

高精度运算

高精度加法：

```
11 int main() {
12     scanf("%s%s",&a1,&b1);
13     if(a1[0] == '0' && b1[0] == '0') {
14         cout << "0";
15         return 0;
16     }
17     for(int i = 0; i < strlen(a1); ++i)
18         a[strlen(a1) - i - 1] = a1[i] - '0';
19     for(int i = 0; i < strlen(b1); ++i)
20         b[strlen(b1) - i - 1] = b1[i] - '0';
21     m = max(strlen(a1), strlen(b1));
22     for(int i = 0; i < m; ++i)
23         c[i] = a[i] + b[i];
24     for (int i = 0; i <= m; ++i) {
25         c[i + 1] = c[i + 1] + c[i] / 10;
26         c[i] = c[i] % 10;
27     }
28     m++;
29     while(!c[m]) m--;
30     for(int i = m; i >= 0; --i)
31         cout << c[i];
32     return 0;
33 }
```

高精度减法:

```
13 int main() {
14     scanf("%s%s",&x,&y);
15     int lena = strlen(x),lenb = strlen(y);
16     for(int i = 0;i < lena;++i)
17         a[lena - i - 1] = x[i] - '0';
18     for(int i = 0;i < lenb;++i)
19         b[lenb - i - 1]=y[i] - '0';
20     if(lena > lenb) vis = 1;
21     if(lena < lenb) vis = 0;
22     if(lena == lenb) {
23         string temp1 = x,temp2 = y;
24         if(temp1 > temp2) vis = 1;
25         if(temp1 == temp2) vis = 1;
26         else vis = 0;
27     }
28     m = max(lena,lenb);
29     if(vis) {
30         for(int i = 0;i < m;++i)
31             c[i]=a[i]-b[i];
32     }
33     else {
34         for(int i = 0;i < m;++i)
35             c[i]=b[i]-a[i];
36     }
37     for(int i = 0;i <= m;++i) {
38         if(c[i] < 0){
39             c[i] += 10;
40             c[i + 1]--;
41         }
42     }
43     temp = m;
44     for(int i = m;i >= 0;--i) {
45         if(c[i]) break;
46         temp--;
47         if(i == 0 && c[i] == 0) {
48             temp = 0;
49             break;
50         }
51     }
52     if(!vis) cout << "-";
53     for(int i = temp;i >= 0;--i)
54         cout << c[i];
55     return 0;
56 }
```

高精度乘法：

```
10 int main() {
11     scanf("%s%s",&x,&y);
12     int lena = strlen(x),lenb = strlen(y);
13     for(int i = 0;i < lena;++i)
14         a[lena - i - 1] = x[i] - '0';
15     for(int i = 0;i < lenb;++i)
16         b[lenb - i - 1] = y[i] - '0';
17     for(int i = 0;i < lena;++i)
18         for(int j = 0;j < lenb;++j) {
19             c[i + j] += a[i] * b[j];
20             c[i + j + 1] += c[i+j] / 10;
21             c[i + j] %= 10;
22         }
23     int lenc = lena + lenb;
24     while(lenc > 1 && c[lenc - 1] == 0) lenc--;
25     lenc--;
26     for(int i = lenc;i >= 0;--i)
27         cout << c[i];
28     return 0;
29 }
30
```

数学

快速幂：

```
9 LL power(LL a, LL b) {
10     LL t = 1, y = a;
11     while(b) {
12         if (b & 1) t = t * y % k;
13         y = y * y % k;
14         b >>= 1;
15     }
16     return t;
17 }
```

埃式素数判定：

```
7 bool is_prime(int x) {
8     if(x == 1 || x == 0) return false;
9     for(int i = 2; i * i <= x; ++i)
10         if(x % i == 0) return false;
11     return true;
12 }
```

欧拉筛素数：

```
11 void is_prime(int list) {
12     memset(vis, true, sizeof vis);
13     vis[0] = vis[1] = false;
14     for(int i = 2; i <= list; ++i){
15         if(vis[i]) prime[++tot] = i;
16         for(int j = 1; j <= list && i * prime[j] <= list; ++j){
17             vis[i * prime[j]] = false;
18             if(i % prime[j] == 0) break;
19         }
20     }
21 }
```

欧几里得算法：

```
8 □ int gcd(int a,int b) {  
9   if(!b) return a;  
10  else return gcd(b,a % b);  
11 }
```

拓展欧几里得：

```
24 □ void ex_gcd(int &x,int &y,int a,int b) {  
25   if(b == 0) {  
26     y = 0;  
27     x = 1;  
28   }  
29   else {  
30     ex_gcd(y,x,b,a % b);  
31     y -= x * (a / b);  
32   }  
33 }
```

欧拉函数：

```
38 □ int euler(int n) {  
39   int res = n;  
40   for(int i = 2;i * i <= n;++i) {  
41     if(n % i == 0) res = res / i * (i - 1);  
42     while(n % i == 0) n /= i;  
43   }  
44   if(n > 1) res = res / n * (n - 1);  
45   return res;  
46 }
```

线性筛欧拉函数：

```
11 void get_phi(int list) {
12     memset(vis,true,sizeof vis);
13     vis[0] = vis[1] = false;
14     phi[1] = 1;
15     for(int i = 2;i <= list;++i){
16         if(vis[i]) prime[++tot] = i,phi[i] = i - 1;
17         for(int j = 1;j <= list && i * prime[j] <= list;++j){
18             vis[i * prime[j]] = false;
19             if(i % prime[j]) phi[i * prime[j]] = phi[i] * (prime[j] - 1);
20             else {
21                 phi[i * prime[j]] = phi[i] * prime[j];
22                 break;
23             }
24         }
25         phi[i] += phi[i - 1];
26     }
27 }
```

矩阵运算：

```
23 inline Mat Mul(Mat a,Mat b){
24     Mat tmp;
25     tmp.clear();
26     for(int i = 1;i <= n;++i)
27         for(int j = 1;j <= n;++j)
28             for(int k = 1;k <= n;++k)
29                 tmp.a[i][j] = (tmp.a[i][j] + (a.a[i][k] * b.a[k][j]) % MOD) % MOD;
30     return tmp;
31 }
32 inline Mat power(Mat a,long long b){
33     Mat ans;
34     ans.clear();
35     for(int i = 1;i <= n;++i)
36         ans.a[i][i]=1;
37     while(b){
38         if (b & 1) ans = Mul(ans,a);
39         a = Mul(a,a);
40         b >>= 1;
41     }
42     return ans;
43 }
44
45 inline Mat Add(Mat T_1,Mat T_2) {
46     Mat Tmp;
47     Tmp.clear();
48     for(int i = 1;i <= n;++i)
49         for(int j = 1;j <= n;++j)
50             Tmp.a[i][j] = T_1.a[i][j] + T_2.a[i][j],Tmp.a[i][j] %= MOD;
51     return Tmp;
52 }
```

SPFA 算法:

```

27 void Spfa() {
28     queue <int> Q;
29     Q.push(s);
30     vis[s] = true;
31     memset(dis, 0x3f, sizeof dis);
32     dis[s] = 0;
33     while(!Q.empty()) {
34         int u = Q.front();
35         Q.pop();
36         vis[u] = false;
37         for(int i = head[u]; i; i = e[i].nxt) {
38             int v = e[i].to;
39             if(dis[v] > dis[u] + e[i].dis) {
40                 dis[v] = dis[u] + e[i].dis;
41                 if(!vis[v]) {
42                     Q.push(v);
43                     vis[v] = true;
44                 }
45             }
46         }
47     }
48 }

```

Dijkstra 算法:

```

26 void Dijkstra() {
27     memset(dis, 0x3f, sizeof dis);
28     dis[s] = 0;
29     priority_queue <pair <int, int>, vector <pair <int, int>>, greater <pair <int, int>>> Q;
30     Q.push(make_pair(0, s));
31     while(!Q.empty()) {
32         int u = Q.top().second;
33         Q.pop();
34         if(vis[u]) continue;
35         vis[u] = true;
36         for(int i = head[u]; i; i = e[i].nxt) {
37             int v = e[i].to;
38             if(dis[v] > dis[u] + e[i].dis) {
39                 dis[v] = dis[u] + e[i].dis;
40                 if(!vis[v]) Q.push(make_pair(dis[v], v));
41             }
42         }
43     }
44 }

```

最短路计数:

```
28 void Dijkstra_Heap() {
29     priority_queue <pair <int ,int > ,vector <pair <int ,int > >,greater <pair <int ,int > > > Q;
30     memset(dis,0x3f,sizeof dis);
31     dis[s] = 0;
32     ans[1] = 1;
33     Q.push(make_pair(0,s));
34     while(!Q.empty()) {
35         int u = Q.top().second;
36         Q.pop();
37         if(vis[u]) continue;
38         vis[u] = true;
39         for(int i = head[u];i;i = e[i].nxt) {
40             int v = e[i].to;
41             if(dis[v] > dis[u] + 1) {
42                 dis[v] = dis[u] + 1;
43                 ans[v] = ans[u];
44                 if(!vis[v]) Q.push(make_pair(dis[v],v));
45             }
46             else if(dis[v] == dis[u] + 1) {
47                 ans[v] += ans[u];
48                 ans[v] %= MOD;
49             }
50         }
51     }
52 }
```

负环:

```
50     memset(dis,0x3f,sizeof dis);
51     dis[1] = 0;
52     Q.push(1);
53     vis[1] = true;
54     while(!Q.empty()) {
55         int u = Q.front();
56         Q.pop();
57         vis[u] = false;
58         for(int i = head[u];i;i = e[i].nxt) {
59             int v = e[i].to;
60             if(dis[v] > dis[u] + e[i].dis) {
61                 dis[v] = dis[u] + e[i].dis;
62                 num[v] = num[u] + 1;
63                 if(num[v] >= n) return true;
64                 if(!vis[v]) {
65                     Q.push(v);
66                     vis[v] = true;
67                 }
68             }
69         }
70     }
71     return false;
72 }
```


Tarjan 割点:

```
27 void Tarjan(int u,int fa) {
28     dfn[u] = low[u] = ++cmt;
29     int child = 0;
30     for(int i = head[u];i;i = e[i].nxt) {
31         int v = e[i].to;
32         if(!dfn[v]) {
33             Tarjan(v,fa);
34             low[u] = min(low[u],low[v]);
35             if(low[v] >= dfn[u] && u != fa) cut[u] = true;
36             if(u == fa) child++;
37         }
38         else low[u] = min(low[u],dfn[v]);
39     }
40     if(u == fa && child >= 2) cut[fa]=true;
41 }
```

Tarjan 强连通分量 (缩点):

匈牙利算法:

```
24 inline bool dfs(int u) {
25     for(int i = head[u];i;i = e[i].next) {
26         int v = e[i].to;
27         if(!used[v]) {
28             used[v] = true;
29             if(!Matched[v] || dfs(Matched[v])) {
30                 Matched[v] = u;
31                 return true;
32             }
33         }
34     }
35     return false;
36 }
```

倍增 LCA:

```
29 inline void dfs(int now,int f) {
30     depth[now] = depth[f] + 1;
31     fa[now][0] = f;
32     for(int i = 1;(1 << i) <= depth[now];++i) fa[now][i] = fa[fa[now][i - 1]][i - 1];
33     for(int i = head[now];i;i = e[i].next) {
34         int v = e[i].to;
35         if(v != f) dfs(v,now);
36     }
37 }
38
39 inline int get_lca(int x,int y) {
40     if(depth[x] < depth[y]) swap(x,y);
41     while(depth[x] > depth[y]) x = fa[x][lg[depth[x] - depth[y] - 1]];
42     if(x == y) return x;
43     for(int k = lg[depth[x]];k >= 0;--k) if(fa[x][k] != fa[y][k]) x = fa[x][k],y = fa[y][k];
44     return fa[x][0];
45 }
46
47
48 inline void Init() {
49     for(int i = 1;i <= n;++i) lg[i] = lg[i >> 1] + 1;
50 }
```

Kruskal 算法:

```
15 int find(int x){
16     if(fa[x] != x) fa[x] = find(fa[x]);
17     return fa[x];
18 }
19
20 bool cmp(node a,node b){
21     return a.w < b.w;
22 }
23
24 void Kruskal() {
25     for(int i = 1;i <= n;i++)
26         fa[i] = i;
27     sort(a + 1,a + 1 + m,cmp);
28     for(int i = 1;i <= m;++i){
29         int fx = find(a[i].x);
30         int fy = find(a[i].y);
31         if(rand() & 1) swap(fx,fy);
32         if(fx == fy) continue;
33         ans += a[i].w;
34         fa[fy] = fx;
35         cnt++;
36         if(cnt == n - 1) break;
37     }
38 }
```

Prim 算法:

```
31 void Prim_Heap(int s) {
32     memset(dis, 0x3f, sizeof dis);
33     memset(vis, false, sizeof vis);
34     priority_queue <pair <int ,int > , vector < pair <int ,int > > , greater < pair <int ,int > > > Q;
35     dis[s] = 0;
36     Q.push(make_pair(0,s));
37     while(!Q.empty()) {
38         int u = Q.top().second;
39         int d = Q.top().first;
40         Q.pop();
41         if(vis[u]) continue;
42         num++;
43         sum += d;
44         vis[u] = true;
45         for(int i = head[u]; i = e[i].next) {
46             int v = e[i].to;
47             if(dis[v] > e[i].dis) {
48                 dis[v] = e[i].dis;
49                 Q.push(make_pair(dis[v],v));
50             }
51         }
52     }
53 }
```

数据结构

一维树状数组：

```
1 void add(int x,int t) {
2     while(x <= n) {
3         v[x] += t;
4         x += lowbit(x);
5     }
6 }
7
8 int query(int x) {
9     int res = 0;
10    while(x) {
11        res += v[x];
12        x -= lowbit(x);
13    }
14    return res;
15 }
```

二维树状数组：

```
17 void add(int x,int y,int t) {
18     while(x <= n) {
19         for(int k = y;k <= m;k += lowbit(k))
20             v[x][k] += t;
21         x += lowbit(x);
22     }
23 }
24
25 int query(int x,int y) {
26     int res = 0;
27     while(x) {
28         for(int k = y;k >= 1;k -= lowbit(k))
29             res += v[x][k];
30         x -= lowbit(x);
31     }
32     return res;
33 }
```

线段树 1 (单点修改, 区间查询):

```
19 void push_up(LL k) {
20     tree[k] = tree[ls(k)] + tree[rs(k)];
21 }
22
23 inline void f(LL k, LL l, LL r, LL p) {
24     tag[k] += p;
25     tree[k] += p * (r - l + 1);
26 }
27
28 void push_down(LL k, LL l, LL r) {
29     LL mid = (l + r) >> 1;
30     f(ls(k), l, mid, tag[k]);
31     f(rs(k), mid + 1, r, tag[k]);
32     tag[k] = 0;
33 }
34
35 void build(LL k, LL l, LL r) {
36     tag[k] = 0;
37     if(l == r) {
38         tree[k] = a[l];
39         return ;
40     }
41     LL mid = (l + r) >> 1;
42     build(ls(k), l, mid);
43     build(rs(k), mid + 1, r);
44     push_up(k);
45 }
46
47 void update(LL k, LL l, LL r, LL x, LL y, LL p) {
48     if(y < l || x > r) return ;
49     if(x <= l && y >= r) {
50         tree[k] += p * (r - l + 1);
51         tag[k] += p;
52         return ;
53     }
54     push_down(k, l, r);
55     LL mid = (l + r) >> 1;
56     if(x <= mid) update(ls(k), l, mid, x, y, p);
57     if(y > mid) update(rs(k), mid + 1, r, x, y, p);
58     push_up(k);
59 }
60
```

```

61 LL Query(LL k, LL l, LL r, LL x, LL y) {
62     LL res = 0;
63     if(y < l || x > r) return 0;
64     if(x <= l && y >= r) return tree[k];
65     LL mid = (l + r) >> 1;
66     push_down(k, l, r);
67     if(x <= mid) res += Query(ls(k), l, mid, x, y);
68     if(y > mid) res += Query(rs(k), mid + 1, r, x, y);
69     return res;
70 }

```

线段树 2 (单点修改, 区间查询):

```

15 void push_up(LL k) {
16     tree[k] = tree[ls(k)] + tree[rs(k)];
17     tree[k] %= MOD;
18 }
19
20 inline void f(LL k, LL l, LL r, LL p) {
21     tree[k] += p * (r - l + 1);
22     tree[k] %= MOD;
23     tag[k] += p;
24     tag[k] %= MOD;
25 }
26
27 void f_2(LL k, LL l, LL r, LL p) {
28     tree[k] *= p;
29     tree[k] %= MOD;
30     tag[k] *= p;
31     tag[k] %= MOD;
32     tag2[k] *= p;
33     tag2[k] %= MOD;
34 }

```

```

36 void push_down(LL k,LL l,LL r) {
37     LL mid = (l + r) >> 1;
38     if(tag2[k] != 1) {
39         f_2(ls(k),l,mid,tag2[k]);
40         f_2(rs(k),mid + 1,r,tag2[k]);
41         tag2[k] = 1;
42     }
43     if(tag[k]) {
44         f(ls(k),l,mid,tag[k]);
45         f(rs(k),mid + 1,r,tag[k]);
46         tag[k] = 0;
47     }
48 }
49
50 void build(LL k,LL l,LL r) {
51     tag[k] = 0;
52     tag2[k] = 1;
53     if(l == r) {
54         tree[k] = a[l];
55         return ;
56     }
57     LL mid = (l + r) >> 1;
58     build(ls(k),l,mid);
59     build(rs(k),mid + 1,r);
60     push_up(k);
61 }

63 void Update_Add(LL k,LL l,LL r,LL x,LL y,LL p) {
64     if(y < l or x > r) return ;
65     if(l >= x and r <= y) {
66         tag[k] += p;
67         tag[k] %= MOD;
68         tree[k] += p * (r - l + 1);
69         tree[k] %= MOD;
70         return ;
71     }
72     push_down(k,l,r);
73     LL mid = (l + r) >> 1;
74     if(x <= mid) Update_Add(ls(k),l,mid,x,y,p);
75     if(y > mid) Update_Add(rs(k),mid + 1,r,x,y,p);
76     push_up(k);
77 }

78
79 LL Query(LL k,LL l,LL r,LL x,LL y) {
80     if(y < l or x > r) return 0;
81     if(l >= x and r <= y) return tree[k];
82     push_down(k,l,r);
83     LL mid = (l + r) >> 1;
84     LL res = 0;
85     if(x <= mid) res = (res + Query(ls(k),l,mid,x,y)) % MOD;
86     if(y > mid) res = (res + Query(rs(k),mid + 1,r,x,y)) % MOD;
87     return res % MOD;
88 }

```

```

90 void Update_Cheng(LL k,LL l,LL r,LL x,LL y,LL p) {
91     if(y < l or x > r) return ;
92     if(l >= x and r <= y) {
93         tag[k] *= p;
94         tag[k] %= MOD;
95         tree[k] *= p;
96         tree[k] %= MOD;
97         tag2[k] *= p;
98         tag2[k] %= MOD;
99         return ;
100     }
101     push_down(k,l,r);
102     LL mid = (l + r) >> 1;
103     if(x <= mid) Update_Cheng(ls(k),l,mid,x,y,p);
104     if(y > mid) Update_Cheng(rs(k),mid + 1,r,x,y,p);
105     push_up(k);
106 }

```

ST 表:

```

18 void Init() {
19     lg[0] = -1;
20     for(int i = 1;i <= n;++i) lg[i] = lg[i >> 1] + 1;
21     for(int i = 1;i <= n;++i) scanf("%d",&Max[i][0]);
22     for(int i = 1;i <= n;++i) Min[i][0] = Max[i][0];
23     for(int j = 1;j <= 21;++j)
24         for(int i = 1;i + (1 << j) - 1 <= n;++i) {
25             Max[i][j] = max(Max[i][j - 1],Max[i + (1 << (j - 1))][j - 1]);
26             Min[i][j] = min(Min[i][j - 1],Min[i + (1 << (j - 1))][j - 1]);
27         }
28 }
29
30 int QueryMax(int l,int r) {
31     int k = lg[r - l + 1];
32     return max(Max[l][k],Max[r - (1 << k) + 1][k]);
33 }
34
35 int QueryMin(int l,int r) {
36     int k = lg[r - l + 1];
37     return min(Min[l][k],Min[r - (1 << k) + 1][k]);
38 }

```

并查集:

```

7 void Init() {
8     for(int i = 1;i <= n;++i) father[i] = i;
9 }
10
11 int Find(int x) {
12     if(x != father[x]) father[x] = Find(father[x]);
13     return father[x];
14 }

```


字符串算法：

Manacher 算法：

```
16 int Init() { //初始化并返回new_s长度
17     int len = strlen(s);
18     new_s[0] = '$';
19     new_s[1] = '#';
20     int j = 2;
21     for(int i = 0; i < len; ++i) {
22         new_s[j++] = s[i];
23         new_s[j++] = '#';
24     }
25     new_s[j] = '\0';
26     return j;
27 }
28
29 int Manacher() {
30     int len = Init();
31     int max_len = -1;
32     int id = 0, mx = 0;
33     for(int i = 1; i < len; ++i) {
34         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
35         else p[i] = 1;
36         while(new_s[i - p[i]] == new_s[i + p[i]]) p[i]++;
37         if(mx < i + p[i]) {
38             id = i;
39             mx = i + p[i];
40         }
41         max_len = max(max_len, p[i] - 1);
42     }
43     return max_len;
44 }
```

KMP 算法:

```
17 void kmp1() {
18     int j = 0;
19     for(int i = 2; i <= lenb; ++i) {
20         while(j && b[j + 1] != b[i]) j = next[j];
21         if(b[j + 1] == b[i]) j++;
22         next[i] = j;
23     }
24 }
25
26 void kmp2() {
27     int j = 0;
28     for(int i = 1; i <= lena; ++i) {
29         while(j && b[j + 1] != a[i]) j = next[j];
30         if(b[j + 1] == a[i]) j++;
31         if(j == lenb) {
32             cout << i - lenb + 1 << endl;
33             j = next[j];
34         }
35     }
36 }
```

Hash:

```
1  const int Base = 131;
2  const int Prime = 19260817;
3  const LL MOD = 21237044013013795711;
4  LL hash(char s[]) {
5      int len = strlen(s);
6      LL cnt = 0;
7      for(int i = 0; i < len; ++i)
8          cnt = (cnt * Base + (LL)s[i]) % MOD + Prime;
9      return cnt;
10 }
```