for(int i=ql;i<(blk_l+1)*len;i++)    //最左的那一塊
        ret+=b[blk_l].local[i%len]+b[blk_l].tag;
    for(int i=blk_l+1;i<blk_r;i++)    //中間每塊的總和
        ret+=b[i].global;
    for(int i=blk_r*len;i<=qr;i++)    //最右的那一塊
        ret+=b[blk_r].local[i%len]+b[blk_r].tag;
    return ret;
}
```

2.2 Mo's Algorithm

```
// n為序列總長度，q為詢問比數，p為移動一格的複雜度
// O(p(q+n)√n)
int n,k = sqrt(n);    //每塊大小為k
struct query{
    int l,r,id;       //詢問的左界右界 以及 第幾筆詢問
    friend bool operator<(const query& lhs,const query&
        rhs){
        return lhs.l/k==rhs.l/k ? lhs.r<rhs.r : lhs.l<
            rhs.l;
    }    //先判斷是不是在同一塊
         //不同塊的話就比較塊的順序，否則比較右界r
};
int num = 0;
int cnt[1000005], ans[30005];
vector<query> q;
void add(int x){    //新增元素到區間內
    ++cnt[x];
    if(cnt[x] == 1)    ++num;
}
void sub(int x){    //從區間內移除元素
    --cnt[x];
    if(cnt[x] == 0)    --num;
}
void solve(){
    sort(q.begin(),q.end());
    for(int i=0,l=-1,r=0;i<n;i++){
        while(l>q[i].l) add(--l);
        while(r<q[i].r) add(++r);//記得要先做新增元素的
        while(l<q[i].l) sub(l++);//再做移除元素的
        while(r>q[i].r) sub(r--);
        ans[q[i].id] = num;      //移到區間後儲存答案
}    }
```

2.3 BIT

```
/*BIT 樹狀數組(動態前綴和)*/
//BIT and Array start at 1
#define MAXN 100005 //最大區間<MAXN
vector <int> arr(MAXN); //原始陣列
vector <int> bit(MAXN); //BIT數組

//前綴和查詢
ll query(int i) { //index
    ll ret = 0;
    while(i > 0) ret += bit[i], i -= i & -i; // 1-base i-lowbit(i)
    return ret;
}

//單點增值
void modify(int i, int val) { //index,value
    while(i <= MAXN) bit[i] += val, i += i & -i; // i+lowbit(i)
}
```

2.4 Segment tree

```
/*Segment tree 線段樹(區間問題)*/
//segment tree and Array start at 1
// [l,r] 最大區間設為[1,n]
// [ql,qr] 目標區間
// pos,val 修改位置,修改值
#define MAXN 100005*4  //tree大小為4n
#define cl(x) (x*2)    //左子節點index
#define cr(x) (x*2+1)  //右子節點index
#define NO_TAG 0 //懶惰記號
vector <int> tag(MAXN);
vector <int> arr(MAXN);
vector <int> tree(MAXN);

void build(int i,int l,int r){ //i為當前節點index，l,r為當前遞迴區間
    if(l == r){ // 遞迴到區間大小為1
        tree[i] = arr[l];
        return;
    }
    int mid=(l+r)/2; //往兩邊遞迴
    build(cl(i),l,mid);
    build(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]); //<-可修改條件
    //將節點的值設成左右子節點的最大值
}

// i 為當前節點index, l, r當前區間左右界，ql, qr詢問左右界
int query(int i,int l,int r,int ql,int qr){
    if(ql <= l && r <= qr){ //若當前區間在詢問區間內，直接回傳區間最大值
        return tree[i];
    }
    int mid=(l+r)/2, ret=0; //<-可修改條件
    if(ql<=mid) // 如果左子區間在詢問區間內
        ret = max(ret, query(cl(i),l,mid,ql,qr));   // <-可修改條件
    if(qr> mid) // 如果右子區間在詢問區間內
        ret = max(ret, query(cr(i),mid+1,r,ql,qr));  // <-可修改條件
    return ret;
}

/*單點修改*/
void update(int i,int l,int r,int pos,int val){
    if(l == r){ // 修改 a[pos] 的值為 val
        tree[i] = val;
        return;
    }
    int mid=(l+r)/2;
    if(pos <= mid) // 如果修改位置在左子節點，往左遞迴
        update(cl(i),l,mid,pos,val);
    else // 否則往右遞迴
        update(cr(i),mid+1,r,pos,val);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);    //<-可修改條件
}

/*區間修改*/
//將區間 [l, r] 的值都加 v
void push(int i,int l,int r){
    if(tag[i] != NO_TAG){ // 判斷是否有打標記,NO_TAG=0
        tree[i] += tag[i];  // 有的話就更新當前節點的值
        if(l != r){  // 如果有左右子節點把標記往下打
            tag[cl(i)] += tag[i];
            tag[cr(i)] += tag[i];
        }
        tag[i] = NO_TAG; // 更新後把標記消掉
    }
}
void pull(int i,int l,int r){
    int mid = (l+r)/2;
    push(cl(i),l,mid); push(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);
}
void update(int i,int l,int r,int ql,int qr,int v){
    push(i,l,r);
    if(ql<=l && r<=qr){
        tag[i] += v; //將區間 [l, r] 的值都加 v
        return;
    }
    int mid=(l+r)/2;
    if(ql<=mid)  update(cl(i),l,mid,ql,qr,v);
    if(qr> mid)  update(cr(i),mid+1,r,ql,qr,v);
    pull(i,l,r);
}
```

2.5 Treap

```
struct Treap{
    int key,pri,sz;     //key,priority,size
    Treap *l, *r;       //左右子樹
    Treap(){}
    Treap(int _key){
        key = _key;
        pri = rand();   //隨機的數維持樹的平衡
        sz  = 1;
        l = r = nullptr;
    }
};
Treap *root;
int Size(Treap* x){ return x ? x->sz : 0 ; }
void pull(Treap *x){ x->sz = Size(x->l) + Size(x->r) + 1;}
Treap* merge(Treap *a,Treap *b){
    //其中一個子樹為空則回傳另一個
    if(!a || !b)    return a ? a : b;
    if(a->pri > b->pri){//如果a的pri比較大則a比較上面
        a->r = merge(a->r,b);//將a的右子樹跟b合併
        pull(a);
        return a;
    }
    else{  //如果b的pri比較大則b比較上面
        b->l = merge(a,b->l);//將b的左子樹根a合併
        pull(b);
        return b;
    }
}
void splitByKth(Treap *x,int k,Treap*& a,Treap*& b){
    if(!x){  a = b = nullptr;  }
```

```
    else if(Size(x->l) + 1 <= k){
        a = x;
        splitByKth(x->r, k - Size(x->l) - 1, a->r, b);
        pull(a);
    }
    else{
        b = x;
        splitByKth(x->l, k, a, b->l);
        pull(b);
    }
}
void splitByKey(Treap *x,int k,Treap*& a,Treap*& b){
    if(!x){  a = b = nullptr;  }
    else if(x->key<=k){
        a = x;
        splitByKey(x->r, k, a->r, b);
        pull(a);
    }
    else{
        b = x;
        splitByKey(x->l, k, a, b->l);
        pull(b);
    }
}
void insert(int val){            //新增一個值為val的元素
    Treap *x = new Treap(val);  //設一個treap節點
    Treap *l,*r;
    splitByKey(root, val, l, r);//找到新節點要放的位置
    root = merge(merge(l,x),r); //合併到原本的treap裡
}
void erase(int val){             //移除所有值為val的元素
    Treap *l,*mid,*r;
    splitByKey(root, val, l, r);//把小於等於val的丟到l
    splitByKey(l, val-1, l, mid);
    //小於val的丟到l,等於val的就會在mid裡
    root = merge(l,r);           //將除了val以外的值合併
}
int findVal(int val){ //小於等於val的size
    int size = -1;
    Treap *l, *r;
    splitByKey(root, val, l, r); //把小於等於val的丟到l
    size = Size(l);
    root = merge(l,r);
    return size;
}
void interval(Treap *&o, int l, int r) {// [l,r]區間
    Treap *a, *b, *c;
    splitByKey(o, l - 1, a, b), splitByKey(b, r, b, c);
    // operate
    o = merge(a, merge(b, c));
}
void inOrderTraverse(Treap* o, int print) {// 中序
    if (o != NULL){
        push(o);
        inOrderTraverse(o->l, print);
        // print
        if(print) cout << o->val <<"  ";
        inOrderTraverse(o->r, print);
    }
}
// Rank Tree
// Kth(k)：查找第k小的元素
// Rank(x)：x的名次，即x是第幾小的元素
int kth(Treap* o, int k){
    if(o == NULL || k > o -> sz || k <= 0)   return 0;
    int s = (o -> l == NULL ? 0 : o -> l -> sz);
    if(k == s + 1)  return o -> key;
    else if(k <= s) return kth(o -> l, k);
    else            return kth(o -> r, k - s - 1);
}
int rank(Node* o, int x){
    if(o == NULL) return 0;
    int res = 0;
    int s = (o -> l == NULL ? 0 : o -> l -> sz);
    if(x <= o -> key){
        res += rank(o -> l, x);
        res += x == o -> key;
    }
    else{
        res += s + 1;
        res += rank(o -> r, x);
    }
    return res;
}
```

3 Graph

3.1 DFS and BFS

```
//DFS
void dfs(int x){
    vis[x]=1;
    for(int i:adj[x]){
        if(!vis[i])
            dfs(i);
    }
}
//BFS
void bfs(int s){
    queue<int> q;
    q.push(s);
    vis[s]=1;
    while(!q.empty()){
        int x=q.front();q.pop();
        for(int i:ADJ[x]){
            if(!vis[i])
                q.push(i),vis[i]=1;
        }
    }
}
void init(int N){
    for(int i=0;i<N;i++){
        if(!adj[i].empty()) adj[i].clear();
    }
}
```

3.2 Disjoint Set(Union-Find)

```
/*Disjoint Set(Union-Find) 並查集*/
int f[N];    // 宣告父節點陣列 f
void init(int n) {
    for (int i = 0; i < n; i++)
        f[i] = i;
}
int find(int x) {
    return f[x] == x ? x : f[x] = find(f[x]);
}
void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x != y)    f[y] = x;
}
```

3.3 Kruskal’s algorithm 最小生成樹

```
/*Kruskal’s algorithm 最小生成樹*/
//搭配 Disjoint Set(Union-Find)
struct Edge {
    int u, v, w;  // 點 u 連到點 v 並且邊權為 w
    friend bool operator<(const Edge& lhs, const Edge&
        rhs) {
        return lhs.w > rhs.w;//兩條邊比較大小用邊權比較
    }
};
priority_queue<Edge> graph();// 宣告邊型態的陣列 graph
int kruskal(int m){
```

```cpp
    int tot = 0;
    for (int i = 0; i < m ; i++) {
        if (find(graph.top().u) != find(graph.top().v))
             { // 如果兩點未聯通
            merge(graph.top().u, graph.top().v);
               // 將兩點設成同一個集合
            tot += graph.top().w;  // 權重加進答案
        }
        graph.pop();
    }
    return tot;
}
int main() {
    int u, v, w, n, m,;
    cin >> n >> m; //node,edge
    init(n);
    for (int i = 0; i < m; i++) {
        cin >> u >> v >> w;
        graph.push(Edge{u,v,w});
    }
    cout << kruskal(m) << "\n";
    return 0;
}
```

3.4 Dijkstra’s algorithm

```cpp
/*Dijkstra's algorithm 單源最短路徑*/
#define MAX_V 100
#define INF 10000
struct Edge {
  int idx,w;
};
bool operator>(const Edge& a, const Edge& b) {
  return a.w > b.w;
}
int dist[MAX_V];
vector<vector<Edge> > adj(MAX_V);
void dijkstra(int vn, int s) {
  vector <bool> vis(vn, false);
  fill(dist, dist + vn, INF); dist[s] = 0;

  priority_queue <Edge, vector<Edge>, greater<Edge> >
      pq;
  Edge node;
  node.idx = s; node.w = 0;
  pq.emplace(node);
  while (!pq.empty()) {
    int u = pq.top().idx; pq.pop();
    if (vis[u])continue;
    vis[u] = true;
    for (auto v : adj[u]) {
      if (dist[v.idx] > dist[u] + v.w) {
        dist[v.idx] = dist[u] + v.w;
        node.w = dist[v.idx];
        node.idx = v.idx;
        pq.emplace(node);
      }

    }
  }
}
int main() {
    int start, end, u, v, w, i, n, m;
    cin >> n >> m; //node,edge
    for(i=0;i<m;i++){
        cin >> u >> v >> w;
        Edge node;
        node.idx = v; node.w = w;
        adj[u].push_back(node);
    }
    //從start連接到end的最短路徑
    cin >> start >> end;
    dijkstra(n, start);
    if(dist[end]==INF) cout << "NO\n";
    else cout << dist[end] <<"\n";
  return 0;
}
```

3.5 Floyd-Warshall

```cpp
/*Floyd-Warshall 全點對最短路徑*/
//建立dp表，查詢任一點對最短路徑。
void floyd(){
    //將每個點對距離設為INF
    memset(dist,0x3f3f3f3f,sizeof(dist));
    //dist[u][v]為點u到點v的最短路徑
    //自己到自己的距離設為0
    for(int i=0;i<n;i++)  dist[i][i]=0;
    //輸入圖
    for(int i=0;i<m;i++)  cin>>u>>v>>w,dist[u][v]=w;
    for(int i=0;i<n;i++)     //窮舉中繼點
        for(int j=0;j<n;j++) //j,k窮舉點對
            for(int k=0;k<n;k++)
                dist[j][k]=min(dist[j][k],dist[j][i]+
                    dist[i][k]);
}
```

3.6 BellmanFord algorithml

```cpp
#define N 100
#define INF 1000
int dist[N][N];
vector<vector<int> > length(N,vector<int>(N));
void BellmanFord(int n, int v)
{ /* n為節點總數，計算單一起點v/所有終點的最短路徑，其
    中邊長允許是負值，length為adjacency matrix */
    for (int k = 0; k < n; k++)for (int i = 0; i < n; i
        ++)i == 0 ? dist[k][i] = 0 : dist[k][i] = INF;
        /* 對dist做初始化 */
    for (int i = 0; i < n; i++)if(length[v][i])dist[1][
        i] = length[v][i]; /* 對dist[1]做初始化 */
    for (int i = 0; i < n; i++) {
        dist[1][i] == INF ? cout << "i" : cout << dist
            [1][i];
        if (i != n - 1)cout << " ";
    }cout << "\n";
    for (int k = 2; k <= n - 1; k++) {
        for (int u = 0; u < n; u++) {
            for (int i = 0; i < length[u].size(); i++)
                {
                if (!length[u][i])continue;
                if (length[u][i] == INF)continue;
                if (dist[k][i] > dist[k-1][u] + length[
                    u][i])
                    dist[k][i] = dist[k-1][u] + length[
                        u][i];
            }
        }
        for (int i = 0; i < n; i++) {
            dist[k][i] == INF ? cout << "i" : cout <<
                dist[k][i];
            if (i != n - 1)cout << " ";
        }if (k != n - 1)cout << "\n";
    }
}
int main() {
    int i, u, v, w,s,vn;
    set<int> _set;

    while (cin >> u >> v >> w) {
        length[u][v] = w;
        _set.insert(u);
        _set.insert(v);
    }

    s = 0;
    vn = _set.size();
```

```
    BellmanFord(vn,s);

    return 0;
}
```

3.7 SPFA

```
/*SPFA 單源最短路徑(negative cycle)*/
struct Edge {
    int idx, w;
};
vector<Edge> adj[MAX_V]; //adjacency list
vector<bool> inp(MAX_V);
int dist[MAX_V];
//return true if negative cycle exists
bool spfa(int vn, int s) {
    fill(dist, dist + vn, INF); dist[s] = 0;
    vector<int> cnt(vn, 0);
    vector<bool> inq(vn, 0);
    queue<int> q; q.push(s); inq[s] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        inq[u] = false;
        for (auto v : adj[u]) {
            if (dist[v.idx] > dist[u] + v.w) {
                if (++cnt[v.idx] >= vn)return true;
                dist[v.idx] = dist[u] + v.w;
                if (!inq[v.idx]) inq[v.idx] = true, q.
                    push(v.idx);
            }
        }
    }
    return false;
}
```

4 Graph(Tree)

4.1 Eulerian Path and Circuit

```
// O(M)
//              歐拉迴路              歐拉路徑
// 無向圖/所有點的度數為偶數/度數為奇數的點數量不超過2
// 有向圖/所有點入度等於出度/全部點的入度出度一樣
//或剛好一個點出度-1=入度 另一點入度-1=出度，其他點入度
    等於出度
vector<int> path;
void dfs(int x){
    while(!edge[x].empty()){
        int u = edge[x].back();
        edge[x].pop_back();
        dfs(u);
    }
    path.push_back(x);
}
int main(){
    build_graph();
    dfs(st); // 如果剛好一個點出度-1=入度 則為起點
    reverse(path.begin(),path.end());
}
```

4.2 Topological Sort

```
// O(N+M)
for(int i=0;i<m;i++){
    cin >> u >> v;    //點 u 連到點 v
    edge[u].push_back(v);
    ++deg[v];
}
for(int i=0;i<n;i++)
    if(!deg[i]) que.push(i); //入度0先出
for(int i:edge[u]){
    --deg[i];
    if(deg[i] == 0) que.push(i);
}
```

4.3 LCA

```
// pre   O(NlgN)
// query O(lgN)
// 最近共同祖先
// 兩點間距離 / 兩點間最大邊 / 兩點間重合長度
// 時間戳記，判斷祖先關係
int ti = 0; // 當前時間
int tin[MAXN+5],tout[MAXN+5];
int dis[MAXN+5]; // 計算距離深度
int query[MAXN+5][lgN+5]; //點N的2^lgN祖先的最大邊
void dfs(int x,int f,int deep){
    fa[x] = f;
    tin[x] = ti++;
    dis[x] = deep;
    for(auto i:edge[x]){
        if(i.v == f){
            //query[x][0] = i.w;
            continue; //如果是父節點，已經走到底
        }
        dfs(i.v, x, deep+i.w);
    }
    tout[x] = ti++;
}
bool isAncestor(int u, int v){
    return tin[u]<=tin[v] && tout[u] >= tout[v];
}
// LCA
int n,lgn;
int anc[MAXN+5][lgN+5]; //點N的2^lgN祖先
int getLca(int u, int v){
    if(isAncestor(u, v))    return u;
    // 如果 u 為 v 的祖先則 lca 為 u
    if(isAncestor(v, u))    return v;
    // 如果 v 為 u 的祖先則 lca 為 u
    for(int i=lgn;i>=0;i--){
        // 判斷 2^lgN, 2^(lgN-1),...2^1, 2^0 倍祖先
        if(!isAncestor(anc[u][i], v))
        // 如果 2^i 倍祖先不是 v 的祖先
            u = anc[u][i];   // 則往上移動
    }
    return anc[u][0]; // 回傳此點的父節點即為答案
}
// 找出路徑最大邊
int max_cost(int u, int v){
    int max_cost = 0;
    if(u == v) return max_cost;
    for(int i=lgn;i>=0;i--){
        // 判斷 2^lgN, 2^(lgN-1),...2^1, 2^0 倍祖先
        if(!isAncestor(anc[u][i], v)){
            // 如果 2^i 倍祖先不是 v 的祖先
            max_cost = max(max_cost,query[u][i]);
            u = anc[u][i];   // 則往上移動
        }
    }
    return max(max_cost,query[u][0]);
} // max(max_cost(u,nodeLca), max_cost(v,nodeLca))
// 兩點距離
int dist(int u, int v){
    //depth[X] + depth[Y] - 2 * depth[ancestor]
    return dis[u] + dis[v]-2*dis[find(v)];
}
// init 建表
for(s=1;s<=n;s++) anc[s][0] = fa[s];
for(i=1;i<=lgn;i++){
    for(s=1;s<=n;s++){
        //點 s 的 2^i 倍祖先即為
```

```
        //s 的 2^(i-1) 倍祖先的 2^(i-1) 倍祖先
        anc[s][i] = anc[anc[s][i-1]][i-1];
        //建最大邊的表
        query[s][i] = max(query[s][i-1],query[anc[s][i
            -1]][i-1]);
    }
}
```

4.4 樹上差分

```
#include<bits/stdc++.h>
#define MAX 3e5+5
using namespace std;

int n;
vector<vector<int>>edge(MAX), fa(MAX, vector<int>(21,
    0));
vector<int>a(MAX), dep(MAX), cnt(MAX, 0);
void dfs(int rt,int f) {
  fa[rt][0] = f;
  dep[rt] = dep[f] + 1;
  for (int i = 1; i <= 20; i++) {
    fa[rt][i] = fa[fa[rt][i - 1]][i - 1];
  }
  for (auto i : edge[rt]) {
    if (i == f)continue;
    dfs(i, rt);
  }
}
int lca(int a, int b) {
  if (dep[a] < dep[b]) {
    swap(a, b);
  }
  for (int i = 20; i >= 0; i--) {
    if (dep[fa[a][i]] >= dep[b]) {
      a = fa[a][i];  //上跳
    }
  }
  if (a == b)
    return a;
  for (int i = 20; i >= 0; i--) {
    if (fa[a][i] != fa[b][i]) {
      a = fa[a][i];
      b = fa[b][i];
    }
  }
  return fa[a][0];
}
void dfssum(int rt,int f) {
  for (auto i : edge[rt]) {
    if (i == f)continue;
    dfssum(i, rt);
    cnt[rt] += cnt[i];
  }
}
void solve() {
  int u, v, cmnlca;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  for (int i = 0; i < n - 1; i++) {
    cin >> u >> v;
    edge[u].push_back(v);
    edge[v].push_back(u);
  }
  dfs(1, 0);
  for (int i = 0; i < n - 1; i++) {
    cmnlca = lca(a[i], a[i + 1]);
    cnt[fa[cmnlca][0]]--;  //父節點 -v
    cnt[cmnlca]--;        //lca -v
    cnt[a[i]]++;        //兩端點 +v
    cnt[a[i + 1]]++;
  }
  dfssum(1, 0);
  for (int i = 1; i <= n; i++) {  //多加的減回去
    cnt[a[i]]--;
  }
  for (int i = 1; i <= n; i++) {
    cout << cnt[i] << "\n";
  }
}
int main() {
  cin >> n;
  solve();
}
```

4.5 HLD with Segment tree

```
#define MXN 10005
#define cl(x) (x<<1)
#define cr(x) (x<<1|1)
#define INF 1e9+5
int n;
int sz[MXN], fa[MXN], heavy[MXN], dep[MXN];
int root[MXN]; //鍊的根節點
int len[MXN]; //鍊長度
struct Edge {int u, v;};
struct node {int v, w;};
vector<Edge> edge;
vector<node> graph[MXN];
vector<int> tree[MXN];  // 第i個節點為根的線段樹
vector<int> val[MXN];   // 第i個節點為根的序列
// 子樹大小
void dfs_sz(int u, int f, int d){
  sz[u] = 1, fa[u] = f, dep[u] = d;
    for(auto v : graph[u]){
        if(v.v != f){
            dfs_sz(v.v, u, d+1);
            sz[u] += sz[v.v];
            if(sz[v.v] > sz[heavy[u]]) heavy[u] = v.v;
                //重兒子
        }
    }
}
// 樹鍊剖分
void dfs_hld(int u,int f){
    for(auto v : graph[u]){
        if(v.v != f){
            if(v.v == heavy[u])root[v.v] = root[u];
            //重兒子的根，重鍊的頭
            else root[v.v] = v.v; //輕兒子的根
            val[root[v.v]].push_back(v.w); //點權
            dfs_hld(v.v, u);
        }
    }
    len[root[u]]++; //鍊長度
}
// LCA
int getLca(int x,int y){
    while(root[x] != root[y]){
        if(dep[root[x]] > dep[root[y]])
            x = fa[root[x]]; //跳鍊
        else
            y = fa[root[y]];
    }
    return (dep[x] <= dep[y] ? x : y);
}
// 線段樹
void build(int ver,int i,int l,int r){
    if(l == r){
        tree[ver][i] = val[ver][l];
        return;
    }
    int mid=(l+r)>>1;
    build(ver, cl(i), l, mid);
    build(ver, cr(i), mid+1, r);
    tree[ver][i] = max(tree[ver][cl(i)], tree[ver][cr(i
        )]); //最大邊
}
```

```
void update(int ver,int i,int l,int r,int pos,int val){
    if(l == r){ // 修改 a[pos] 的值為 val
        tree[ver][i] = val; return;
    }
    int mid=(l+r)>>1;
    if(pos <= mid) update(ver,cl(i),l,mid,pos,val);
    else           update(ver,cr(i),mid+1,r,pos,val);
    tree[ver][i] = max(tree[ver][cl(i)], tree[ver][cr(i
        )]);
}
 // i 為當前節點index, l, r當前區間左右界，ql, qr詢問左
    右界
int query(int ver,int i,int l,int r,int ql,int qr){
    if(ql <= l && r <= qr){
        return tree[ver][i];
    }
    int mid=(l+r)>>1, ret=-INF;
    if(ql<=mid) ret =
        max(ret, query(ver,cl(i),l,mid,ql,qr));
    if(qr> mid) ret =
        max(ret, query(ver,cr(i),mid+1,r,ql,qr));
    return ret;
}
void init(){
    edge.clear(); edge.resize(n-1);
    for(int i=1;i<=n;i++){
        graph[i].clear();
        tree[i].clear();
        val[i].clear();
        heavy[i]=len[i]=0;
    }
}
signed main(){
    int i,t,a,b,w,ti;
    string op;
    cin >> n;
    init();
    for(i=0; i+1<n; i++){
        cin >> a >> b >> w;
        graph[a].push_back(node{b,w});
        graph[b].push_back(node{a,w});
        edge[i] = Edge{a,b};
    }
    val[1].push_back(-INF);
    root[1] = 1;
    dfs_sz(1, 1, 0);
    dfs_hld(1, 1);
    // build tree
    for(i=1;i<=n;i++){ // 第i個節點為根的線段樹
        if(root[i] == i){
            tree[i].resize(len[i]*4,0);
            build(i, 1, 0, len[i]-1);
        }
    }
    // query
    while(cin >> op){
        if(op == "DONE") break;
        else if(op == "CHANGE"){
            cin >> i >> ti; i--;
            if(dep[edge[i].u] < dep[edge[i].v])
                swap(edge[i].u,edge[i].v);
            i = edge[i].u;
            update(root[i], 1, 0, len[root[i]]-1,
            dep[i]-dep[root[i]], ti);
        }
        else if(op == "QUERY"){
            cin >> a >> b;
            int ans = -INF;
            while(root[a] != root[b]){ //不同鍊
                if(dep[root[a]] < dep[root[b]])
                    swap(a, b);
                // 深鍊的最大邊
                ans = max(ans, query(root[a], 1, 0, len
                    [root[a]]-1, 0, dep[a]-dep[root[a
                    ]]));
                a = fa[root[a]]; //跳鍊
            }
            if(a != b){ //不同節點
                int mn =
                    min(dep[a],dep[b])-dep[root[a]]+1;
                int mx =
                    max(dep[a],dep[b])-dep[root[a]];
                //所在節點區間 mn,mx
                ans = max(ans, query(root[a], 1, 0, len
                    [root[a]]-1, mn, mx));
            }
            cout << ans <<"\n";
}   }   }
```

4.6 DSU on Tree

```
void add(int v, int p, int x){
    cnt[ col[v] ] += x;
    // now you can insert test
    for(auto u: g[v])
        if(u != p && !big[u])
            add(u, v, x);
}
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
        // run a dfs on small childs and clear them
            from cnt
            dfs(u, v, 0);

    if(bigChild != -1)
    // bigChild marked as big and not cleared from cnt
        dfs(bigChild, v, 1), big[bigChild] = 1;
    add(v, p, 1);
    ans[v] = sum;
    //now cnt[c] is the number of vertices in subtree
        of vertex v that has color c. You can answer
        the queries easily.
    if(bigChild != -1)
        big[bigChild] = 0;
    if(keep == 0){
        add(v, p, -1);
        // now you can init to 0
    }
}
```

5 DP

5.1 背包問題

```
/*背包問題*/

// n：第0種到第n種物品要放進背包內。
// w：背包耐重限制。
// c(n, w)：只有第0種到第n種物品
// 耐重限制為w，此時的背包問題答案。
// weight[n]：第n種物品的重量。
// cost[n]：第n種物品的價值。
// number[n]：第n種物品的數量。

// 0/1背包滾動
// 每種物品只會放進背包零個或一個。
const int N = 500, W = 2000000; //N個物品,耐重W
int cost[N], weight[N];
int c[W + 1];
void knapsack(int n, int w)
{
  c[0] = 0;
  for (int i = 0; i < n; ++i)
```

```
    for (int j = w; j - weight[i] >= 0; --j)
      c[j] = max(c[j], c[j - weight[i]] + cost[i]);
  cout << c[w];
}
// 0/1背包可用於：
// 一個數字集合，挑幾個數字，總和恰為零（Subset Sum
    Problem）
// 一個數字集合，挑幾個數字，總和恰為整體總和的一半 (
    Partition Problem）
// N個不同重量物品，M個不同耐重箱子，用最少箱子裝所有物
    品（Bin Packing Problem）

// 無限背包
// 物品有許多種類，每一種物品都無限量供應的背包問題。
void knapsack(int n, int w)
{
    memset(c, 0, sizeof(c));
    for (int i=0; i<n; ++i)
        for (int j = weight[i]; j <= w; ++j)
            c[j] = max(c[j], c[j - weight[i]] + cost[i
                ]);

    cout << "最高的價值為" << c[w];
}

// 有限背包
// 物品有許多種類，每一種物品都是限量供應的背包問題。
int cost[N], weight[N], number[N];
// number[n]：第n種物品的數量。
void knapsack(int n, int w)
{
    for (int i = 0; i < n; ++i)
    {
        int num = min(number[i], w / weight[i]);
        for (int k = 1; num > 0; k *= 2)
        {
            if (k > num) k = num;
            num -= k;
            for (int j = w; j >= weight[i] * k; --j)
                c[j] = max(c[j], c[j - weight[i] * k] +
                     cost[i] * k);
        }
    }
    cout << "最高的價值為" << c[w];
}
```

5.2 找零問題

```
/*Money Changing Problem*/
// n：用第0種到第n種錢幣來湊得價位。
// m：欲湊得的價位值。
// c(n, m)：用第0種到第n種錢幣湊得價位m的湊法數目。
// price[n]：第n種錢幣的面額大小。

// 能否湊得某個價位（ Money Changing Problem ）
// 給定許多種不同面額的錢幣，
// 能否湊得某個價位？
// 每種面額的錢幣都無限供應。

int price[5] = {5, 2, 6, 11, 17};// 錢幣面額
bool c[1000+1];
// 這些面額湊不湊得到價位 m
void change(int m){
    memset(c, false, sizeof(c));
    c[0] = true;

    // 依序加入各種面額
    for (int i = 0; i < 5; ++i)
        // 由低價位逐步到高價位
        for (int j = price[i]; j <= m; ++j)
            // 湊、湊、湊
            c[j] |= c[j-price[i]];

    if (c[m])
        cout << "湊得到";
    else
        cout << "湊不到";
}
// 湊得某個價位的湊法總共幾種（ Coin Change Problem ）
void change(int m){
    memset(c, 0, sizeof(c));
    c[0] = 1;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            c[j] += c[j-price[i]];

    cout << "湊得價位" << m;
    cout << "湊法總共" << c[m] << "種";
}
// 湊得某個價位的最少錢幣用量（ Change-Making Problem
    ）
// c(n, m)：用第0種到第n種錢幣湊得價位m，最少所需要的錢
    幣數量。
void change(int m){
    memset(c, 0x7f, sizeof(c));
    c[0] = 0;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            c[j] = min(c[j], c[j-price[i]] + 1);

    cout << "湊得價位" << m;
    cout << "最少需（只）要" << c[m] << "個錢幣";
}
// 湊得某個價位的錢幣用量，有哪幾種可能性。
void change(int m){
    memset(c, 0, sizeof(c));
    c[0] = 1;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            // 錢幣數量加一，每一種可能性都加一。
            c[j] |= c[j-price[i]] << 1;

    for (int i = 1; i <= 63; ++i)
        if (c[m] & (1 << i))
            cout << "用" << i << "個錢幣可湊得價位" <<
                 m;
}
// 能否湊得某個價位，但是錢幣限量供應！
int price[5] = {5, 2, 6, 11, 17};
int number[5] = {4, 5, 5, 3, 2}; // 各種錢幣的供應數量
bool c[1000+1];
void change(int m){
    memset(c, 0, sizeof(c));
    c[0] = true;
    for (int i = 0; i < 5; ++i)
        // 各種餘數分開處理
        for (int k = 0; k < price[i]; ++k){
            int left = number[i]; // 補充彈藥
            // 由低價位到高價位
            for (int j = k; j <= m; j += price[i])
                // 先前的面額已能湊得，當前面額可以省著
                    用。
                if (c[j])
                    left = number[i];   // 補充彈藥
                // 過去都無法湊得，一定要用目前面額硬
                    湊。
                else if (left > 0){
                    left--; // 用掉一個錢幣
                    c[j] = true;
                }
        }
    if (c[m])
        cout << "湊得到";
```

```
    else
        cout << "湊不到";
}
// Cashier's Algorithm
// 買東西找回最少硬幣。
int price[5] = {50, 20, 10, 4, 2};  // 面額由大到小排列
void cashier(int n){ // n 是總共要找的錢。
    int c = 0;
    for (int i=0; i<5; ++i)
        while (n >= price[i])
        {
            n -= price[i];  // 找了 price[i] 元
            c++;
        }

    if (n != 0)
        cout << "找不出來";
    else
        cout << "找了" << c << "個錢幣";
}
```

6 DP on tree

6.1 全點對距離 Tree Distance

```
int dp[MAXN]={0};
void dfs_sz(int x,int f){
    sz[x] = 1, fa[x] = f;
    for(int i:edge[x]){
        if(i == f) continue;
        dfs1(i, x);  // 先計算完子節點的答案再算自己的
        sz[x]+=sz[i];
        dp[x]+=(dp[i]+sz[i]);
}   }
void dfs_dp(int x,int f,ll sum){
    ans += sum + dp[x]; //所有點到結點x距離總和為父節點
        方向距離總和 + 子樹到自己距離總和
    for(int i:edge[x]){
        if(i == f) continue;
        //tmp 為從父節點x到子節點i的距離總合為
        ll tmp = sum  //x的父節點總和 sum 到結點x的距離
                + dp[x] - (dp[i]+sz[i])
                //加上x的子樹(除了i方向)到x的距離總和
                + (n - sz[i]);
                //加上從節點x到節點i的距離
        dfs2(i, x, tmp);
}   }
```

6.2 最大獨立集 Independent set

```
int dp[MAXN][2]; //此點，選或不選
void dfs(int x,int f){
    dp[x][1] = 1; // 狀態[1] 計算自己數量 +1
    for(int i:edge[x]){
        if(i == f)    continue;
        dfs(i, x);    // 先計算完子節點的答案再算自己的
        dp[x][0] += max(dp[i][0], dp[i][1]);
        dp[x][1] += dp[i][0];
}   }
```

6.3 最小點覆蓋 Vertex Cover

```
int dp[MAXN][2]; //此點，選或不選
void dfs(int x,int f){
    dp[x][1] = 1; // 狀態[1] 計算自己數量 +1
    for(int i:edge[x]){
        if(i == f)    continue;
        dfs(i, x);    // 先計算完子節點的答案再算自己的
        dp[x][0] += dp[i][1];
        dp[x][1] += min(dp[i][0],dp[i][1]);
}   }
```

6.4 最小支配集 Dominating Set

```
//狀態
dp[i][0]: 點i屬於支配集，並且以點i為根的子樹都被覆蓋了
    的情況下，支配集中包含的最少點數。
dp[i][1]: 點i不屬於支配集，且以i為根的子樹都被覆蓋，且i
    被其中不少於1個子結點覆蓋的情況下，支配集包含的最少
    點數。
dp[i][2]: 點i不屬於支配集，且以i為根的子樹都被覆蓋，且i
    沒被子結點覆蓋的情況下，支配集包含的最少點數。
// 狀態轉移
dp[i][0] = 1 + Σmin( dp[u][0], dp[u][1], dp[u][2] )
if(i沒有子結點) dp[i][1] = INF
else dp[i][1] = Σmin( dp[u][0], dp[u][1] )
dp[i][2] = Σdp[u][1]
```

7 String

7.1 Trie

```
// insert O(|s|)
// query  O(|s|)
struct trie{
    trie *nxt[26];
    int cnt;    //紀錄有多少個字串以此節點結尾
    int sz;     //有多少字串的前綴包括此節點
    set<int> cnt_idx, sz_idx;
    trie():cnt(0),sz(0){
        memset(nxt,0,sizeof(nxt));
}   };
trie *root = new trie();
void insert(string& s, int idx){
    trie *now = root;  // 每次從根結點出發
    for(auto i:s){
        now->sz++; now->sz_idx.emplace(idx); //被誰經過
        if(now->nxt[i-'a'] == NULL){
            now->nxt[i-'a'] = new trie();
        }
        now = now->nxt[i-'a'];  //走到下一個字母
    }
    now->cnt++; now->cnt_idx.emplace(idx);//以此點結尾
    now->sz++;  now->sz_idx.emplace(idx); //被誰經過
}
//query
int query_prefix(string& s){  //查詢有多少前綴為 s
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL){
            return 0;
        }
        now = now->nxt[i-'a'];
    }
    return now->sz;
}
int query_count(string& s){  //查詢字串 s 出現次數
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] == NULL){
            return 0;
        }
        now = now->nxt[i-'a'];
    }
    return now->cnt;
}
//str有沒有在[l,r]的前綴中
bool query_ArrPrefix(string& s,int l,int r){
```

```
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] != NULL && now->nxt[i - 'a'
            ]->sz > 0 ){ //存在
            now = now->nxt[i-'a'];
        }else return false; //不存在，無解
    }
    // 這個s的節點，[l,r]有沒有經過
    auto L = now->sz_idx.lower_bound(l);
    if(l<=*L && *L<=r)return true;
    else return false;}
//[l,r]有沒有存在於str的前綴中
bool query_StrPrefix(string& s,int l,int r){
    trie *now = root;    // 每次從根結點出發
    for(auto i:s){
        if(now->nxt[i-'a'] != NULL && now->nxt[i - 'a'
            ]->sz > 0 ){ //存在
            now = now->nxt[i-'a'];
        }else return false; //不存在，無解
    // [l,r]存在於str的前綴中，代表有字串以str為結尾
        auto L = now->cnt_idx.lower_bound(l);
        if(l<=*L && *L<=r)return true;
    } return false;
}
```

7.2 01Trie

```
// insert O(lgx)
// query  O(lgx)
// 處理XOR問題
// struct
struct trie{
    trie *nxt[2];  // 差別
    int cnt;     //紀錄有多少個數字以此節點結尾
    int sz;      //有多少數字的前綴包括此節點
    trie():cnt(0),sz(0){
        memset(nxt,0,sizeof(nxt));
    }
};
//創建新的字典樹
trie *root = new trie();
void insert(int x){
    trie *now = root;  // 每次從根結點出發
    for(int i=30;i>=0;i--)
        now->sz++;
        if(now->nxt[x>>i&1] == NULL){
            now->nxt[x>>i&1] = new trie();
        }
        now = now->nxt[x>>i&1];  //走到下一個字母
    }
    now->cnt++;
    now->sz++;
}
// in this set, the maximum value of bitwise XOR x
int query(int x){
    trie *now = root;
    int ans=0;
    for(int i=30;i>=0;i--){// 不等於為1(0xr1=1,1xr0=1)
        if (now->nxt[!(x>>i&1)] != NULL && now->nxt[!(x
            >>i&1)]-> sz > 0){ //下一個存在
            ans += 1<<i;
            now = now->nxt[!(x>>i&1)];
        }
        else now = now->nxt[x>>i&1];
    }
    return ans;
}
```

7.3 Hash

```
// build O(n)
// query O(1)
// double hash
// P = 53,97,193,49157,805306457,1610612741,1e9+9,1e9+7
const ll P1 = 75577;
const ll P2 = 12721;    // 多一個質數 p2
const ll MOD = 998244353;
pair<ll,ll> Hash[MXN];//Hash[i] 為字串 [0,i] 的hash值
void build(const string& s){
    pair<ll,ll> val = make_pair(0,0);
    for(int i=0; i<s.size(); i++){
        val.first = (val.first * P1 + s[i]) % MOD;
        val.second = (val.second * P2 + s[i]) % MOD;
        Hash[i] = val;
    }
}
// query:
// H[l,r] = Hr - H(l-1) * p^(r-l+1) %MOD + MOD )%MOD
```

8 Math

8.1 Epsilon

```
/*精準度(Epsilon)*/
float eps = 1e-8;
bool Equal(float a, float b)
    return fabs(a - b) < eps
bool NEqual(float a, float b)
    return fabs(a - b) > eps
bool Less(float a, float b)
    return (a - b) < -eps
bool Greater(float a, float b)
    return (a - b) > eps
```

8.2 Floor-Ceil

```
/*floor向下取整，ceil向上取整*/
int floor(int a,int b){ return a/b - (a%b and a<0^b<0);
    }
int ceil (int a,int b){ return a/b + (a%b and a<0^b>0);
    }
```

8.3 josephusl

```
/*約瑟夫問題：n個人圍成一桌，數到m的人出列*/
int josephus(int n, int m) {  //n人每m次
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        ans = (ans + m) % i;
    return ans;
}
```

8.4 快速冪

```
/*快速冪*/
ll mypow(ll x, ll y, ll p) {
    long long ans = 1;
    while (y) {
        if (y & 1)    ans = ans * x % p;    //prime
        x = x * x % p;    //每次把自己平方
        y >>= 1;    //每次右移一格
    }
    return ans;
}
```

8.5 Sieve Prime

```
/*Sieve_Prime*/
const int N = 20000000;//質數表大小
bool sieve[N];
vector<int> prime;
void linear_sieve(){
    for (int i = 2; i < N; i++)
    {
        if (!sieve[i]) prime.push_back(i);
        for (int p : prime)
        {
            if (i * p >= N) break;
            sieve[i * p] = true;
            if (i % p == 0) break;
        }
    }
}
```

8.6 Prime factorization

```
/*質因數分解*/
list<int> breakdown(int N) {
  list<int> result;
  for (int i = 2; i * i <= N; i++) {
    if (N % i == 0) {  // 如果 i 能够整除 N，说明 i 为
        N 的一个质因子。
      while (N % i == 0) N /= i;
      result.push_back(i);
    }
  }
  if (N != 1) {  // 说明再经过操作之后 N 留下了一个素数
    result.push_back(N);
  }
  return result;
}
```

8.7 Miller Rabinn

```
/*Miller_Rabin 質數判定*/
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
ll magic[N] = {};
bool witness(ll a, ll n, ll u, int t) {
    if (!a) return 0;
    ll x = mypow(a, u, n);  //快速冪
    for (int i = 0; i < t; i++) {
        ll nx = mul(x, x, n);   //快速乘
        if (nx == 1 && x != 1 && x != n - 1) return 1;
        x = nx;
    }
    return x != 1;
}
bool miller_rabin(ll n) {
    int s = (magic number size);
        // iterate s times of witness on n
        if (n < 2) return 0;
    if (!(n & 1)) return n == 2;
    ll u = n - 1; int t = 0;
    // n-1 = u*2^t
    while (!(u & 1)) u >>= 1, t++;
    while (s--) {
        ll a = magic[s] % n;
        if (witness(a, n, u, t)) return 0;
    }
    return 1;
}
```

8.8 乘法取餘 Multiplication

```
/*大數乘法取餘數*/
ll mul(ll x, ll y, ll mod) {
    ll ret = x * y - (ll)((long double)x / mod * y) *
        mod;
    // LL ret=x*y-(LL)((long double)x*y/mod+0.5)*mod;
    return ret < 0 ? ret + mod : ret;
}
```

8.9 快速乘法 karatsuba

```
/*karatsuba 快速乘法*/
// Get size of the numbers
int getSize(ll num){
    int count = 0;
    while (num > 0)
    {
        count++;
        num /= 10;
    }
    return count;
}
ll karatsuba(ll X, ll Y){
    // Base Case
    if (X < 10 && Y < 10)
        return X * Y;
    // determine the size of X and Y
    int size = fmax(getSize(X), getSize(Y));
    // Split X and Y
    int n = (int)ceil(size / 2.0);
    ll p = (ll)pow(10, n);
    ll a = (ll)floor(X / (double)p);
    ll b = X % p;
    ll c = (ll)floor(Y / (double)p);
    ll d = Y % p;
    // Recur until base case
    ll ac = karatsuba(a, c);
    ll bd = karatsuba(b, d);
    ll e = karatsuba(a + b, c + d) - ac - bd;
    // return the equation
    return (ll)(pow(10 * 1L, 2 * n) * ac + pow(10 * 1L,
         n) * e + bd);
}
```

8.10 ax+by=gcd(a,b)

```
/*ax+by=gcd(a,b) 一組解*/
ll a, b, x, y;
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    }
    return x = 1, y = 0, a;
}
```

8.11 GaussElimination

```
/*GaussElimination*/
// by bcw_codebook
const int MAXN = 300;
const double EPS = 1e-8;
int n;
double A[MAXN][MAXN];
void Gauss() {
  for(int i = 0; i < n; i++) {
    bool ok = 0;
    for(int j = i; j < n; j++) {
      if(fabs(A[j][i]) > EPS) {
        swap(A[j], A[i]);
```

```
        ok = 1;
        break;
      }
    }
    if(!ok) continue;

    double fs = A[i][i];
    for(int j = i+1; j < n; j++) {
      double r = A[j][i] / fs;
      for(int k = i; k < n; k++) {
        A[j][k] -= A[i][k] * r;
      }
    }
  }
}
```

8.12 大數 Big number

```
/*大數(Big Number)*/
void add(int a[100], int b[100], int c[100]){
    int i = 0, carry = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] + b[i] + carry;
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void sub(int a[100], int b[100], int c[100]){
    int i = 0, borrow = 0;
    for (i = 0; i < 100; ++i) {
        c[i] = a[i] - b[i] - borrow;
        if (c[i] < 0) {
            borrow = 1;
            c[i] += 10;
        }
        else
            borrow = 0;
    }
}
void mul(int a[100], int b[100], int c[100]){
    int i = 0, j = 0,carry = 0;
    for (i = 0; i < 100; ++i) {
        if (a[i] == 0) continue;
        for (j = 0; j < MAX; ++j)
            c[i + j] += a[i] * b[i];
    }
    for (i = 0; i < MAX; ++i) {
        carry = c[i] / 10;
        c[i] %= 10;
    }
}
void div(int a[100], int b[100], int c[100]){
    int t[100];
    for (i = 100 - 1; i >= 0; i--) {
        for (int k = 9; k > 0; k--) // 嘗試商數
        {
            mul(b + i, k, t);
            if (largerthan(a + i, t))
            {
                sub(a + i, t, c + i);
                break;
            }
        }
    }
}
```

9 STL

9.1 常用 tool

```
swap(a,b);
min(a,b);
max({ a, b, c });
//二進制"1"的個數
__builtin_popcount(n)   -> int
__builtin_popcountl(n)  -> long int
__builtin_popcountll(n) -> long long
//math
abs(x);
pow(x);
sqrt(x);
__gcd(x, y);
__lg(x)  //以2為底數
log(x)   //以e為底數
log10(x) //以10為底數
do {     //排列組合
    cout << s << "\n";
} while (next_permutation(s.begin(), s.end()));
//陣列處理
sort(arr,arr+n);
reverse(arr,arr+n);
*min_element(arr, arr+n); //value
min_element(arr, arr+n) - arr; //index
*lower_bound(arr, arr+4, c) << '\n'; //第一個大於等於c
*upper_bound(arr, arr+4, c) << '\n'; //第一個大於c
//填充 arr[0]=123 arr[1]=123 arr[2]=123
fill(arr, arr+3, 123);
//輸出
//四捨五入 或是更高精度(int)10 * 位數 + 0.5
cout << fixed << setprecision(10);
//寬度n 用char(c)填補
cout << setw(n) << setfill(c) << ;
//迭代器
T.begin()
T.end()
T.rbegin() //逆序迭代器
T.rend() //逆序迭代器
T.find() //可用於set,map的earse()。
```

9.2 Sort

```
//cmp
struct T {int val, num;};
bool cmp(const T &a, const T &b) {
    return a.num < b.num;
}
sort(arr.begin(), arr.end(), cmp);
//operator
struct Point {
    int x, y;
    bool operator<(Point b) {
        if (x != b.x) return x < b.x;
        else return y < b.y;
    }
};
Point arr[n];
sort(arr, arr+n); //二維平面，從小到大排列。
```

9.3 Priority Queue

```
//預設由大排到小
priority_queue<T> pq
priority_queue<int, vector<int>, less<int> > pq;
//改成由小排到大
priority_queue<T, vector<T>, greater<T> > pq;
//自行定義 cmp 排序
priority_queue<T, vector<T>, cmp> pq;
struct cmp {
    bool operator()(node a, node b) {
        //priority_queue優先判定為!cmp
  //，所以「由大排到小」需「反向」定義
        //實現「最小值優先」
        return a.x < b.x;
```

```
    }
};
```

9.4 Bitset

```
//init
b.set();    //每個位元設 '1'
b.reset();  //每個位元設 '0'
b[pos] = 1;
//轉換
s = b.to_string();
unsigned long x = b.to_ulong();
//sum
b.any();//判別是否有 '1'
b.none();//判別是否沒 '1'
cnt = b.count();// 判別 '1' 之個數
cnt = b.size() - b.count();//判別 '0' 之個數
```

9.5 StringStream

```
stringstream ss;
getline(cin, str);
ss.str("");
ss.clear();
ss << oct << s;    //以8進制讀入流中
ss << hex << s;    //以16進制讀入流中
ss >> n;         //10進制int型輸出
ss >> s;         //x進制str型輸出
```

9.6 List

```
• push_back()
• pop_back()
• push_front()
• pop_front()
• back()
• front()
• insert(index, obj)
• erase()
//遍歷
for (auto iter = _list.begin(); iter != _list.end();
    iter++)
    cout << *iter << "\n";
```

9.7 Set

```
• insert()
• erase(l, r) //l與r皆為iterator
• erase()
• empty()
• clear()
• count() //元素是否存在
//遍歷
int mints[] = { 75,23,65,42,13,75,65 };
set<int> myset(myints, myints + 7);
for (auto it = myset.begin(); it != myset.end(); it++)
    cout << ' ' << *it;
```

9.8 Map

```
//find
auto iter = mymap.find("a");
if (iter != mapStudent.end())
    cout << "Find, the value is" << iter->second <<
        endl;
else
    cout << "Do not Find" << endl;
//erase
auto iter = mymap.find("a");
mymap.erase(iter);
//map遍歷
for (auto it = mymap.begin(); it != mymap.end(); it++)
cout << it->first << ", " << it->second << endl
```

10 Other

```
/*前置作業*/
#include <bits/stdc++.h>
#define ll long long
#define ld long double
using namespace std;
cin.tie(0);cout.tie(0);
ios_base::sync_with_stdio(false);
/*unroll-loops*/
#pragma GCC optimize("O0")//不優化(預設)
#pragma GCC optimize("O1")//優化一點
#pragma GCC optimize("O2")//優化更多
#pragma GCC optimize("O3")//O2優化再加上inline函式優化
#pragma GCC optimize("unroll-loops")
/*常數宣告*/
// 數字中可以加 ' 方便看出幾位數
#define MXN 1'000'005
// 1e-6 為科學記號 代表 1 * 10^-6
#define EPS 1e-6
// 0x3f3f3f3f為一個接近10^9的數字0x為16進位
#define INF 0x3f3f3f3f
// acos(-1) 等同圓周率
#define PI acos(-1)
/*位元運算*/
if(x&1) cout<<奇數;
else    cout<<偶數;
x <<= 1    //將x左移1，等同 *2
x >>= 2    //將x右移2，等同 /4
```

```
                                  @
               (__)    (__) ____/ Hong~Long~Long~Long~
            /| (oo) _  (oo)/----/______   *
  _o\______/_|\_\/_/_|__\/|____|//////== *- *  * -
 /__________  \   AC |  AC | NO BUG /== -*
[_____/^^\_____\_____|_____/^^\_____]     *- * -
      \__/                 \__/      Chong~Chong~Chong~
```