gcd&lcm

```
int gcd(int a, int b){
	if (!b)
		return a;
	return gcd(b, a % b);
}
int lcm(int a, int b) {
	return a / gcd(a, b) * b;
}
```

前缀和

```
//一维前缀和
int n, m, l, r, a[100005];
long long sum_a[100005];
signed main(){
	scanf("%d%d",&n,&m);
	for(int i=1;i<=n;i++){
		scanf("%d",&a[i]);
		sum_a[i]=a[i]+sum_a[i-1];
	}
	while(m--){
		scanf("%d%d",&l,&r);
		printf("%lld\n", sum_a[r]-sum_a[l-1]);
	}
	return 0;
}

//二维前缀和
int main(){
	int n, m, t;
	scanf("%d%d%d", &n, &m, &t);
	vector<vector<long long >> s(n+1, vector<long long>(m+1,0));
	for(int i=1;i<=n;i++){
		for(int j=1;j<=m;j++){
			scanf("%lld", &s[i][j]);
			s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];
		}
	}
	while(t--){
		int a, b, c, d;
		scanf("%d%d%d%d", &a, &b, &c, &d);
		long long ans=s[b][d]-s[a-1][d]-s[b][c-1]+s[a-1][c-1];
```

```
        printf("%lld\n", ans);
    }
    return 0;
}
```

差分

```
//一维
int main(){
    int n, m;
    scanf("%d%d", &n, &m);
    vector<long long> s(n+1,0);
    while(m--){
        int l, r, v;
        scanf("%d%d%d", &l, &r, &v);
        s[l]+=v;
        if(r<n) s[r+1]-=v;
    }
    for(int i=1;i<=n;i++){
        s[i]+=s[i-1];
        printf("%lld ", s[i]);
    }
    return 0;
}

//二维差分
int main(){
    int n, m, t;
    scanf("%d%d%d", &n, &m, &t);
    vector<vector<long long >> s(n+2, vector<long long>(m+2,0));
    while(t--){
        int a, b, c, d, x;
        scanf("%d%d%d%d%d", &a, &b, &c, &d, &x);
        s[a][c]+=x;
        if(b<n) s[b+1][c]-=x;
        if(d<m) s[a][d+1]-=x;
        if(b<n&&d<m) s[b+1][d+1]+=x;
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];
            printf("%lld ", s[i][j]);
        }
        printf("\n");
```

```
	}
	return 0;
}
```

二分

```
//查找>=x
void solve(){
	int n, x, a[100];
	cin>>n>>x;
	for(int i=0;i<n;i++) cin>>a[i];
	//sort(a,a+n);
	int l=0, r=n-1;
	while(l<r){
		int mid=(l+r)>>1;
		if(a[mid]>=x) r=mid;
		else l=mid+1;
	}
	cout<<l<<'\n';
}

//查找<=x
void solve(){
	int n, x, a[100];
	cin>>n>>x;
	for(int i=0;i<n;i++) cin>>a[i];
	//sort(a,a+n);
	int l=0, r=n-1;
	while(l<r){
		int mid=(l+r+1)>>1;
		if(a[mid]<=x) l=mid;
		else r=mid-1;
	}
	cout<<l<<'\n';
}
```

二叉树遍历

```
string inor, aft;
void solve(string inor, string aft){
	if (aft.empty()) return ;
	int l = aft.size()-1;
	char root = aft[l];
	printf("%c", root);
```

```
        int k = inor.find(root);
        aft.erase(l);
        string linor = inor.substr(0, k);
        string rinor = inor.substr(k+1);
        string laft = aft.substr(0, k);
        string raft = aft.substr(k);
        solve(linor, laft);
        solve(rinor, raft);
}
int main(){
        cin>>inor>>aft;
        solve(inor, aft);
        return 0;
}
```

单调栈

```
//模板分析
vector<int> h(n+1);//原数组
vector<int> c(n+1);//答案数组
stack<int> st;

for(int i = 0;i < n;i++){
    //如果 目前的数 大于 栈顶的数 则踢走栈顶的数
        while(!st.empty() && h[i] > h[st.top()]){
                st.pop();
        }
        if(!st.empty()){
            //此处记录的是元素下标
                c[i] = st.top();
        }
        else c[i] = -1;
        st.push(i);
}

//如果求右边的数
for(int i = nums.size()-1; i >= 0; i--){
        while(!st.empty() && nums[i] >= st.top()){
                st.pop();
        }
        if(!st.empty()){
                res[i] = st.top();
        }
        else res[i] = -1;
    //如果需要  数组元素  并非下标
        st.push(nums[i]);
}
```

二叉搜索树

```
typedef struct SortTree {
        int data;
        struct SortTree* left;
        struct SortTree* right;
}Node;

Node* root;//根节点
int deep;

//初始化
void Init(int key) {
        root = (Node*)malloc(sizeof(Node));
        root->data = key;
        root->left = NULL;
        root->right = NULL;
}

//节点插入
void insert(int key) {
        Node* temp = root;//方便移动    以及    跳出循环
        Node* prev = NULL;//定位到待插入位置的前一个结点
        while (temp != NULL) {
                prev = temp;
                if (key < temp->data) {
                        temp = temp->left;
                } else if (key > temp->data) {
                        temp = temp->right;
                } else {
                        return;
                }
        }

        if (key < prev->data) {
                prev->left = (Node*)malloc(sizeof(Node));
                prev->left->data = key;
                prev->left->left = NULL;
                prev->left->right = NULL;
        } else {
```

```
		prev->right = (Node*)malloc(sizeof(Node));
		prev->right->data = key;
		prev->right->left = NULL;
		prev->right->right = NULL;
	}
}

//删除操作 留个坑以后会填
int delete_node(Node* node, int key)
{
	if (node == NULL)
	{
		return -1;
	}
	else
	{
		if (node->data == key)
		{
			//当我执行删除操作 需要先定位到前一个结点
			Node* tempNode = prev_node(root, node, key);
			Node* temp = NULL;
			/*
			如果右子树为空 只需要重新连接结点
			叶子的情况也包含进去了 直接删除
			*/
			if (node->right == NULL)
			{
				temp = node;
				node = node->left;
				/*为了判断 待删除结点是前一个结点的左边还是右边*/
				if (tempNode->left->data == temp->data)
				{
					Node* free_node = temp;//释放用的指针
					tempNode->left = node;
					free(free_node);
					free_node = NULL;
				}
				else
				{
					Node* free_node = temp;//释放用的指针
					tempNode->right = node;
					free(free_node);
					free_node = NULL;
				}
```

```
}
else if (node->left == NULL)
{
        temp = node;
        node = node->right;
        if (tempNode->left->data == temp->data)
        {
                Node* free_node = temp;//释放用的指针
                tempNode->left = node;
                free(free_node);
                free_node = NULL;
        }
        else
        {
                Node* free_node = temp;//释放用的指针
                tempNode->right = node;
                free(free_node);
                free_node = NULL;
        }
}
else//左右子树都不为空
{
        temp = node;
        /*往左子树  找最大值*/
        Node* left_max = node;//找最大值的临时指针
        left_max = left_max->left;//先到左孩子结点
        while (left_max->right != NULL)
        {
                temp = left_max;
                left_max = left_max->right;
        }
        node->data = left_max->data;
        if (temp != node)
        {
                temp->right = left_max->left;
                free(left_max);
                left_max = NULL;
        }
        else
        {
                temp->left = left_max->left;
                free(left_max);
                left_max = NULL;
        }
```

```
                        }

                }
                else if(key < node->data)
                {
                        delete_node(node->left, key);
                }
                else if (key > node->data)
                {
                        delete_node(node->right, key);
                }
        }
}

//查找元素 key
bool search(Node* root, int key) {
        while (root != NULL)
        {
                if (key == root->data)
                        return true;
                else if (key < root->data)
                        root = root->left;
                else
                        root = root->right;
        }
        return false;
  }

//  返回节点深度
int deepSearch(int key, Node* root) {
        deep = 0;
        while (root != NULL) {
                deep++;
                if (key < root->data) {
                        root = root->left;
                } else {
                        root = root->right;
                }
        }
        return deep;
}

//中序遍历
void show(Node* root) {
	if (root == NULL)
	{
		return;
	}
	show(root->left);
	printf("%d ", root->data);
	show(root->right);
}

int main() {
	int n, num;
	cin >> n;
	cin >> num;
	Init(num);
	for (int i=1;i<n;i++) {
		cin >> num;
		insert(num);
	}
	show();
	return 0;
}
```

最大子段和

```
int n, a, b, ans=-2147483647;
int main(){
	cin>>n;
	for(int i=1;i<=n;i++){
		cin>>a;
		if(i==1) b=a;
		else b=max(a,a+b);
		ans=max(ans,b);
	}
	cout<<ans;
	return 0;
}
```

最短路

```
int main(){
	std::ios::sync_with_stdio(false);
	std::cin.tie(nullptr);
	int n, m, s;
	std::cin >> n >> m >> s;
	s--;

	const ll inf = ll(2e18);
	std::vector<std::vector<std::array<ll, 2>>> adj(n);
	while (m--){
		int u, v;
		ll w;
		std::cin >> u >> v >> w;
		u--, v--;
		adj[u].push_back({v, w});
	}

	std::priority_queue<std::array<ll, 2>> q;
	std::vector<ll> dis(n, inf);
	dis[s] = 0;
	q.push({-dis[s], s});
	while (!q.empty()){
		auto [d, u] = q.top();
		d = -d;
		q.pop();

		if (d > dis[u]) continue;
		for (auto [v, w] : adj[u]){
			if (dis[v] > dis[u] + w){
				dis[v] = dis[u] + w;
				q.push({-dis[v], v});
			}
		}
	}

	for (int i = 0; i < n; i++){
		std::cout << dis[i] << " \n"[i == n - 1];
	}
	return 0;
}
```

线段树

```
const int maxn = 1e5 + 10;
int n, q, m, a[maxn];

struct seg_tree{

	struct node{
		int l, r;
		ll v, mul, add;
	}tr[maxn*4];

	void update(int p) {
		tr[p].v = (tr[p<<1].v + tr[p<<1|1].v) % m;
	}

	void build(int p, int l, int r) {
		tr[p].l = l, tr[p].r = r, tr[p].mul = 1;
		if (l == r) {
			tr[p].v = a[l] % m;
			return;
		}
		int m = (l+r) >> 1;
		build(p<<1, l, m);
		build(p<<1|1, m+1, r);
		update(p);
	}
	//important code
	void push_down(int p) {
		//儿子 = 儿子*mul + 儿子长度*add
		tr[p<<1].v = (tr[p<<1].v*tr[p].mul + (tr[p<<1].r-tr[p<<1].l+1) * tr[p].add) % m;
		tr[p<<1|1].v = (tr[p<<1|1].v*tr[p].mul + (tr[p<<1|1].r-tr[p<<1|1].l+1) * tr[p].add) %
m;
		//维护 lazytag
		tr[p<<1].mul = (tr[p<<1].mul*tr[p].mul) % m;
		tr[p<<1|1].mul = (tr[p<<1|1].mul*tr[p].mul) % m;
		tr[p<<1].add = (tr[p<<1].add*tr[p].mul+tr[p].add) % m;
		tr[p<<1|1].add = (tr[p<<1|1].add*tr[p].mul+tr[p].add) % m;
		//初始化
		tr[p].mul = 1, tr[p].add = 0;
	}

	void Mul(int p, int l, int r, int k) {
		if(tr[p].l >= l && tr[p].r <= r) {
			tr[p].add = (tr[p].add * k) % m;
			tr[p].mul = (tr[p].mul * k) % m;
			tr[p].v = (tr[p].v * k) % m;
			return;
		}
		push_down(p);
		int mid = (tr[p].l + tr[p].r) >> 1;
		if (l <= mid) Mul(p<<1, l, r, k);
```

```
			if (r > mid) Mul(p<<1|1, l, r, k);
			update(p);
		}

		void Add(int p, int l, int r, int k) {
			if(tr[p].l >= l && tr[p].r <= r) {
				tr[p].add = (tr[p].add + k) % m;
				tr[p].v = (tr[p].v + k * (tr[p].r - tr[p].l + 1)) % m;
				return;
			}
			push_down(p);
			int mid = (tr[p].l + tr[p].r) >> 1;
			if (l <= mid) Add(p<<1, l, r, k);
			if (r > mid) Add(p<<1|1, l, r, k);
			update(p);
		}
		//区间查询
		ll query(int p, int l, int r) {
			if(tr[p].l >= l && tr[p].r <= r) {
				return tr[p].v;
			}
			push_down(p);
			ll s = 0;
			int mid = (tr[p].l + tr[p].r) >> 1;
			if (l <= mid) s = (s + query(p<<1, l, r)) % m;
			if (r > mid) s = (s + query(p<<1|1, l, r)) % m;
			return s;
		}
}ST;
signed main() {
		scanf("%d%d%d", &n, &q, &m);
		for(int i=1;i<=n;i++) scanf("%d", &a[i]);
		ST.build(1, 1, n);

		while(q--) {
			int c, x, y;
			scanf("%d%d%d", &c, &x, &y);
			if (c == 1){
				int k;
				scanf("%d", &k);
				ST.Mul(1, x, y, k);
			} else if(c == 2){
				int k;
				scanf("%d", &k);
```

```
                ST.Add(1, x, y, k);
            } else {
                printf("%lld\n", ST.query(1, x, y));
            }
        }
        return 0;
}
```

组合数

```
//求组合数，在 n 个中取 m 个
ll C(int n, int m) {
        if (m < n - m) m = n - m;
        ll ans = 1;
        for (int i=m+1;i<=n;i++) ans *= i;
        for (int i=1;i<=n-m;i++) ans /= i;
        return ans;
}
```

进制哈希

```
//核心：将字符串转换为不能高概率重复的数字，进行排序再计数
typedef unsigned long long ull;
ull base = 131, mod = 212370440130137957ll;
ull a[10010];
char s[10010];
int n, ans = 1, prime = 233317;//大质数，降低哈希冲突

ull hashe(char s[]) {
        ull ans = 0;
        for (int i=0;i<strlen(s);i++) {
                ans = (ans*base+(ull)s[i])%mod+prime;
        }
        return ans;
}

int main() {
        scanf("%d", &n);
        for(int i=1;i<=n;i++) {
                scanf("%s", s);
                a[i] = hashe(s);
        }
        sort(a+1, a+n+1);
        for(int i=1;i<n;i++) {
```

```
		if(a[i+1]!=a[i]) ans++;
	}
	printf("%d", ans);
	return 0;
}
```

逆元&快速幂

```
//求非质数逆元
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll p=1e9+7;
ll mod(ll number) {return number % p;}
void exgcd(ll a, ll b, ll &x, ll &y){
	if(!b){
		x=1, y=0;
		return ;
	}
	exgcd(b, a%b, y, x);
	y-=(a/b)*x;
}
ll inv(ll number, ll p){
	ll x=1, y=0;
	exgcd(number, p, x, y);
	return (x%p+p)%p;
}
signed main(){
	ll k;cin>>k;
	ll k_inv=inv(k,p);
	cout<<k_inv;
	return 0;
}

//求质数逆元
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
//ll p=1e9+7;
ll p, k;
ll mod(ll number) {return number % p;}
ll fast_pow(ll number, ll power){
	ll ans=1LL;number%=p;
	while(power){
		if(power&1) ans=mod(ans*number);
		number=mod(number*number);
		power >>= 1;
	}
	return ans;
}
ll inv(ll number, ll p){
	return fast_pow(number, p-2);
}
signed main(){
	cin>>p>>k;
	//7 2CR
	ll k_inv=inv(k,p);
	cout<<k_inv;//4
	return 0;
}
```

高精

乘法

```
string mul(string a, string b) {
	string s;
	int na[L], nb[L], nc[L], La=a.size(), Lb=b.size();
	fill(na, na+L, 0);fill(nb, nb+L, 0);fill(nc, nc+L, 0);
	for(int i=La-1;i>=0;i--) na[La-i] = a[i] - '0';
	for(int i=Lb-1;i>=0;i--) nb[Lb-i] = b[i] - '0';
	for (int i=1;i<=La;i++) {
		for (int j=1;j<=Lb;j++) {
			nc[i+j-1] += na[i]*nb[j];
		}
	}
	for (int i=1;i<=La+Lb;i++) {
		nc[i+1] += nc[i]/10, nc[i]%=10;
	}
	if (nc[La+Lb]) s+=nc[La+Lb]+'0';
	for (int i=La+Lb;i>=1;i--) {
		s += nc[i]+'0';
	}
	return s;
}

//加法
string add(string a, string b) {
	string s;
```

```
    int na[L], nb[L], nc[L], La=a.size(), Lb=b.size(), Lc=1;
    fill(na, na+L, 0);fill(nb, nb+L, 0);fill(nc, nc+L, 0);
    for(int i=La-1;i>=0;i--) na[La-i] = a[i] - '0';
    for(int i=Lb-1;i>=0;i--) nb[Lb-i] = b[i] - '0';

    int x = 0;
    while(Lc <= La || Lc <= Lb) {
        nc[Lc] = na[Lc] + nb[Lc] + x;
        x = nc[Lc]/10, nc[Lc] %= 10;
        Lc++;
    }
    nc[Lc] = x;
    if(nc[Lc] == 0) Lc--;
    for (int i=Lc;i>=1;i--) {
        s += nc[i]+'0';
    }
    return s;
}

//减法
string Minus(string a, string b) {
    string s;
    int na[L], nb[L], nc[L], La=a.size(), Lb=b.size(), Lc=1;
    fill(na, na+L, 0);fill(nb, nb+L, 0);fill(nc, nc+L, 0);
    for(int i=La-1;i>=0;i--) na[La-i] = a[i] - '0';
    for(int i=Lb-1;i>=0;i--) nb[Lb-i] = b[i] - '0';

    int i = 1;
    while(i <= La || i <= Lb) {
        if(na[i] < nb[i]) {
            nc[i] = na[i] + 10 - nb[i];
            na[i+1]--; //借位处理
        } else {
            nc[i] = na[i] - nb[i];
        }
        i++;
    }
    Lc = i;
    while(nc[Lc] == 0 && Lc > 1) Lc--;
    for (int i=Lc;i>=1;i--) {
        s += nc[i]+'0';
    }
    return s;
}
```