高精度运算

高精度加法：

```
int main() {
    scanf("%s%s",&a1,&b1);
    if(a1[0] == '0' && b1[0] == '0') {
        cout << "0";
        return 0;
    }
    for(int i = 0;i < strlen(a1);++i)
        a[strlen(a1) - i - 1] = a1[i] - '0';
    for(int i = 0;i < strlen(b1);++i)
        b[strlen(b1) - i - 1] = b1[i] - '0';
    m = max(strlen(a1),strlen(b1));
    for(int i = 0;i < m;++i)
        c[i] = a[i] + b[i];
    for (int i = 0;i <= m;++i) {
        c[i + 1] = c[i + 1] + c[i] / 10;
        c[i] = c[i] % 10;
    }
    m++;
    while(!c[m]) m--;
    for(int i = m;i >= 0;--i)
        cout << c[i];
    return 0;
}
```

高精度减法：

```
int main() {
    scanf("%s%s",&x,&y);
    int lena = strlen(x),lenb = strlen(y);
    for(int i = 0;i < lena;++i)
        a[lena - i - 1] = x[i] - '0';
    for(int i = 0;i < lenb;++i)
        b[lenb - i - 1]=y[i] - '0';
    if(lena > lenb) vis = 1;
    if(lena < lenb) vis = 0;
    if(lena == lenb) {
        string temp1 = x,temp2 = y;
        if(temp1 > temp2) vis = 1;
        if(temp1 == temp2) vis = 1;
        else vis = 0;
    }
    m = max(lena,lenb);
    if(vis) {
        for(int i = 0;i < m;++i)
        c[i]=a[i]-b[i];
    }
    else {
        for(int i = 0;i < m;++i)
            c[i]=b[i]-a[i];
    }
    for(int i = 0;i <= m;++i) {
        if(c[i] < 0){
            c[i] += 10;
            c[i + 1]--;
        }
    }
    temp = m;
    for(int i = m;i >= 0;--i) {
        if(c[i]) break;
        temp--;
        if(i == 0 && c[i] == 0) {
            temp = 0;
            break;
        }
    }
    if(!vis) cout << "-";
    for(int i = temp;i >= 0;--i)
        cout << c[i];
    return 0;
}
```

高精度乘法：

```
int main() {
    scanf("%s%s",&x,&y);
    int lena = strlen(x),lenb = strlen(y);
    for(int i = 0;i < lena;++i)
        a[lena - i - 1] = x[i] - '0';
    for(int i = 0;i < lenb;++i)
        b[lenb - i - 1] = y[i] - '0';
    for(int i = 0;i < lena;++i)
        for(int j = 0;j < lenb;++j) {
            c[i + j] += a[i] * b[j];
            c[i + j + 1] += c[i+j] / 10;
            c[i + j] %= 10;
        }
    int lenc = lena + lenb;
    while(lenc > 1 && c[lenc - 1] == 0) lenc--;
    lenc--;
    for(int i = lenc;i >= 0;--i)
        cout << c[i];
    return 0;
}
```

数学

快速幂：

```
LL power(LL a,LL b) {
	LL t = 1,y = a;
	while(b) {
		if (b & 1) t = t * y % k;
		y = y * y % k;
		b >>= 1;
	}
   return t;
}
```

埃式素数判定：

```
bool is_prime(int x) {
	if(x == 1 || x == 0) return false;
	for(int i = 2;i * i <= x;++i)
		if(x % i == 0) return false;
	return true;
}
```

欧拉筛素数：

```
void is_prime(int list) {
	memset(vis,true,sizeof vis);
	vis[0] = vis[1] = false;
	for(int i = 2;i <= list;++i){
		if(vis[i]) prime[++tot] = i;
		for(int j = 1;j <= list && i * prime[j] <= list;++j){
			vis[i * prime[j]] = false;
			if(i % prime[j] == 0) break;
		}
	}
}
```

欧几里得算法：

```
int gcd(int a,int b) {
    if(!b) return a;
    else return gcd(b,a % b);
}
```

拓展欧几里得：

```
void ex_gcd(int &x,int &y,int a,int b) {
    if(b == 0) {
        y = 0;
        x = 1;
    }
    else {
        ex_gcd(y,x,b,a % b);
        y -= x * (a / b);
    }
}
```

欧拉函数：

```
int euler(int n) {
    int res = n;
    for(int i = 2;i * i <= n;++i) {
        if(n % i == 0) res = res / i * (i - 1);
        while(n % i == 0) n /= i;
    }
    if(n > 1) res = res / n * (n - 1);
    return res;
}
```

线性筛欧拉函数：

```
void get_phi(int list) {
    memset(vis,true,sizeof vis);
    vis[0] = vis[1] = false;
    phi[1] = 1;
    for(int i = 2;i <= list;++i){
        if(vis[i]) prime[++tot] = i,phi[i] = i - 1;
        for(int j = 1;j <= list && i * prime[j] <= list;++j){
            vis[i * prime[j]] = false;
            if(i % prime[j]) phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            else {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
        }
        phi[i] += phi[i - 1];
    }
}
```

矩阵运算：

```
inline Mat Mul(Mat a,Mat b){
    Mat tmp;
    tmp.clear();
    for(int i = 1;i <= n;++i)
        for(int j = 1;j <= n;++j)
            for(int k = 1;k <= n;++k)
                tmp.a[i][j] = (tmp.a[i][j] + (a.a[i][k] * b.a[k][j]) % MOD) % MOD;
    return tmp;
}
inline Mat power(Mat a,long long b){
    Mat ans;
    ans.clear();
    for(int i = 1;i <= n;++i)
        ans.a[i][i]=1;
    while(b){
        if (b & 1) ans = Mul(ans,a);
        a = Mul(a,a);
        b >>= 1;
    }
    return ans;
}

inline Mat Add(Mat T_1,Mat T_2) {
    Mat Tmp;
    Tmp.clear();
    for(int i = 1;i <= n;++i)
        for(int j = 1;j <= n;++j)
            Tmp.a[i][j] = T_1.a[i][j] + T_2.a[i][j],Tmp.a[i][j] %= MOD;
    return Tmp;
}
```

图论

SPFA 算法：

```cpp
void Spfa() {
	queue <int > Q;
	Q.push(s);
	vis[s] = true;
	memset(dis,0x3f,sizeof dis);
	dis[s] = 0;
	while(!Q.empty()) {
		int u = Q.front();
		Q.pop();
		vis[u] = false;
		for(int i = head[u];i;i = e[i].nxt) {
			int v = e[i].to;
			if(dis[v] > dis[u] + e[i].dis) {
				dis[v] = dis[u] + e[i].dis;
				if(!vis[v]) {
					Q.push(v);
					vis[v] = true;
				}
			}
		}
	}
}
```

Dijkstra 算法：

```cpp
void Dijkstra() {
	memset(dis,0x3f,sizeof dis);
	dis[s] = 0;
	priority_queue <pair <int ,int >,vector <pair <int ,int > >,greater <pair <int ,int > > > Q;
	Q.push(make_pair(0,s));
	while(!Q.empty()) {
		int u = Q.top().second;
		Q.pop();
		if(vis[u]) continue;
		vis[u] = true;
		for(int i = head[u];i;i = e[i].nxt) {
			int v = e[i].to;
			if(dis[v] > dis[u] + e[i].dis) {
				dis[v] = dis[u] + e[i].dis;
				if(!vis[v]) Q.push(make_pair(dis[v],v));
			}
		}
	}
}
```

最短路计数：

```
void Dijkstra_Heap() {
    priority_queue <pair <int ,int > ,vector <pair <int ,int > >,greater <pair <int ,int > > > Q;
    memset(dis,0x3f,sizeof dis);
    dis[s] = 0;
    ans[1] = 1;
    Q.push(make_pair(0,s));
    while(!Q.empty()) {
        int u = Q.top().second;
        Q.pop();
        if(vis[u]) continue;
        vis[u] = true;
        for(int i = head[u];i;i = e[i].nxt) {
            int v = e[i].to;
            if(dis[v] > dis[u] + 1) {
                dis[v] = dis[u] + 1;
                ans[v] = ans[u];
                if(!vis[v]) Q.push(make_pair(dis[v],v));
            }
            else if(dis[v] == dis[u] + 1) {
                ans[v] += ans[u];
                ans[v] %= MOD;
            }
        }
    }
}
```

负环：

```
    memset(dis,0x3f,sizeof dis);
    dis[1] = 0;
    Q.push(1);
    vis[1] = true;
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        vis[u] = false;
        for(int i = head[u];i;i = e[i].nxt) {
            int v = e[i].to;
            if(dis[v] > dis[u] + e[i].dis) {
                dis[v] = dis[u] + e[i].dis;
                num[v] = num[u] + 1;
                if(num[v] >= n) return true;
                if(!vis[v]) {
                    Q.push(v);
                    vis[v] = true;
                }
            }
        }
    }
    return false;
}
```

Tarjan 割点：

```
void Tarjan(int u,int fa) {
    dfn[u] = low[u] = ++cmt;
    int child = 0;
    for(int i = head[u];i;i = e[i].nxt) {
        int v = e[i].to;
        if(!dfn[v]) {
            Tarjan(v,fa);
            low[u] = min(low[u],low[v]);
            if(low[v] >= dfn[u] && u != fa) cut[u] = true;
            if(u == fa) child++;
        }
        else low[u] = min(low[u],dfn[v]);
    }
    if(u == fa && child >= 2) cut[fa]=true;
}
```

Tarjan 强连通分量（缩点）：

匈牙利算法：

```
inline bool dfs(int u) {
    for(int i = head[u];i;i = e[i].next) {
        int v = e[i].to;
        if(!used[v]) {
            used[v] = true;
            if(!Matched[v] || dfs(Matched[v])) {
                Matched[v] = u;
                return true;
            }
        }
    }
    return false;
}
```

倍增 LCA：

```cpp
inline void dfs(int now,int f) {
    deepth[now] = deepth[f] + 1;
    fa[now][0] = f;
    for(int i = 1;(1 << i) <= deepth[now];++i) fa[now][i] = fa[fa[now][i - 1]][i - 1];
    for(int i = head[now];i;i = e[i].next) {
        int v = e[i].to;
        if(v != f) dfs(v,now);
    }
}

inline int get_lca(int x,int y) {
    if(deepth[x] < deepth[y]) swap(x,y);
    while(deepth[x] > deepth[y]) x = fa[x][lg[deepth[x] - deepth[y]] - 1];
    if(x == y) return x;
    for(int k = lg[deepth[x]];k >= 0;--k) if(fa[x][k] != fa[y][k]) x = fa[x][k],y = fa[y][k];
    return fa[x][0];

}

inline void Init() {
    for(int i = 1;i <= n;++i) lg[i] = lg[i >> 1] + 1;
}
```

Kruskal 算法：

```cpp
int find(int x){
    if(fa[x] != x) fa[x] = find(fa[x]);
    return fa[x];
}

bool cmp(node a,node b){
    return a.w < b.w;
}

void Kruskal() {
    for(int i = 1;i <= n;i++)
        fa[i] = i;
    sort(a + 1,a + 1 + m,cmp);
    for(int i = 1;i <= m;++i){
        int fx = find(a[i].x);
        int fy = find(a[i].y);
        if(rand() & 1) swap(fx,fy);
        if(fx == fy) continue;
        ans += a[i].w;
        fa[fy] = fx;
        cnt++;
        if(cnt == n - 1) break;
    }
}
```

Prim 算法：

```
void Prim_Heap(int s) {
    memset(dis,0x3f,sizeof dis);
    memset(vis,false,sizeof vis);
    priority_queue <pair <int ,int >, vector < pair <int ,int > >, greater < pair <int ,int > > > Q;
    dis[s] = 0;
    Q.push(make_pair(0,s));
    while(!Q.empty()) {
        int u = Q.top().second;
        int d = Q.top().first;
        Q.pop();
        if(vis[u]) continue;
        num++;
        sum += d;
        vis[u] = true;
        for(int i = head[u];i;i = e[i].next) {
            int v = e[i].to;
            if(dis[v] > e[i].dis) {
                dis[v] = e[i].dis;
                Q.push(make_pair(dis[v],v));
            }
        }
    }
}
```

数据结构

一维树状数组：

```
void add(int x,int t) {
    while(x <= n) {
        v[x] += t;
        x += lowbit(x);
    }
}

int query(int x) {
    int res = 0;
    while(x) {
        res += v[x];
        x -= lowbit(x);
    }
    return res;
}
```

二维树状数组：

```
void add(int x,int y,int t) {
    while(x <= n) {
        for(int k = y;k <= m;k += lowbit(k))
            v[x][k] += t;
        x += lowbit(x);
    }
}

int query(int x,int y) {
    int res = 0;
    while(x) {
        for(int k = y;k;k -= lowbit(k))
            res += v[x][k];
        x -= lowbit(x);
    }
    return res;
}
```

线段树 1（单点修改，区间查询）：

```
void push_up(LL k) {
    tree[k] = tree[ls(k)] + tree[rs(k)];
}

inline void f(LL k,LL l,LL r,LL p) {
    tag[k] += p;
    tree[k] += p * (r - l + 1);
}

void push_down(LL k,LL l,LL r) {
    LL mid = (l + r) >> 1;
    f(ls(k),l,mid,tag[k]);
    f(rs(k),mid + 1,r,tag[k]);
    tag[k] = 0;
}
```

```
void build(LL k,LL l,LL r) {
    tag[k] = 0;
    if(l == r) {
        tree[k] = a[l];
        return ;
    }
    LL mid = (l + r) >> 1;
    build(ls(k),l,mid);
    build(rs(k),mid + 1,r);
    push_up(k);
}

void update(LL k,LL l,LL r,LL x,LL y,LL p) {
    if(y < l || x > r) return ;
    if(x <= l && y >= r) {
        tree[k] += p * (r - l + 1);
        tag[k] += p;
        return ;
    }
    push_down(k,l,r);
    LL mid = (l + r) >> 1;
    if(x <= mid) update(ls(k),l,mid,x,y,p);
    if(y > mid) update(rs(k),mid + 1,r,x,y,p);
    push_up(k);
}

```

```cpp
LL Query(LL k,LL l,LL r,LL x,LL y) {
    LL res = 0;
    if(y < l || x > r) return 0;
    if(x <= l && y >= r) return tree[k];
    LL mid = (l + r) >> 1;
    push_down(k,l,r);
    if(x <= mid) res += Query(ls(k),l,mid,x,y);
    if(y > mid) res += Query(rs(k),mid + 1,r,x,y);
    return res;
}
```

线段树 2（单点修改，区间查询）：

```cpp
void push_up(LL k) {
    tree[k] = tree[ls(k)] + tree[rs(k)];
    tree[k] %= MOD;
}

inline void f(LL k,LL l,LL r,LL p) {
    tree[k] += p * (r - l + 1);
    tree[k] %= MOD;
    tag[k] += p;
    tag[k] %= MOD;
}

void f_2(LL k,LL l,LL r,LL p) {
    tree[k] *= p;
    tree[k] %= MOD;
    tag[k] *= p;
    tag[k] %= MOD;
    tag2[k] *= p;
    tag2[k] %= MOD;
}
```

```
void push_down(LL k,LL l,LL r) {
    LL mid = (l + r) >> 1;
    if(tag2[k] != 1) {
        f_2(ls(k),l,mid,tag2[k]);
        f_2(rs(k),mid + 1,r,tag2[k]);
        tag2[k] = 1;
    }
    if(tag[k]) {
        f(ls(k),l,mid,tag[k]);
        f(rs(k),mid + 1,r,tag[k]);
        tag[k] = 0;
    }
}

void build(LL k,LL l,LL r) {
    tag[k] = 0;
    tag2[k] = 1;
    if(l == r) {
        tree[k] = a[l];
        return ;
    }
    LL mid = (l + r) >> 1;
    build(ls(k),l,mid);
    build(rs(k),mid + 1,r);
    push_up(k);
}
```

```
void Update_Add(LL k,LL l,LL r,LL x,LL y,LL p) {
    if(y < l or x > r) return ;
    if(l >= x and r <= y) {
        tag[k] += p;
        tag[k] %= MOD;
        tree[k] += p * (r - l + 1);
        tree[k] %= MOD;
        return ;
    }
    push_down(k,l,r);
    LL mid = (l + r) >> 1;
    if(x <= mid) Update_Add(ls(k),l,mid,x,y,p);
    if(y > mid) Update_Add(rs(k),mid + 1,r,x,y,p);
    push_up(k);
}

LL Query(LL k,LL l,LL r,LL x,LL y) {
    if(y < l or x > r) return 0;
    if(l >= x and r <= y) return tree[k];
    push_down(k,l,r);
    LL mid = (l + r) >> 1;
    LL res = 0;
    if(x <= mid) res = (res + Query(ls(k),l,mid,x,y)) % MOD;
    if(y > mid) res = (res + Query(rs(k),mid + 1,r,x,y)) % MOD;
    return res % MOD;
}
```

```
void Update_Cheng(LL k,LL l,LL r,LL x,LL y,LL p) {
    if(y < l or x > r) return ;
    if(l >= x and r <= y) {
        tag[k] *= p;
        tag[k] %= MOD;
        tree[k] *= p;
        tree[k] %= MOD;
        tag2[k] *= p;
        tag2[k] %= MOD;
        return ;
    }
    push_down(k,l,r);
    LL mid = (l + r) >> 1;
    if(x <= mid) Update_Cheng(ls(k),l,mid,x,y,p);
    if(y > mid) Update_Cheng(rs(k),mid + 1,r,x,y,p);
    push_up(k);
}
```

ST 表：

```
void Init() {
    lg[0] = -1;
    for(int i = 1;i <= n;++i) lg[i] = lg[i >> 1] + 1;
    for(int i = 1;i <= n;++i) scanf("%d",&Max[i][0]);
    for(int i = 1;i <= n;++i) Min[i][0] = Max[i][0];
    for(int j = 1;j <= 21;++j)
        for(int i = 1;i + (1 << j) - 1 <= n;++i) {
            Max[i][j] = max(Max[i][j - 1],Max[i + (1 << (j - 1))][j - 1]);
            Min[i][j] = min(Min[i][j - 1],Min[i + (1 << (j - 1))][j - 1]);
        }
}

int QueryMax(int l,int r) {
    int k = lg[r - l + 1];
    return max(Max[l][k],Max[r - (1 << k) + 1][k]);
}

int QueryMin(int l,int r) {
    int k = lg[r - l + 1];
    return min(Min[l][k],Min[r - (1 << k) + 1][k]);
}
```

并查集：

```
void Init() {
    for(int i = 1;i <= n;++i) father[i] = i;
}

int Find(int x) {
    if(x != father[x]) father[x] = Find(father[x]);
    return father[x];
}
```

字符串算法：

Manacher 算法：

```c
int Init() {  //初始化并返回new_s长度
    int len = strlen(s);
    new_s[0] = '$';
    new_s[1] = '#';
    int j = 2;
    for(int i = 0; i < len;++i) {
        new_s[j++] = s[i];
        new_s[j++] = '#';
    }
    new_s[j] = '\0';
    return j;
}

int Manacher() {
    int len = Init();
    int max_len = -1;
    int id = 0,mx = 0;
    for(int i = 1;i < len;++i) {
        if(i < mx) p[i] = min(p[2 * id - i],mx - i); // 2 * id - i指i关于id的对称点j
        else p[i] = 1;
        while(new_s[i - p[i]] == new_s[i + p[i]]) p[i]++;
        if(mx < i + p[i]) {
            id = i;
            mx = i + p[i];
        }
        max_len = max(max_len,p[i] - 1);
    }
    return max_len;
}
```

KMP 算法：

```cpp
void kmp1() {
    int j = 0;
    for(int i = 2;i <= lenb;++i) {
        while(j && b[j + 1] != b[i]) j = next[j];
        if(b[j + 1] == b[i]) j++;
        next[i] = j;
    }
}

void kmp2() {
    int j = 0;
    for(int i = 1;i <= lena;++i) {
        while(j && b[j + 1] != a[i]) j = next[j];
        if(b[j + 1] == a[i]) j++;
        if(j == lenb) {
            cout << i - lenb + 1 <<endl;
            j = next[j];
        }
    }
}
```

Hash：

```cpp
const int Base = 131;
const int Prime = 19260817;
const LL MOD = 212370440130137957ll;
LL hash(char s[]) {
    int len = strlen(s);
    LL cnt = 0;
    for(int i = 0;i < len;++i)
        cnt = (cnt * Base + (LL)s[i]) % MOD + Prime;
    return cnt;
}
```