

# ICPC-ACM Template

Ruoyisius

2023 年 9 月 22 日

## 目录

<b>一、 图论</b>	<b>3</b>
1.1 增广路	3
1.2 SPFA	4
1.3 Tarjan	5
1.4 基环树	5
<b>二、 网络流</b>	<b>7</b>
2.1 Dinic	7
<b>三、 数据结构</b>	<b>9</b>
3.1 并查集	9
3.2 树状数组	9
3.3 线段树	10
3.4 懒标记线段树	13
3.5 倍增 LCA-最近公共祖先	14
3.6 珂朵莉树	14
3.7 ST	15
3.8 支配树	16
<b>四、 计算几何</b>	<b>19</b>
<b>五、 字符串</b>	<b>20</b>
5.1 AC 自动机	20
5.2 KMP& EXKMP	21
5.3 Manacher	22
5.4 后缀自动机	23
5.5 序列自动机	25
<b>六、 数学</b>	<b>26</b>
6.1 欧拉函数	26
6.2 扩展 GCD	26
6.3 各种筛	26
6.4 多项式	28
6.5 斯特林数	37
6.6 高斯消元	38
<b>七、 其他</b>	<b>39</b>
7.1 快读快写	39

## 一、 图论

## 1.1 增广路

```
1 struct augment_path {
2     vector<vector<int>> > g;
3     vector<int> pa; // 匹配
4     vector<int> pb;
5     vector<int> vis; // 访问
6     int n, m; // 两个点集中的顶点数量
7     int dfn; // 时间戳记
8     int res; // 匹配数
9
10    augment_path(int _n, int _m) : n(_n), m(_m) {
11        assert(0 <= n && 0 <= m);
12        pa = vector<int>(n, -1);
13        pb = vector<int>(m, -1);
14        vis = vector<int>(n);
15        g.resize(n);
16        res = 0;
17        dfn = 0;
18    }
19
20    void add(int from, int to) {
21        assert(0 <= from && from < n && 0 <= to && to < m);
22        g[from].push_back(to);
23    }
24
25    bool dfs(int v) {
26        vis[v] = dfn;
27        for (int u : g[v]) {
28            if (pb[u] == -1) {
29                pb[u] = v;
30                pa[v] = u;
31                return true;
32            }
33        }
34        for (int u : g[v]) {
35            if (vis[pb[u]] != dfn && dfs(pb[u])) {
36                pa[v] = u;
37                pb[u] = v;
38                return true;
39            }
40        }
41        return false;
42    }
43
44    int solve() {
45        while (true) {
46            dfn++;
47            int cnt = 0;
48            for (int i = 0; i < n; i++) {
```

```

48         if (pa[i] == -1 && dfs(i)) {
49             cnt++;
50         }
51     }
52     if (cnt == 0){
53         break;
54     }
55     res += cnt;
56 }
57 return res;
58 }
59 };

```

## 1.2 SPFA

```

1  /*
2   * Args:
3   *   g[]: graph, (u, v, w) = (u, g[u][i].first, g[u][i].second)
4   *   st: source vertex
5   * Return:
6   *   dis[]: distance from source vertex to each other vertex
7   */
8  vector<pair<int, int> > g[N];
9  int dis[N], vis[N];
10 void spfa(int st)
11 {
12     memset(dis, -1, sizeof(dis));
13     memset(vis, 0, sizeof(vis));
14     queue<int> q;
15     q.push(st);
16     dis[st] = 0;
17     vis[st] = true;
18     while (!q.empty()) {
19         int u = q.front();
20         q.pop();
21         vis[u] = false;
22         for (auto x : g[u]) {
23             int v = x.first, w = x.second;
24             if (dis[v] == -1 || dis[u] + w < dis[v]) {
25                 dis[v] = dis[u] + w;
26                 if (!vis[v]) {
27                     vis[v] = true;
28                     q.push(v);
29                 }
30             }
31         }
32     }
33 }

```

### 1.3 Tarjan

```
1  vector<int> dfn(n, -1), low(n, -1), be(n, -1);
2  int tot = 0, cnt = 0;
3  vector<int> st;
4  function<void(int)> tarjan = [&](int cur) {
5      dfn[cur] = low[cur] = tot++;
6      st.push_back(cur);
7      for (auto &nex : g[cur]) {
8          if (dfn[nex] == -1) {
9              tarjan(nex);
10             low[cur] = min(low[cur], low[nex]);
11         } else if (be[nex] == -1) {
12             low[cur] = min(low[cur], dfn[nex]);
13         }
14     }
15     if (dfn[cur] == low[cur]) {
16         int v;
17         do {
18             v = st.back();
19             st.pop_back();
20             be[v] = cnt;
21         } while (v != cur);
22         cnt++;
23     }
24 };
25
26 for (int i = 0; i < n; i++) {
27     if (dfn[i] == -1) {
28         tarjan(i);
29     }
30 }
```

### 1.4 基环树

```
1  vector<vector<int>> E(n);
2  vector<int> deg(n);
3  for (int j = 0; j < n; j++) {
4      deg[j] = E[j].size();
5  }
6  queue<int> Q;
7  for (int j = 0; j < n; j++) {
8      if (deg[j] == 1) {
9          Q.push(j);
10     }
11 }
12 vector<bool> used(n, false);
13 while (!Q.empty()) {
```

```
14     int v = Q.front();
15     Q.pop();
16     used[v] = true;
17     for (int w : E[v]) {
18         deg[w]--;
19         if (deg[w] == 1) {
20             Q.push(w);
21         }
22     }
23 }
24 vector<int> p(n, -1);
25 vector<int> d(n, -1);
26 for (int j = 0; j < n; j++) {
27     if (!used[j]) {
28         d[j] = 0;
29         Q.push(j);
30     }
31 }
32 while (!Q.empty()) {
33     int v = Q.front();
34     Q.pop();
35     for (int w : E[v]) {
36         if (d[w] == -1) {
37             d[w] = d[v] + 1;
38             p[w] = v;
39             Q.push(w);
40         }
41     }
42 }
```

## 二、 网络流

## 2.1 Dinic

```

1  struct Edge {
2      int from, to, cap, flow;
3  };
4  struct Dinic {
5      int s, t;
6      bool vis[N];
7      int d[N];
8      int cur[N];
9
10     vector<Edge> edges;
11     vector<int> G[N];
12     void AddEdge(int from, int to, int cap) {
13         edges.push_back({from, to, cap, 0});
14         edges.push_back({to, from, 0, 0});
15         //edges.push_back({to, from, cap, 0}); 如果是无向图.
16         G[from].push_back(edges.size() - 2);
17         G[to].push_back(edges.size() - 1);
18     }
19
20     bool BFS() {
21         memset(vis, 0, sizeof(vis));
22         queue<int> Q;
23         Q.push(s);
24         d[s] = 0;
25         vis[s] = 1;
26         while (!Q.empty()) {
27             int u = Q.front(); Q.pop();
28             for (int i = 0; i < G[u].size(); i++) {
29                 Edge& e = edges[G[u][i]];
30                 int v = e.to;
31                 if (!vis[v] && e.cap > e.flow) {
32                     vis[v] = 1;
33                     d[v] = d[u] + 1;
34                     Q.push(v);
35                 }
36             }
37         }
38         return vis[t];
39     }
40
41     int DFS(int u, int a) {
42         if (u == t || a == 0) return a;
43         int flow = 0, f;
44         for (int& i = cur[u]; i < G[u].size(); i++) { //这里取引用, 使得u的当前弧
            //被i改变, 再次访问到u时, 将跳过u已经访问过的支路
45             Edge& e = edges[G[u][i]], ee = edges[G[u][i] ^ 1];
46             int v = e.to;

```

```
47         if (d[v] == d[u] + 1 && (f = DFS(v, min(a, e.cap - e.flow))) > 0) {
48             e.flow += f;
49             ee.flow -= f;
50             flow += f;
51             a -= f;
52             if (a == 0) break;
53         }
54     }
55     return flow;
56 }
57
58 int Maxflow(int s, int t) {
59     this->s = s; this->t = t;
60     int flow = 0;
61     while (BFS()) {
62         memset(cur, 0, sizeof(cur));
63         flow += DFS(s, INF);
64     }
65     return flow;
66 }
67 };
```



## 三、 数据结构

### 3.1 并查集

```
1 struct DSU {
2     std::vector<int> f, siz;
3
4     DSU() {}
5     DSU(int n) {
6         init(n);
7     }
8
9     void init(int n) {
10         f.resize(n);
11         std::iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x) {
16         while (x != f[x]) {
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y) {
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y) {
27         x = find(x);
28         y = find(y);
29         if (x == y) {
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36
37     int size(int x) {
38         return siz[find(x)];
39     }
40 };
```

### 3.2 树状数组

```
1 template <typename T>
```

```
2 struct Fenwick {
3     int n;
4     std::vector<T> a;
5
6     Fenwick(int n = 0) {
7         init(n);
8     }
9
10    void init(int n) {
11        this->n = n;
12        a.assign(n, T());
13    }
14
15    void add(int x, T v) {
16        for (int i = x + 1; i <= n; i += i & -i) {
17            a[i - 1] += v;
18        }
19    }
20
21    T sum(int x) {
22        auto ans = T();
23        for (int i = x; i > 0; i -= i & -i) {
24            ans += a[i - 1];
25        }
26        return ans;
27    }
28
29    T rangeSum(int l, int r) {
30        return sum(r) - sum(l);
31    }
32
33    int kth(T k) {
34        int x = 0;
35        for (int i = 1 << std::lg(n); i; i /= 2) {
36            if (x + i <= n && k >= a[x + i - 1]) {
37                x += i;
38                k -= a[x - 1];
39            }
40        }
41        return x;
42    }
43 };
```

### 3.3 线段树

```
1 #include<bits/stdc++.h>
2 using i64=long long;
3 template<class Info>
4 struct SegmentTree {
```

```

5      int n;
6      std::vector<Info> info;
7      SegmentTree() : n(0) {}
8      SegmentTree(int n_, Info v_ = Info()) {
9          init(n_, v_);
10     }
11     template<class T>
12     SegmentTree(std::vector<T> init_) {
13         init(init_);
14     }
15     void init(int n_, Info v_ = Info()) {
16         init(std::vector(n_, v_));
17     }
18     template<class T>
19     void init(std::vector<T> init_) {
20         n = init_.size();
21         info.assign(4 << std::__lg(n), Info());
22         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
23             if (r - l == 1) {
24                 info[p] = init_[l];
25                 return;
26             }
27             int m = (l + r) / 2;
28             build(2 * p, l, m);
29             build(2 * p + 1, m, r);
30             pull(p);
31         };
32         build(1, 0, n);
33     }
34     void pull(int p) {
35         info[p] = info[2 * p] + info[2 * p + 1];
36     }
37     void modify(int p, int l, int r, int x, const Info &v) {
38         if (r - l == 1) {
39             info[p] = v;
40             return;
41         }
42         int m = (l + r) / 2;
43         if (x < m) {
44             modify(2 * p, l, m, x, v);
45         } else {
46             modify(2 * p + 1, m, r, x, v);
47         }
48         pull(p);
49     }
50     void modify(int p, const Info &v) {
51         modify(1, 0, n, p, v);
52     }
53     Info rangeQuery(int p, int l, int r, int x, int y) {
54         if (l >= y || r <= x) {
55             return Info();
56         }

```

```

57         if (l >= x && r <= y) {
58             return info[p];
59         }
60         int m = (l + r) / 2;
61         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y
62             );
63     }
64     Info rangeQuery(int l, int r) {
65         return rangeQuery(1, 0, n, l, r);
66     }
67     template<class F>
68     int findFirst(int p, int l, int r, int x, int y, F pred) {
69         if (l >= y || r <= x || !pred(info[p])) {
70             return -1;
71         }
72         if (r - l == 1) {
73             return l;
74         }
75         int m = (l + r) / 2;
76         int res = findFirst(2 * p, l, m, x, y, pred);
77         if (res == -1) {
78             res = findFirst(2 * p + 1, m, r, x, y, pred);
79         }
80         return res;
81     }
82     template<class F>
83     int findFirst(int l, int r, F pred) {
84         return findFirst(1, 0, n, l, r, pred);
85     }
86     template<class F>
87     int findLast(int p, int l, int r, int x, int y, F pred) {
88         if (l >= y || r <= x || !pred(info[p])) {
89             return -1;
90         }
91         if (r - l == 1) {
92             return l;
93         }
94         int m = (l + r) / 2;
95         int res = findLast(2 * p + 1, m, r, x, y, pred);
96         if (res == -1) {
97             res = findLast(2 * p, l, m, x, y, pred);
98         }
99         return res;
100     }
101     template<class F>
102     int findLast(int l, int r, F pred) {
103         return findLast(1, 0, n, l, r, pred);
104     }
105 };
106 constexpr i64 inf = 1E18;
107

```

```

108 struct Info {
109     i64 cnt = 0;
110     i64 sum = 0;
111     i64 min = inf;
112 };
113
114 Info operator+(Info a, Info b) {
115     Info c;
116     c.cnt = a.cnt + b.cnt;
117     c.sum = a.sum + b.sum;
118     c.min = std::min(a.min, b.min);
119     return c;
120 }

```

### 3.4 懒标记线段树

```

1  const int N = 1e5 + 5;
2  int n, m, a[N], t[N<<2], x, y, w, lazy[N<<2];
3  void build(int rt, int l, int r) {
4      if (l == r) { t[rt] = a[l]; return; }
5      int mid = (l + r) >> 1;
6      build(rt << 1, l, mid);
7      build(rt << 1 | 1, mid + 1, r);
8      t[rt] = t[rt<<1] + t[rt<<1|1];
9  }
10 void updata(int rt, int l, int r, int w) {
11     t[rt] += (r - l + 1) * w;
12     lazy[rt] += w;
13 }
14 void pushdown(int rt, int l, int r) {
15     int mid = (l + r) >> 1;
16     updata(rt << 1, l, mid, lazy[rt]);
17     updata(rt << 1 | 1, mid + 1, r, lazy[rt]);
18     lazy[rt] = 0;
19 }
20 void add(int rt, int l, int r) {
21     if (x <= l && r <= y) {
22         updata(rt, l, r, w);
23         return;
24     }
25     pushdown(rt, l, r);
26     int mid = (l + r) >> 1;
27     if (x <= mid) add(rt << 1, l, mid);
28     if (y > mid) add(rt << 1 | 1, mid + 1, r);
29     t[rt] = t[rt<<1] + t[rt<<1|1];
30 }
31 int sum(int rt, int l, int r) {
32     if (x <= l && r <= y) return t[rt];
33     int mid = (l + r) >> 1, ans = 0;

```

```

34     pushdown(rt, l, r);
35     if (x <= mid) ans += sum(rt << 1, l, mid);
36     if (y > mid) ans += sum(rt << 1 | 1, mid + 1, r);
37     return ans;
38 }

```

### 3.5 倍增 LCA-最近公共祖先

```

1  void bfs()
2  {
3      dep[1]=1;
4      que[tail++]=1;
5      while(head<tail)
6      {
7          int p=que[head++];
8          for(int x=last[p];x!=0;x=pre[x])
9              {
10                 if(f[p][0]!=son[x])
11                     {
12                         dep[son[x]]=dep[p]+1;
13                         f[son[x]][0]=p;
14                         que[tail++]=son[x];
15                         for(int j=1;j<=20;j++)
16                             f[son[x]][j]=f[f[son[x]][j-1]][j-1];
17                     }
18             }
19     }
20 }
21 int lca(int a,int b)
22 {
23     if(dep[a]>dep[b]) swap(a,b);
24     for(int i=20;i>=0;i--)
25     {
26         if(dep[f[b][i]]>=dep[a]) b=f[b][i];
27         if(a==b) return a;
28     }
29     for(int i=20;i>=0;i--)
30         if(f[a][i]!=f[b][i]) a=f[a][i],b=f[b][i];
31     return f[a][0];
32 }

```

### 3.6 珂朵莉树

```

1  struct node {
2      int l, r;
3      mutable int v;
4      node(int a = 0, int b = 0, int c = 0) :l(a), r(b), v(c) { };

```

```

5      friend bool operator<(const node& a, const node& b) {
6          return a.l < b.l;
7      }
8  };
9
10 class ODT :public set<node> {
11 public:
12     ODT() :st(*this) { };
13     ODT(int l, int r, int v) :set<node>({ node(l,r,v) }), st(*this) { };
14     set<node>& st;
15
16     set<node>::iterator split(int pos) {
17         auto it = --st.upper_bound(node{ pos,0,0 });
18         if (it->l == pos) return it;
19         int l = it->l, r = it->r, v = it->v;
20         st.erase(it);
21         st.insert(node(l, pos - 1, v));
22         return st.insert(node(pos, r, v)).first;
23     }
24
25     void assign(int l, int r, int v) {
26         auto itr = split(r + 1), itl = split(l);
27         st.erase(itl, itr);
28         st.insert(node(l, r, v));
29     }
30 };

```

### 3.7 ST

```

1  template <typename T>
2  class SparseTable {
3      using VT = vector<T>;
4      using VVT = vector<VT>;
5      using func_type = function<T(const T &, const T &)>;
6
7      VVT ST;
8      VT A;
9
10     static T default_func(const T &t1, const T &t2) { return max(t1, t2); }
11
12     func_type op;
13
14 public:
15     SparseTable(const vector<T> &v, func_type _func = default_func) {
16         op = _func;
17         A = v;
18         int len = v.size(), l1 = ceil(log2(len)) + 1;
19         ST.assign(len, VT(l1, 0));
20         for (int i = 0; i < len; ++i) {
21             ST[i][0] = v[i];

```

```

22     }
23     for (int j = 1; j < 11; ++j) {
24         int pj = (1 << (j - 1));
25         for (int i = 0; i + pj < len; ++i) {
26             ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
27         }
28     }
29 }
30
31 T query(int l, int r) {
32     if(l == r) return A[l];
33     int lt = r - l + 1;
34     int q = ceil(log2(lt)) - 1;
35     return op(ST[l][q], ST[r - (1 << q) + 1][q]);
36 }
37 };

```

### 3.8 支配树

```

1  struct DominatorTree {
2      int n, cs;
3      std::vector<std::vector<int>>> E, RE, rdom;
4      std::vector<int> S, RS, par, val, sdom, rp, dom;
5
6      DominatorTree(int n) {
7          this->cs = 0;
8          this->n = n;
9          E.resize(n + 1);
10         RE.resize(n + 1);
11         rdom.resize(n + 1);
12         S.resize(n + 1);
13         RS.resize(n + 1);
14         par.resize(n + 1);
15         val.resize(n + 1);
16         sdom.resize(n + 1);
17         rp.resize(n + 1);
18         dom.resize(n + 1);
19         for (int i = 0; i <= n; i++)
20             {
21                 par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
22                 E[i].clear();
23                 RE[i].clear();
24                 rdom[i].clear();
25             }
26     }
27
28     void add_edge(int x, int y) {
29         E[x].push_back(y);
30     }

```



```

31
32 void Union(int x, int y) {
33     par[x] = y;
34 }
35
36 int Find(int x, int c = 0) {
37     if (par[x] == x)
38         return c ? -1 : x;
39     int p = Find(par[x], 1);
40     if (p == -1)
41         return c ? par[x] : val[x];
42     if (sdom[val[x]] > sdom[val[par[x]]])
43         val[x] = val[par[x]];
44     par[x] = p;
45     return c ? p : val[x];
46 }
47
48 void dfs(int x) {
49     RS[S[x] = ++cs] = x;
50     par[cs] = sdom[cs] = val[cs] = cs;
51     for (int e : E[x]) {
52         if (S[e] == 0)
53             dfs(e), rp[S[e]] = S[x];
54         RE[S[e]].push_back(S[x]);
55     }
56 }
57
58 int solve(int s, std::vector<int>& up) {
59     dfs(s);
60     for (int i = cs; i; i--) {
61         for (int e : RE[i])
62             sdom[i] = std::min(sdom[i], sdom[Find(e)]);
63         if (i > 1)
64             rdom[sdom[i]].push_back(i);
65         for (int e : rdom[i]) {
66             int p = Find(e);
67             if (sdom[p] == i)
68                 dom[e] = i;
69             else
70                 dom[e] = p;
71         }
72         if (i > 1)
73             Union(i, rp[i]);
74     }
75     for (int i = 2; i <= cs; i++)
76         if (sdom[i] != dom[i])
77             dom[i] = dom[dom[i]];
78     for (int i = 2; i <= cs; i++)
79         up[RS[i]] = RS[dom[i]];
80     return cs;
81 }
82 };

```



四、 计算几何

## 五、 字符串

## 5.1 AC 自动机

```
1 struct AC {
2     int n, tot, alp;
3     std::vector<int> fail;
4     std::vector<std::vector<int>> tr;
5     AC() {}
6     AC(int n, int m = 26) {
7         alp = m;
8         fail.resize(n);
9         tr.resize(n);
10        fail.assign(n, 0);
11        for (int i = 0; i < n; i++) {
12            tr[i].assign(m, 0);
13        }
14        init();
15    }
16    void init() {
17        tot = -1, new_node();
18    }
19    int new_node() { return ++tot, fail[tot] = 0, tr[tot].assign(tr[tot].size(),
20        0), tot; }
21    void insert(const std::string& s) {
22        for (int i = 0, u = 0; i < s.size(); i++) {
23            int c = s[i];
24            if (!tr[u][c]) tr[u][c] = new_node();
25            u = tr[u][c];
26        }
27    }
28    void build() {
29        std::queue<int> q;
30        int ql = 1, qr = 0;
31        for (int i = 0; i < alp; i++) {
32            if (tr[0][i]) {
33                q.push(tr[0][i]);
34            }
35        }
36        while(!q.empty()) {
37            int u = q.front();
38            q.pop();
39            for (int c = 0; c < alp; c++) {
40                if (tr[u][c]) fail[tr[u][c]] = tr[fail[u]][c], q.push(tr[u][c]);
41                else tr[u][c] = tr[fail[u]][c];
42            }
43        }
44    }
45};
```

## 5.2 KMP&amp; EXKMP

```

1  struct KMP {
2      std::vector<int> fail;
3      std::string pattern;
4      KMP() {}
5
6      KMP(const std::string& p) {
7          init(p);
8      }
9
10     void init(const std::string& p) {
11         pattern = p;
12         fail.resize(pattern.size() + 1);
13         fail[0] = -1; // 失配数组第一个元素为-1
14         int j = -1; // j表示失配数组的值
15         for (int i = 0; i < pattern.size(); i++) { // 遍历模式串
16             while (j >= 0 && pattern[i] != pattern[j]) { // 如果失配，则回溯
17                 j = fail[j]; // 回溯到失配位置的失配数组值
18             }
19             j++; // 失配数组值加1
20             fail[i + 1] = j; // 更新失配数组
21         }
22     }
23     // 匹配函数，返回所有匹配位置
24     std::vector<int> match(const std::string& s) {
25         std::vector<int> res;
26         int j = 0;
27         for (int i = 0; i < s.size(); i++) { // 遍历文本串
28             while (j >= 0 && s[i] != pattern[j]) { // 如果失配，则回溯
29                 j = fail[j]; // 回溯到失配位置的失配数组值
30             }
31             // dbg(i, j, s[i], pattern[j], pattern.size());
32             j++; // 失配数组值加1
33             if (j == pattern.size()) { // 如果匹配成功
34
35                 res.push_back(i - j + 1); // 存储匹配位置
36                 j = fail[j]; // 回溯到失配位置的失配数组值
37             }
38         }
39         return res; // 返回所有匹配位置
40     }
41 };
42 struct EXKMP {
43     string pattern;
44     vector<int> z;
45     EXKMP() {}
46
47     EXKMP(const std::string& p) {
48         init(p);
49     }

```

```

50
51 void init(const std::string& p) {
52     pattern = p;
53     int n = p.size();
54     z.resize(n);
55     z.assign(n, 0);
56     z[0] = p.size();
57     for (int i = 1, l, r = -1; i < n; i++) {
58         if (i <= r) z[i] = min(z[i - 1], r - i + 1);
59         while(i + z[i] < n && p[z[i]] == p[i + z[i]]) z[i]++;
60         if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
61     }
62 }
63 // 匹配函数, 返回所有位置的最长前缀
64 std::vector<int> match(const std::string& s) {
65     vector<int> pre(s.size(), 0);
66     int m = s.size(), n = pattern.size();
67     for (int i = 0, l, r = -1; i < m; i++) {
68         if (i <= r) pre[i] = min(z[i - 1], r - i + 1);
69         while(pre[i] < n && i + pre[i] < m && pattern[pre[i]] == s[i + pre[i]
70             ]) pre[i]++;
71         if (i + pre[i] - 1 > r) l = i, r = i + pre[i] - 1;
72     }
73     return pre;
74 };

```

### 5.3 Manacher

```

1 struct Manacher{
2     string s, t; // s 为原串 t 为补充后的串
3     int n; // t 的长度
4     vector<int> d; // 回文半径
5     ma(string s) : s(s), n(s.size() * 2 + 3) {
6         init(t);
7         build(d);
8     }
9     void init(string &t) {
10         t = "$#";
11         for (int i = 0; i < s.size(); i++) {
12             t += s[i];
13             t += "#";
14         }
15         t += '@';
16     }
17     void build(vector<int> &d) {
18         auto equ = [&](char l, char r) {
19             return l == r;
20         };
21         d = vector<int>(n, 0);

```

```

22     d[1] = 1;
23     for (int i = 2, l = 1, r = 1; i < n; i++) {
24         if (i <= r) d[i] = min(d[r + l - i], r - i + 1);
25         while (equ(t[i - d[i]], t[i + d[i]])) {
26             d[i]++;
27         }
28         if (i + d[i] - 1 > r) r = i + d[i] - 1, l = i - d[i] + 1;
29     }
30     return;
31 }
32 };

```

## 5.4 后缀自动机

```

1  class SAM {
2  public:
3      class state {
4      public:
5          state() = default;
6          state(int len, int link) :len(len), link(link) { };
7          state(int len, int link, map<char, int>& next) :len(len), link(link),
8              next(next) { };
9          int len, link;
10         std::map<char, int> next;
11     };
12     vector<state> st;
13     SAM() {
14         st.push_back(state(0, -1));
15     }
16     SAM(const string& s) :SAM() {
17         for (auto ch : s) {
18             sam_extend(ch);
19         }
20     }
21     void sam_extend(char ch) {
22         int p = st.size() - 1, cur = st.size();
23         st.push_back(state(st.back().len + 1, -1));
24         while (p != -1 && st[p].next.count(ch) == 0) {
25             st[p].next[ch] = cur;
26             p = st[p].link;
27         }
28         if (p == -1) {
29             st[cur].link = 0;
30         } else {
31             int q = st[p].next[ch];
32             if (st[q].len == st[p].len + 1) {
33                 st[cur].link = q;
34             } else {
35                 st.push_back(state(st[p].len + 1, st[q].link, st[q].next));

```

```

35         int clone = st.size() - 1;
36         while (p != -1 && st[p].next[ch] == q) {
37             st[p].next[ch] = clone;
38             p = st[p].link;
39         }
40         st[q].link = st[cur].link = clone;
41     }
42 }
43 }
44 };

```

```

1  struct SAM {
2      int vcnt, last;
3      std::vector<int> len, link;
4      std::vector<std::vector<int>> tr;
5
6      void init(int size) {
7          vcnt = last = 0;
8          len.resize(size);
9          link.resize(size);
10         tr.resize(size, std::vector<int>(26, 0));
11         link[0] = -1;
12     }
13
14     void clear() {
15         vcnt = last = 0;
16         len.clear();
17         link.clear();
18         tr.clear();
19     }
20
21     void add(int c) {
22         int cur = ++vcnt;
23         len[cur] = len[last] + 1;
24         int p = last;
25         while (p != -1 && !tr[p][c])
26             tr[p][c] = cur, p = link[p];
27         if (p == -1)
28             link[cur] = 0;
29         else {
30             int q = tr[p][c];
31             if (len[q] == len[p] + 1)
32                 link[cur] = q;
33             else {
34                 int clone = ++vcnt;
35                 len[clone] = len[p] + 1;
36                 link[clone] = link[q];
37                 tr[clone] = tr[q];
38                 while (p != -1 && tr[p][c] == q)
39                     tr[p][c] = clone, p = link[p];

```



```

40         link[q] = clone;
41         link[cur] = clone;
42     }
43 }
44 last = cur;
45 }
46 };
47
48 int main() {
49     SAM S;
50     int size = 2 * 100; // 根据需要的大小设置
51     S.init(size);
52
53     // 使用 SAM 对字符串进行处理
54     std::string input = "abcabca";
55     for (char c : input) {
56         int index = c - 'a';
57         S.add(index);
58     }
59
60     // 输出 SAM 中的一些信息
61     std::cout << "Number of states: " << S.vcnt << std::endl;
62     std::cout << "Last state: " << S.last << std::endl;
63
64     return 0;
65 }

```

## 5.5 序列自动机

```

1  class SequenceAM :public vector<vector<int>>{
2  public:
3      SequenceAM() = default;
4      SequenceAM(const string& s, int sigma = 26)
5          :vector<vector<int>>(s.size() + 1, vector<int>(sigma, 0)) {
6          auto &nxt=*this;
7          for(int i=s.size();i>=1;i--) {
8              nxt[i][s[i-1]-'a']=i;
9              nxt[i-1]=nxt[i];
10         }
11     }
12 };

```

## 六、 数学

## 6.1 欧拉函数

单独求欧拉函数  $\phi(x)$ 

```
1  int phi(int n) {
2      int m = 1;
3      for (int i = 2; n > 1; ++i) {
4          if (n % i == 0) {
5              m *= i - 1;
6              n /= i;
7              while (n % i == 0) {
8                  m *= i;
9                  n /= i;
10             }
11         }
12     }
13     return m;
14 }
```

预处理  $\phi(x)$ 

```
1  for (i = 1; i <= maxn; i++) phi[i] = i;
2  for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
3  for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
4      for (j = i; j <= maxn; j += i)
5          phi[j] = phi[j] / i * (i - 1);
6  }
```

## 6.2 扩展 GCD

求  $x, y$  满足  $\gcd(a, b) = a * x + b * y$ 

```
1  int exgcd(int a, int b, int & x, int & y) {
2      if(b == 0) {
3          x == 1, y == 1;
4          return a;
5      }
6      int ret = exgcd(b, a % b, x, y);
7      int tmp = x; x = y; y = tmp - a / b * y;
8      return ret;
9  }
```

## 6.3 各种筛

1 线性筛素数

2 保证每个数只会被它的最小质因子给筛掉（不同于埃氏筛中每个数会被它所有质因子筛一遍从而使复杂度过高）

3 `int pri[N],tot,zhi[N];`//`zhi[i]`为1的表示不是质数

4 `void sieve()`

5 {

6     `zhi[1]=1;`

7     `for (int i=2;i<=n;i++)`

8     {

9         `if (!zhi[i]) pri[++tot]=i;`

10         `for (int j=1;j<=tot&&i*pri[j]<=n;j++)`

11         {

12             `zhi[i*pri[j]]=1;`

13             `if (i%pri[j]==0) break;`

14         }

15     }

16 }

17 所有线性筛积性函数都必须基于线性筛素数。

18 线性筛莫比乌斯函数

19 `int mu[N],pri[N],tot,zhi[N];`

20 `void sieve()`

21 {

22     `zhi[1]=mu[1]=1;`

23     `for (int i=2;i<=n;i++)`

24     {

25         `if (!zhi[i]) pri[++tot]=i,mu[i]=-1;`

26         `for (int j=1;j<=tot&&i*pri[j]<=n;j++)`

27         {

28             `zhi[i*pri[j]]=1;`

29             `if (i%pri[j]) mu[i*pri[j]]=-mu[i];`

30             `else {mu[i*pri[j]]=0;break;}`

31         }

32     }

33 }

34 线性筛欧拉函数

35 `int phi[N],pri[N],tot,zhi[N];`

36 `void sieve()`

37 {

38     `zhi[1]=phi[1]=1;`

39     `for (int i=2;i<=n;i++)`

40     {

41         `if (!zhi[i]) pri[++tot]=i,phi[i]=i-1;`

42         `for (int j=1;j<=tot&&i*pri[j]<=n;j++)`

43         {

44             `zhi[i*pri[j]]=1;`

45             `if (i%pri[j]) phi[i*pri[j]]=phi[i]*phi[pri[j]];`

46             `else {phi[i*pri[j]]=phi[i]*pri[j];break;}`

47         }

48     }

49 }

50 线性筛约数个数

```

51 记d(i)
52 d(i)表示i的约数个数,d(i)=k (i=1)(ai+1) d(i)=i=1k(ai+1)
53 维护每一个数的最小值因子出现的次数 (即a1) 即可
54 int d[N],a[N],pri[N],tot,zhi[N];
55 void sieve()
56 {
57     zhi[1]=d[1]=1;
58     for (int i=2;i<=n;i++)
59     {
60         if (!zhi[i]) pri[++tot]=i,d[i]=2,a[i]=1;
61         for (int j=1;j<=tot&&i*pri[j]<=n;j++)
62         {
63             zhi[i*pri[j]]=1;
64             if (i%pri[j]) d[i*pri[j]]=d[i]*d[pri[j]],a[i*pri[j]]=1;
65             else {d[i*pri[j]]=d[i]/(a[i]+1)*(a[i]+2);a[i*pri[j]]=a[i]+1;break;}
66         }
67     }
68 }

```

## 6.4 多项式

```

1  using i64 = long long;
2  template<class T>
3  #define constexpr
4  constexpr T power(T a, i64 b) {
5      T res = 1;
6      for (; b; b /= 2, a *= a) {
7          if (b % 2) {
8              res *= a;
9          }
10     }
11     return res;
12 }
13
14 template<int P>
15 struct MInt {
16     int x;
17     constexpr MInt() : x{} {}
18     constexpr MInt(i64 x) : x{norm(x % getMod())} {}
19
20     static int Mod;
21     constexpr static int getMod() {
22         if (P > 0) {
23             return P;
24         } else {
25             return Mod;
26         }
27     }
28     constexpr static void setMod(int Mod_) {

```

```

29     Mod = Mod_;
30 }
31 constexpr int norm(int x) const {
32     if (x < 0) {
33         x += getMod();
34     }
35     if (x >= getMod()) {
36         x -= getMod();
37     }
38     return x;
39 }
40 constexpr int val() const {
41     return x;
42 }
43 explicit constexpr operator int() const {
44     return x;
45 }
46 constexpr MInt operator-() const {
47     MInt res;
48     res.x = norm(getMod() - x);
49     return res;
50 }
51 constexpr MInt inv() const {
52     assert(x != 0);
53     return power(*this, getMod() - 2);
54 }
55 constexpr MInt &operator*=(MInt rhs) & {
56     x = 1LL * x * rhs.x % getMod();
57     return *this;
58 }
59 constexpr MInt &operator+=(MInt rhs) & {
60     x = norm(x + rhs.x);
61     return *this;
62 }
63 constexpr MInt &operator-=(MInt rhs) & {
64     x = norm(x - rhs.x);
65     return *this;
66 }
67 constexpr MInt &operator/=(MInt rhs) & {
68     return *this *= rhs.inv();
69 }
70 friend constexpr MInt operator*(MInt lhs, MInt rhs) {
71     MInt res = lhs;
72     res *= rhs;
73     return res;
74 }
75 friend constexpr MInt operator+(MInt lhs, MInt rhs) {
76     MInt res = lhs;
77     res += rhs;
78     return res;
79 }
80 friend constexpr MInt operator-(MInt lhs, MInt rhs) {

```

```

81     MInt res = lhs;
82     res -= rhs;
83     return res;
84 }
85 friend constexpr MInt operator/(MInt lhs, MInt rhs) {
86     MInt res = lhs;
87     res /= rhs;
88     return res;
89 }
90 friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
91     i64 v;
92     is >> v;
93     a = MInt(v);
94     return is;
95 }
96 friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a)
97     {
98     return os << a.val();
99 }
100 friend constexpr bool operator==(MInt lhs, MInt rhs) {
101     return lhs.val() == rhs.val();
102 }
103 friend constexpr bool operator!=(MInt lhs, MInt rhs) {
104     return lhs.val() != rhs.val();
105 }
106 };
107 template<>
108 int MInt<0>::Mod = 1;
109
110 template<int V, int P>
111 constexpr MInt<P> CInv = MInt<P>(V).inv();
112
113 const int P = 998244353;
114 using Z = MInt<P>;
115
116 std::vector<int> rev;
117 template<int P>
118 std::vector<MInt<P>> roots{0, 1};
119
120 template<int P>
121 constexpr MInt<P> findPrimitiveRoot() {
122     MInt<P> i = 2;
123     int k = __builtin_ctz(P - 1);
124     while (true) {
125         if (power(i, (P - 1) / 2) != 1) {
126             break;
127         }
128         i += 1;
129     }
130     return power(i, (P - 1) >> k);
131 }

```

```

132
133 template<int P>
134 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
135
136 template<>
137 constexpr MInt<998244353> primitiveRoot<998244353> {31};
138
139 template<int P>
140 constexpr void dft(std::vector<MInt<P>> &a) {
141     int n = a.size();
142
143     if (int(rev.size()) != n) {
144         int k = __builtin_ctz(n) - 1;
145         rev.resize(n);
146         for (int i = 0; i < n; i++) {
147             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
148         }
149     }
150
151     for (int i = 0; i < n; i++) {
152         if (rev[i] < i) {
153             std::swap(a[i], a[rev[i]]);
154         }
155     }
156     if (roots<P>.size() < n) {
157         int k = __builtin_ctz(roots<P>.size());
158         roots<P>.resize(n);
159         while ((1 << k) < n) {
160             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1)
161                             );
162             for (int i = 1 << (k - 1); i < (1 << k); i++) {
163                 roots<P>[2 * i] = roots<P>[i];
164                 roots<P>[2 * i + 1] = roots<P>[i] * e;
165             }
166             k++;
167         }
168         for (int k = 1; k < n; k *= 2) {
169             for (int i = 0; i < n; i += 2 * k) {
170                 for (int j = 0; j < k; j++) {
171                     MInt<P> u = a[i + j];
172                     MInt<P> v = a[i + j + k] * roots<P>[k + j];
173                     a[i + j] = u + v;
174                     a[i + j + k] = u - v;
175                 }
176             }
177         }
178     }
179
180 template<int P>
181 constexpr void idft(std::vector<MInt<P>> &a) {
182     int n = a.size();

```

```

183     std::reverse(a.begin() + 1, a.end());
184     dft(a);
185     MInt<P> inv = (1 - P) / n;
186     for (int i = 0; i < n; i++) {
187         a[i] *= inv;
188     }
189 }
190
191 template<int P = 998244353>
192 struct Poly : public std::vector<MInt<P>> {
193     using Value = MInt<P>;
194
195     Poly() : std::vector<Value>() {}
196     explicit constexpr Poly(int n) : std::vector<Value>(n) {}
197
198     explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
199     constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}
200
201     template<class InputIt, class = std::_RequireInputIter<InputIt>>
202     explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}
203
204     template<class F>
205     explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
206         for (int i = 0; i < n; i++) {
207             (*this)[i] = f(i);
208         }
209     }
210
211     constexpr Poly shift(int k) const {
212         if (k >= 0) {
213             auto b = *this;
214             b.insert(b.begin(), k, 0);
215             return b;
216         } else if (this->size() <= -k) {
217             return Poly();
218         } else {
219             return Poly(this->begin() + (-k), this->end());
220         }
221     }
222     constexpr Poly trunc(int k) const {
223         Poly f = *this;
224         f.resize(k);
225         return f;
226     }
227     constexpr friend Poly operator+(const Poly &a, const Poly &b) {
228         Poly res(std::max(a.size(), b.size()));
229         for (int i = 0; i < a.size(); i++) {
230             res[i] += a[i];
231         }

```



```

232     for (int i = 0; i < b.size(); i++) {
233         res[i] += b[i];
234     }
235     return res;
236 }
237 constexpr friend Poly operator-(const Poly &a, const Poly &b) {
238     Poly res(std::max(a.size(), b.size()));
239     for (int i = 0; i < a.size(); i++) {
240         res[i] += a[i];
241     }
242     for (int i = 0; i < b.size(); i++) {
243         res[i] -= b[i];
244     }
245     return res;
246 }
247 constexpr friend Poly operator-(const Poly &a) {
248     std::vector<Value> res(a.size());
249     for (int i = 0; i < int(res.size()); i++) {
250         res[i] = -a[i];
251     }
252     return Poly(res);
253 }
254 constexpr friend Poly operator*(Poly a, Poly b) {
255     if (a.size() == 0 || b.size() == 0) {
256         return Poly();
257     }
258     if (a.size() < b.size()) {
259         std::swap(a, b);
260     }
261     int n = 1, tot = a.size() + b.size() - 1;
262     while (n < tot) {
263         n *= 2;
264     }
265     if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
266         Poly c(a.size() + b.size() - 1);
267         for (int i = 0; i < a.size(); i++) {
268             for (int j = 0; j < b.size(); j++) {
269                 c[i + j] += a[i] * b[j];
270             }
271         }
272         return c;
273     }
274     a.resize(n);
275     b.resize(n);
276     dft(a);
277     dft(b);
278     for (int i = 0; i < n; ++i) {
279         a[i] *= b[i];
280     }
281     idft(a);
282     a.resize(tot);
283     return a;

```

```

284     }
285     constexpr friend Poly operator*(Value a, Poly b) {
286         for (int i = 0; i < int(b.size()); i++) {
287             b[i] *= a;
288         }
289         return b;
290     }
291     constexpr friend Poly operator*(Poly a, Value b) {
292         for (int i = 0; i < int(a.size()); i++) {
293             a[i] *= b;
294         }
295         return a;
296     }
297     constexpr friend Poly operator/(Poly a, Value b) {
298         for (int i = 0; i < int(a.size()); i++) {
299             a[i] /= b;
300         }
301         return a;
302     }
303     constexpr Poly &operator+=(Poly b) {
304         return (*this) = (*this) + b;
305     }
306     constexpr Poly &operator-=(Poly b) {
307         return (*this) = (*this) - b;
308     }
309     constexpr Poly &operator*=(Poly b) {
310         return (*this) = (*this) * b;
311     }
312     constexpr Poly &operator*=(Value b) {
313         return (*this) = (*this) * b;
314     }
315     constexpr Poly &operator/=(Value b) {
316         return (*this) = (*this) / b;
317     }
318     constexpr Poly deriv() const {
319         if (this->empty()) {
320             return Poly();
321         }
322         Poly res(this->size() - 1);
323         for (int i = 0; i < this->size() - 1; ++i) {
324             res[i] = (i + 1) * (*this)[i + 1];
325         }
326         return res;
327     }
328     constexpr Poly integr() const {
329         Poly res(this->size() + 1);
330         for (int i = 0; i < this->size(); ++i) {
331             res[i + 1] = (*this)[i] / (i + 1);
332         }
333         return res;
334     }
335     constexpr Poly inv(int m) const {

```

```

336     Poly x{(*this)[0].inv()};
337     int k = 1;
338     while (k < m) {
339         k *= 2;
340         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
341     }
342     return x.trunc(m);
343 }
344 constexpr Poly log(int m) const {
345     return (deriv() * inv(m)).integr().trunc(m);
346 }
347 constexpr Poly exp(int m) const {
348     Poly x{1};
349     int k = 1;
350     while (k < m) {
351         k *= 2;
352         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
353     }
354     return x.trunc(m);
355 }
356 constexpr Poly pow(int k, int m) const {
357     int i = 0;
358     while (i < this->size() && (*this)[i] == 0) {
359         i++;
360     }
361     if (i == this->size() || 1LL * i * k >= m) {
362         return Poly(m);
363     }
364     Value v = (*this)[i];
365     auto f = shift(-i) * v.inv();
366     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k)
367         ;
368 }
369 constexpr Poly sqrt(int m) const {
370     Poly x{1};
371     int k = 1;
372     while (k < m) {
373         k *= 2;
374         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
375     }
376     return x.trunc(m);
377 }
378 constexpr Poly multT(Poly b) const {
379     if (b.size() == 0) {
380         return Poly();
381     }
382     int n = b.size();
383     std::reverse(b.begin(), b.end());
384     return ((*this) * b).shift(-(n - 1));
385 }
386 constexpr std::vector<Value> eval(std::vector<Value> x) const {
387     if (this->size() == 0) {

```

```

387         return std::vector<Value>(x.size(), 0);
388     }
389     const int n = std::max(x.size(), this->size());
390     std::vector<Poly> q(4 * n);
391     std::vector<Value> ans(x.size());
392     x.resize(n);
393     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
394         if (r - l == 1) {
395             q[p] = Poly{1, -x[l]};
396         } else {
397             int m = (l + r) / 2;
398             build(2 * p, l, m);
399             build(2 * p + 1, m, r);
400             q[p] = q[2 * p] * q[2 * p + 1];
401         }
402     };
403     build(1, 0, n);
404     std::function<void(int, int, int, const Poly &)> work = [&](int p, int
405         l, int r, const Poly &num) {
406         if (r - l == 1) {
407             if (l < int(ans.size())) {
408                 ans[l] = num[0];
409             }
410         } else {
411             int m = (l + r) / 2;
412             work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - 1));
413             work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
414         }
415     };
416     work(1, 0, n, mulT(q[1].inv(n)));
417     return ans;
418 }
419 };
420 template<int P = 998244353>
421 Poly<P> berlekampMassey(const Poly<P> &s) {
422     Poly<P> c;
423     Poly<P> oldC;
424     int f = -1;
425     for (int i = 0; i < s.size(); i++) {
426         auto delta = s[i];
427         for (int j = 1; j <= c.size(); j++) {
428             delta -= c[j - 1] * s[i - j];
429         }
430         if (delta == 0) {
431             continue;
432         }
433         if (f == -1) {
434             c.resize(i + 1);
435             f = i;
436         } else {
437             auto d = oldC;

```

```

438         d *= -1;
439         d.insert(d.begin(), 1);
440         MInt<P> df1 = 0;
441         for (int j = 1; j <= d.size(); j++) {
442             df1 += d[j - 1] * s[f + 1 - j];
443         }
444         assert(df1 != 0);
445         auto coef = delta / df1;
446         d *= coef;
447         Poly<P> zeros(i - f - 1);
448         zeros.insert(zeros.end(), d.begin(), d.end());
449         d = zeros;
450         auto temp = c;
451         c += d;
452         if (i - temp.size() > f - oldC.size()) {
453             oldC = temp;
454             f = i;
455         }
456     }
457 }
458 c *= -1;
459 c.insert(c.begin(), 1);
460 return c;
461 }
462
463
464 template<int P = 998244353>
465 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
466     int m = q.size() - 1;
467     while (n > 0) {
468         auto newq = q;
469         for (int i = 1; i <= m; i += 2) {
470             newq[i] *= -1;
471         }
472         auto newp = p * newq;
473         newq = q * newq;
474         for (int i = 0; i < m; i++) {
475             p[i] = newp[i * 2 + n % 2];
476         }
477         for (int i = 0; i <= m; i++) {
478             q[i] = newq[i * 2];
479         }
480         n /= 2;
481     }
482     return p[0] / q[0];
483 }

```

## 6.5 斯特林数

```
1 std::vector S(n + 2, std::vector<Z>(n + 2));
2 for (int i = 0; i <= n + 1; i++) {
3     S[i][0] = !i;
4     for (int j = 1; j <= i; j++) {
5         S[i][j] = S[i - 1][j - 1] + S[i - 1][j] * j;
6     }
7 }
```

## 6.6 高斯消元

```
1 void gauss(int n, double g[maxn][maxn]) { // input: N * (N + 1) Matrix
2     for (int i = 1; i <= n; ++i) {
3         double temp = 0;
4         int pos = -1;
5         for (int j = i; j <= n; ++j) {
6             if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
7         }
8         if (pos == -1) continue;
9         for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
10        temp = g[i][i];
11        for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
12        for (int j = i + 1; j <= n; ++j) {
13            temp = g[j][i];
14            for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k];
15        }
16    }
17    for (int i = n; i >= 1; --i) {
18        for (int j = 1; j < i; ++j) {
19            g[j][n + 1] -= g[i][n + 1] * g[j][i];
20            g[j][i] = 0;
21        }
22    }
23 }
```

## 七、 其他

## 7.1 快读快写

```
1  template <typename T> inline void read(T& t) {
2      int f = 0, c = getchar(); t = 0;
3      while (!isdigit(c)) f |= c == '-', c = getchar();
4      while (isdigit(c)) t = t * 10 + c - 48, c = getchar();
5      if (f) t = -t;
6  }
7
8  template <typename T> void print(T x) {
9      if (x < 0) x = -x, putchar('-');
10     if (x > 9) print(x / 10);
11     putchar(x % 10 + 48);
12 }
```