代码簿

August 30, 2022

1. 杂项

- **VIMRC**

```
1 set hls
2 set is
3 set cb=unnamed
4 set ts=4
5 set sw=4
6 set si
7 set nu
8
9 inoremap { {}<Left>
10 inoremap {<CR> {<CR>}<Esc>O
11 inoremap {{ {
12 inoremap {} {}
```

- **Merge Sort**

```
1 int arr[MAXLEN]; // 数组
2 int tmp[MAXLEN]; // 暂存
3 int ans  = 0; // 逆序对
4
5 void mergeSort(int left, int right){
6   if(left >= right) return;
7   int mid = left + ((right - left) >> 1);
8   mergeSort(left, mid);
9   mergeSort(mid + 1, right);
10   int l = left;
11   int r = mid + 1;
12   int idx = 0;
13   while(l <= mid && r <= right){
14     tmp[idx++] = arr[l] <= arr[r] ? arr[l++]
15     : ((ans += mid - l + 1 /* 逆序对 */ ), arr[r++]);
16   }
17   while(l <= mid){
18     tmp[idx++] = arr[l++];
19   }
20   while(r <= right){
21     tmp[idx++] = arr[r++];
22   }
23   for(int i = 0; i < idx; ++i){
24     arr[left + i] = tmp[i];
25   }
26 }
```

- **Disjoint Set**

```
1 int djs[MAXLEN];
2 //init for(int i = 0; i < size; ++i)djs[i] = i;
3
4 int find(int id){
5   return djs[id] == id ? id : djs[id] = find(djs[id]);
6 }
7 bool uSet(int a, int b){
8   int pa = find(a), pb = find(b);
9   return pa == pb ? true : (djs[pa] = pb, false);
10 }
11
12 //with dummy head
13
14 int djs[MAXLEN];
15 //init  for(int i = 0; i < 2 * n; ++i) djs[i] = i; for(int i = 0; i < n; ++i) djs[i] = n + i;
16
17 int find(int id){
18   return djs[id] == id ? id : djs[id] = find(djs[id]);
```

```
19 }
20 bool uSet(int a, int b){
21     int pa = find(a), pb = find(b);
22     return pa == pb ? true : (djs[pa] = pb, false);
23 }
```

2. 图论

- **Eular Path**

```
1 int cd[MAXLEN], rd[MAXLEN]; //有向图
2 vector<pair<int, int>> adjs[MAXLEN]; //first for 点, second fir 边
3 bool visit[MAXLEN]; //边
4
5 int cnt = 0; //判断联通(cnt == size)
6
7 void dfs(int id){
8     for(auto &adj : adjs){
9         if(!visit[adj.second]){
10             ++cnt;
11             visit[adj.second] = true;
12             dfs(adj.first);
13             //一些操作
14         }
15     }
16 }
17
18 bool judge(){
19     //无向(边为偶数除了端点)
20     int cnt = 0;
21     int qidian  = 0; // 起点
22     for(int i = 0; i < n; ++i){
23         if(adjs[i].size() & 1) ++cnt, qidian = i;
24     }
25     return (cnt == 2 || cnt == 0);
26
27     //有向((出度 - 入度) = 1 && (入度 - 出度) = 1 || 欧拉环)
28     int rdc = 0, cdc = 0;
29     int qidian = 0;
30     for(int i = 0; i < n; ++i){
31         if((cd[i] - rd[i] == 1)) ++cdc;
32         else if((rd[i] - cd[i]) == 1) ++rdc, qidian = i;
33         else if((rd[i] - cd[i]) != 0) return false;
34     }
35
36     return ((cdc == 1 && rdc == 1) || (cdc == 0 && rdc == 0));
37 }
```

- **拓扑排序**

```
1 vector<int> adjs[MAXLEN];
2 int rd[MAXLEN]; //入度
3 int n; //点的数量
4 vector<int> ans; //结果
5
6 void topSort(){
7     queue<int> q;
8     int cnt= 0;
9     for(int i = 0; i < n; ++i) if(!rd[i]) q.push(i), ++cnt, ans.push_back(i);
10     while(!q.empty()){
11         int f = q.front(); q.pop();
12         for(auto adj : adjs[f]){
13             --rd[adj];
14             if(!rd[adj]) q.push(adj), ++cnt, ans.push_back(adj);
15         }
16     }
17     if(cnt == n) {}//有找到
18     else {} //没找到
19 }
```

- **多元最短路**

```
1 int v[MAXLEN][MAXLEN];
2 //初始化成0x3f3f3f3f, 有边的地方为权重, 自己为0
3
4 for(int k = 0; k < n; ++k){
5     for(int  i = 0; i < n; ++i){
6         for(int  j = 0; j < n; ++j){
7             v[i][j] = min(v[i][k] + v[k][j], v[i][j]); //最短路径
```

```
8              v[i][j] = min(v[i][j], max(v[i][k], v[k][j])); //最大的最小
9          }
10     }
11 }
```

- **单元最短路**

```
1  vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 边权
2  int dis[MAXLEN];
3  bool visit[MAXLEN];
4
5  void dij(int id){
6      memset(dis, 0x3f, sizeof(dis));
7      memset(visit, 0, sizeof(visit));
8      dis[id] = 0;
9      priority_queue<pii, vector<pii>, greater<pii>> q;
10
11     //first for 目前距离 second for 点
12     q.push({0, id});
13     while(!q.empty()){
14         pair<int, int> front = q.top(); q.pop();
15         if(visit[front.second]) continue;
16         visit[front.second] = true;
17         for(auto adj : adjs[front.second]){
18             if((adj.second + dis[front.second]) <  dis[adj.first]){
19                 dis[adj.first] = adj.second + dis[front.second];
20                 if(visit[adj.first]) continue;
21                 q.push({dis[adj.first], adj.first});
22             }
23         }
24     }
25 }
```

- **找负环**

```
1  vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 权重
2  int dis[MAXLEN]; //最短路径
3  int m, n; //n 点数 m 边数
4
5  bool bellman(){
6      memset(dis, 0x3f, 4 * n);
7      for(int c = 0; c < n - 1; ++c){
8        for(int i = 0; i < n; ++i){
9          for(auto adj : adjs[i]){
10           dis[adj.first] = min(dis[adj.first], dis[i] + adj.second);
11         }
12       }
13     }
14     bool hasAnswer = false;
15   for(int i = 0; i < n; ++i){
16     for(auto adj : adjs[i]){
17       if(dis[adj.first] > dis[i] + adj.second){
18         hasAnswer = true;
19         goto ans;
20       }
21     }
22   }
23     ans:
24     return hasAnswer;
25 }
```

- **MST**

```
1  // kruskal
2
3  pair<int, pii> adjs[MAXLEN]; //first for 权重 //second.first for 起点 //second.second for 终点
4  int n; //边的数量
5  int m; //点的数量
6  int djs[MAXLEN];
7
8
9  int parent(int i){
10     return (djs[i] == i) ? i : djs[i] = parent(djs[i]);
11 }
12
13 void kruskal(){
14     for(int i = 0; i <= m; ++i){
15         djs[i] = i;
16     }
17
```

```
18      sort(adjs, adjs + n);
19      int cnt = m; //计算边的数量
20      for(int i = 0; i < n; ++i){
21          int p1 = parent(adjs[i].second.first), p2 = parent(adj[i].second.second);
22          if(p1 == p2) continue;
23          djs[p1] = p2;
24          //一些操作
25          if((--cnt) == 1) break;
26      }
27  }
28
29  //Prim 普利姆算法
30
31  vector<pii> adjs[MAXLEN]; //first for 点, second for 权重
32  bool visit[MAXLEN];
33
34  int n, m;
35  void prim(){
36    priority_queue<pii, vector<pii>, greater<pii>> q;
37    int ans = 0; //最小距离
38    q.push({0, 1});
39    int cnt = n; //是否联通
40    while(!q.empty()){
41      pii top = q.top();q.pop();
42      if(visit[top.second])continue;
43      ans += top.first;
44      visit[top.second] = true;
45      --cnt;
46      for(const pii &adj : adjs[top.second]){
47        if(!visit[adj.first]){
48          q.push({adj.second, adj.first});
49        }
50      }
51    }
52    if(cnt){
53      printf("orz\n");
54    }else{
55      printf("%d\n", ans);
56    }
57
58  }
```

- **最大流 (Dinic)**

```
1   struct edge
2   {
3       int to;
4       long long val;
5       int rev;
6       edge(int _to, long long _val, int _rev):to(_to), val(_val), rev(_rev){}
7   };
8
9   int n, m, s, t;
10
11
12  vector<edge> adjs[MAXLEN];
13
14  int depth[MAXLEN]; //深度
15
16  bool bfs(){ //分层 + 确定没有回去找
17      memset(depth, 0, sizeof(depth));
18      depth[s] = 1;
19      queue<int> q;
20      q.push(s);
21      while(!q.empty()){
22          int f = q.front(); q.pop();
23          for(edge &adj : adjs[f]){
24              if(!depth[adj.to] && adj.val){
25                  depth[adj.to] = depth[f] + 1;
26                  q.push(adj.to);
27              }
28          }
29      }
30      return depth[t];
31  }
32
33  long long dfs(int u = s, long long in = 0x3f3f3f3f3f3f3f3f){ //多路增广
34      if(u == t) return in;
35      if(in == 0) return 0;
36      long long out = 0;
```

```
37      for(edge &adj : adjs[u]){
38          if((depth[adj.to] == (depth[u] + 1)) && adj.val){
39              long long dist = dfs(adj.to, min(in, adj.val));
40              adj.val -= dist;
41              adjs[adj.to][adj.rev].val += dist;
42              in -= dist;
43              out += dist;
44          }
45      }
46      if(!out) depth[u] = 0;
47      return out;
48  }
49
50  long long ans = 0;
51
52  void dinic(){
53      while(bfs()){
54          while(long long d = dfs()){
55              ans += d;
56          }
57      }
58  }
```

- **费用流 (Dinic)**

```
1  struct edge
2  {
3      int to;
4      int val;
5      int rev;
6      int cost;
7      edge(int _to, int _val, int _rev, int _cost):to(_to), val(_val), rev(_rev), cost(_cost){} //反向边
         的cost要是负数
8  };
9
10 int n, m, s, t;
11
12 vector<edge> adjs[MAXLEN];
13
14 int dis[MAXLEN];
15 bool visit[MAXLEN];
16
17
18
19 long long zuidaliu = 0, zuixiaofeiyong = 0; //最大流and最小费用
20
21 bool spfa(int id = t){ //BellMan-Ford
22   memset(visit, 0, sizeof(visit));
23   memset(dis, 0x3f, sizeof(dis));
24   visit[id]  = true;
25   dis[id] = 0;
26   deque<int> q;
27   q.push_front(id);
28
29   while(!q.empty()){
30     int front = q.front(); q.pop_front();
31
32     for(edge &adj : adjs[front]){
33       if(adjs[adj.to][adj.rev].val > 0 && dis[adj.to]  > dis[front] - adj.cost ){
34         dis[adj.to] = dis[front] - adj.cost;
35         if(!visit[adj.to]){
36           visit[adj.to] = true;
37           if(!q.empty() && dis[adj.to] < dis[q.front()]) q.push_front(adj.to); //SLF优化
38           else q.push_back(adj.to);
39         }
40       }
41     }
42     visit[front] = false;
43   }
44   return dis[s] != 0x3f3f3f3f;
45 }
46
47 int dfs(int u = s, int flow =  0x3f3f3f3f){ //多路增广
48   if(u == t){
49     visit[u] = true;
50     return flow;
51   }
52   int in = flow;
53   int out = 0;
54   visit[u] = true;
```

```
55    for(edge &adj : adjs[u]){
56
57      if(adj.val >0 && !visit[adj.to] && dis[adj.to] == (dis[u] - adj.cost)){
58        int dis = dfs(adj.to, min(in, adj.val));
59        if(dis > 0){
60          in -= dis;
61          adj.val -= dis;
62          out += dis;
63          zuixiaofeiyong += dis * adj.cost;
64          adjs[adj.to][adj.rev].val += dis;
65        }
66        if(!in) break;
67
68      }
69    }
70
71    return out;
72 }
73
74
75
76 void dinic(){
77    while(spfa()){
78      while(int d = dfs()){
79        memset(visit, 0, sizeof(visit));
80        zuidaliu += d;
81      }
82    }
83 }
```

3. 树问题

- **树直径**

```
1  //DFS两次
2
3  int d[MAXN];
4  vector<int> adjs[MAXN];
5
6  int last; //最远点， d[last] 为数的直径 不可做负边
7
8  int bfs(int u, int p){
9      queue<pair<int, int>> q;
10     q.push({u, p});
11     while(!q.empty()){
12         auto [u, p] = q.front(); q.pop();
13         for(const int adj : adjs[u]){
14             if(adj == p)
15                 continue;
16             d[adj] = d[u] + 1;
17             last = adj;
18             q.push({adj, u});
19         }
20     }
21 }
22
23 int farPoint(int u, int p){
24     last = u;
25     bfs(u, p);
26     d[last] = 0;
27     bfs(last, p);
28     return d[last];
29 }
30
31 //树形DP， 可做负边
32
33 int n, d = 0;
34 int d1[N], d2[N];
35 vector<int> adjs[N];
36
37 void dfs(int u, int p) {
38   d1[u] = d2[u] = 0;
39   for (int v : adjs[u]) {
40     if (v == p)
41         continue;
42     dfs(v, u);
43     int t = d1[v] + 1; //可改权重
44     if (t > d1[u])
45         d2[u] = d1[u], d1[u] = t;
```

```
46        else if (t > d2[u])
47            d2[u] = t;
48    }
49    d = max(d, d1[u] + d2[u]);
50 }
```

- **LCA**

```
1  //倍增
2
3  int dep[MAXN];
4  int pa[MAXN][32];
5  vector<int> ch[MAXN];
6
7
8  void dfs(int u, int p){ // Pre-process
9      dep[u] = dep[p] + 1;
10     pa[u][0] = p;
11
12     for(int i = 1; i < 32; ++i)
13         pa[u][i] = pa[pa[u][i - 1]][i - 1];
14
15     for(const int v : ch[u]){
16         if(v == p)
17             continue;
18         dfs(v, u);
19     }
20 }
21
22 int lca(int a, int b){
23     if(dep[a] > dep[b])
24         swap(a, b);
25
26     int diff = dep[b] - dep[a];
27     int i = 0;
28
29     //调整高度
30     for(; diff; ++i, diff >>= 1)
31         if(diff & 1)
32             b = pa[b][i];
33
34     if(a == b)
35         return a;
36
37     i = 0;
38
39     for(i = 31; i >= 0; --i)
40         if(pa[a][i] != pa[b][i])
41             a = pa[a][i], b = pa[b][i];
42
43     return pa[a][0];
44
45 }
```

4. 计算几何

- **凸包**

```
1  //Andrew
2
3
4  // 点，上凸包，下凸包
5
6  using pdd = pair<double, double>
7  pdd ps[MAXN], u[MAXN], d[MAXN];
8
9  int ui = 0, di = 0;
10
11 inline double cross(const pdd &p1, const pdd &p2, const pdd &p3){
12     return (p2.first - p1.first) * (p3.second - p1.second) - (p2.second - p1.second) * (p3.first - p1.
       first);
13 }
14
15
16 void andrew(){
17     sort(ps, ps + n);
18     for(int i = 0; i < n; ++i){
19         while(ui >= 2 && cross(u[ui - 2], u[ui - 1], ps[i]) <= 0)
20             --ui;
```

```
21        while(di >= 2 && cross(d[di - 2], d[di - 1], ps[i]) >= 0)
22            --di;
23
24        u[ui++] = ps[i];
25        d[di++] = ps[i];
26    }
27
28 }
```

5. 数论

- **Miller Robin(确认素数)**

```
1 // allow condition 1, 1, 1, 1, .....
2 //                 X, X, X, X, X, p - 1, 1, 1, 1, ....
3 //                 p - 1, 1, 1, 1, 1, 1, .....
4 //                 0 (cannot check) multiply of p
5
6 bool is_prime(ll p){
7     if(p < 3 || p % 2 == 0)
8         return p == 2;
9     ll u = p - 1;
10    int t = 0;
11    while(!(u & 1))
12        u >>= 1, ++t;
13
14    const static ll ud[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
15
16    for(const ll a : ud){
17        ll v = qpow(a, u, p);
18        if(v <= 1 || v == p - 1)
19            continue;
20
21        for(int j = 1; j <= t; ++j){
22            v = qmul(v, v, p);
23            if(v == p - 1 && j != t){
24                v = 1;
25                break;
26            }
27            if(v == 1)
28                return false;
29        }
30        if(v != 1)
31            return false;
32    }
33    return true;
34 }
```

- **Pollard-Rho(确认因数)**

```
1 // remember srand(time(NULL))
2
3 ll PR(ll n){
4     if(n == 4)
5         return 2;
6
7     if(is_prime(n))
8         return n;
9
10    while(1){
11        ll c = (rand() % (n - 1)) + 1;
12        auto f = [&](const ll x){
13            return ((lll) x * x + c) % n;
14        };
15        ll t = 0, r = 0, p = 1, q;
16        do{
17            for(int i = 0; i < 128; ++i){
18                t = f(t), r = f(f(r));
19                if(t == r || (q = (lll) p * abs(t - r)  % n) == 0)
20                    break;
21                p = q;
22            }
23            ll d = __gcd(p, n);
24            if(d > 1)
25                return d;
26        }while(t != r);
27    }
28 }
```

- **FastGCD**

```cpp
// 定理1: gcd(x, y) = gcd(y, x mod y) 辗转相除法
// 定理2: x = a * b * c, where a, b,c <= sqrt(x) or isPrime
// 定理3: gcd(x, y) = gcd(a, y) * gcd(x / a, y / gcd(a, y))


int minp[MAXN] = {0}; //最小质因数
int g[1010][1010]; //1000*1000 最大公因数表
int fac[MAXN][3]; //三个拆分的因数
bitset<MAXN> vis;

void init(){
    fac[1][0] = fac[1][1] = fac[1][2] = 1;
    for(int i = 1; i < MAXN; ++i){
        if(!vis[i]){ // Prime
            fac[i][0] = fac[i][1] = fac[i][2] = i;
            if(1ll * i * i > MAXN)
                continue;
            for(int j = i * i; j <= MAXN; j += i){
                vis[j] = 1;
                if(!minp[j])
                    minp[j] = i;
            }
        }else{
            int tmp = i / minp[i], a = minp[i] * fac[tmp][0], b = fac[tmp][1], c = fac[tmp][2];
            if(a > b)
                swap(a, b);
            if(b > c)
                swap(b, c);
            fac[i][0] = a, fac[i][1] = b, fac[i][2] = c;
        }
    }

    for(int i = 1; i <= 1000; ++i){
        g[i][0] = g[0][i] = i;
        for(int j = 1; j <= i; ++j)
            g[i][j] = g[j][i] = g[j][i % j];
    }
}

itn gcd(int x, int y){
    int tmp = g[fac[x][0]][y % fac[x][0]], res = tmp;
    y /= tmp;
    tmp = g[fac[x][1]][y % fac[x][1]], res *= tmp, y /= tmp;
    tmp = fac[x][2] > 1000 ? (y % fac[x][2] == 0 ? fac[x][2] : 1) : g[fac[x][2]][y % fac[x][2]], res *= tmp;
    return res;
}
```

- **欧拉函数**

```cpp
// 单一

int Ep(int n){
    int lim = sqrt(n);
    int res = n;

    for(int i = 2; i <= lim; ++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;
}

// 值域

bool vis[MAXN] = {false};
int primes[MAXN];
int phi[MAXN];

void build(){
    int tot = 0;
    phi[1] = 1;
```

```
29    for(int i = 2; i <= n; ++i){
30        if(!vis[i])
31            primes[tot++] = i, phi[i] = i - 1;
32        for(int j = 0; j < tot && i * primes[j] <= n; ++j){
33            vis[i * primes[j]]  = true;
34            if(i % primes[j])
35                phi[i * primes[j]] = phi[i] * (primes[j] - 1);
36            else{
37                phi[i * primes[j]] = primes[j] * phi[i];
38                break;
39            }
40        }
41    }
42 }
```

- **扩展欧几里得**

```
1  // ax+by=gcd(a, b) if gcd(a, b) == 1
2  // ax同余c(mod b) <=> ax + by = c
3  // c 同余 a / b (mod M) = bx 同余 a (mod M)
4
5  ll Exgcd(ll a, ll b, ll &x, ll &y){
6      if(!b)
7          return x = 1, y = 0, a;
8      ll d = Exgcd(b, a % b, x, y);
9      ll t = x;
10     x = y;
11     y = t - (a / b) * y;
12     return d;
13 }
14
15 //线性
16
17 void inv(){
18     inv[1] = 1;
19     for(int i = 2; i <= n; ++i)
20       inv[i] = (ll)(p - p / i) * inv[p % i] % p;
21 }
22
23
24 //随机线性
25
26 int arr[n + 1], ji[n + 1], rev[n + 1], inv[n + 1]; // 数组，前缀积，前缀积逆元
27
28 ji[0] = 1;
29
30 for(int i = 1; i <= n; ++i)
31     ji[i] = ji[i - 1] * arr[i] % p;
32
33 rev[n] = (exgcd(ji[n], p), x);
34
35 for(int i = n; i >= 1; --i)
36     rev[i - 1] = (ll)rev[i] * arr[i] % p;
37
38 for(int i = 1; i <= n; ++i)
39     inv[i] = (ll)rev[i] * ji[i - 1] % p;
```

- **中国剩余定理**

```
1  // r互素
2
3  int a[MAXN], r[MAXN];
4  int n = 1, ans = 0;
5
6
7  for(int i = 0; i < k; ++i)
8    n *= r[i];
9
10 for(int i = 0; i < k; ++i){
11   ll m = n / r[i];
12   exgcd(m, r[i]);
13   ans = (ans + ((((lll)a[i] * m) % n) * x % n)) % n;
14 }
15
16
17 // r非互素
18
19 ll  M = r[0],  ans = a[0];
20
21 for(int i = 1; i < k; ++i){
```

```
22    ll da = (((a[i] - ans) % r[i]) + r[i]) % r[i];
23    ll g = exgcd(M, r[i]);
24    if(da % g)
25      return -1;
26    auto tmp = (lll)M * x * (da / g);
27    M = M * (r[i] / g);
28    ans = (((ans + tmp) % M) + M) % M;
29 }
```

- **exBSGS**

```
1  // a^x 同于 b (mod p)
2
3  int a, b, p, res;
4  unordered_map<int, int> mp;
5
6  bool bsgs(){
7      a %= p, b %= p;
8      if(b == 1 || p == 1){
9        res = 0;
10       return true;
11     }
12     ll ax =1;
13     int k = 0, g;
14     while((g = exgcd(a, p)) != 1){
15       if(b % g)
16         return false;
17       ++k;
18       b /= g, p /= g;
19       ax = ((ll)ax * (a / g)) % p;
20       if(ax == b){
21         res = k;
22         return true;
23       }
24     }
25
26     exgcd(ax , p);
27     b = ((ll)b * (((x % p) + p) % p)) % p;
28
29     int lim = ceil(sqrt(p));
30
31     int cur = 1;
32
33     mp.clear();
34
35     for(int i = 0; i < lim; ++i){
36       mp[(ll)cur * b % p] = i;
37       cur = (ll)cur * a % p;
38     }
39     a = cur, cur = 1;
40     if(!a){
41       if(b)
42         return false;
43       res = k + 1;
44       return true;
45     }
46
47     for(int i = 0; i <= lim; ++i){
48       if(mp.count(cur) && i * lim - mp[cur] >= 0){
49         res = i * lim - mp[cur] + k;
50         return true;
51       }
52       cur = (ll)cur * a % p;
53     }
54     return false;
```

- **卢卡斯**

```
1  ll lucas(ll n, ll m, ll p){
2    if(m == 0)
3      return 1ll;
4    else
5      return C(n % p, m % p, p) * lucas(n / p, m / p) % p;
6  }
```

- **高斯消去**

```
1  bool gauss() {
2    for(int i = 0; i < n; ++i){
3      int r = i;
```

```
 4      for(int j = i + 1; j < n; ++j)
 5        if(arr[r][i] < arr[j][i])
 6          r = j;
 7      if(r != i)
 8        swap(arr[r], arr[i]);
 9
10      if(!arr[i][i])
11        return false;
12
13      int rev = (exgcd(arr[i][i], MOD), (x % MOD + MOD) % MOD);
14
15      for(int k = 0; k < n; ++k){
16        if(k == i)
17          continue;
18        int times = arr[k][i] * rev % MOD;
19        for(int j = i; j < (n << 1); ++j)
20          arr[k][j] = ((arr[k][j] - times * arr[i][j]) % MOD + MOD) % MOD;
21
22      }
23
24      for(int j = i; j < (n << 1); ++j)
25        arr[i][j] = arr[i][j] * rev % MOD;
26    }
27    return true;
28 }
```

6. 动态规划

- **区间覆盖**

```
 1 pii intv[MAXLEN];
 2 int n; //区间数量
 3 int end; //终点
 4
 5
 6 int solve(){
 7     sort(intv, intv + n);
 8     int tmp = 0 /* 暂存 */ , r = 0 /* 右边更新 */;
 9     int sum = 0; //答案
10     int i = 0;
11     while(i < n && r < end){
12         if(intv[i].first > r) break;
13         while(i < n && intv[i].first <= r  && tmp < end){
14             tmp = max(intv[i].second, tmp);
15             ++i;
16         }
17         r = tmp;
18         ++sum;
19     }
20     return r >= end ? sum : 0; // 没有则回传0
21 }
```

- **LIS**

```
 1 int tmp[MAXLEN]; //暂存
 2 int arr[MAXLEN]; //待处理数组
 3 int dp[MAXLEN]; //各个结尾的值
 4
 5 void LIS(){
 6     for(int i = 0; i <= n; ++i) tmp[i] = INT_MAX;
 7     tmp[0] = 0;
 8     for(int i = 0; i < n; ++i){
 9         int pos = lower_bound(tmp, tmp + n, arr[i]) - tmp; //lower_bound找< // upper_bound找<=
10         tmp[pos] = arr[i], dp[i] = pos;
11     }
12 }
```

7. 字串

- **KMP**

```
 1 int f[MAXLEN]; // failure Function 1-index
 2
 3 void kmp(char s[], char p[]){
 4   int i = 0, j = -1;
 5   int n = strlen(s), m = strlen(p);f[0]=-1;
 6   while(i < m){
```

```
7       if(j == -1 || p[i] == p[j]){
8         f[++i] = ++j;
9       }else{
10        j = f[j];
11      }
12  }
13  i = 0; j = 0;
14  while(i <= n){
15    if(j == -1 || s[i] == p[j]){
16      ++j, ++i;
17      if(j == m){
18        //一些操作
19      }
20
21    }else{
22      j = f[j];
23    }
24  }
25 }
```

- **马拉车**

```
1  char s[MAXLEN]; //字串 记得填充$#^(aaa-> $#a#a#a#^)
2  int p[MAXLEN]; //回文长度
3
4  int len = 0 //s的长度
5
6  void manache(){
7    int i_r = 0, c = 0, r = 0; //i_r 反射点，c 中心点 r 右端点
8    for(int i = 1; i < len; ++i){
9      i_r = 2 * c - i;
10     if(r > i) p[i] = min(r - i, p[i_r]);
11     else p[i] = 0;
12     while(s[i + 1 + p[i]] == s[i - 1 - p[i]]) ++p[i];
13     if(i + p[i] > r) c = i, r = i + p[i];
14   }
15   int m = 0; // 长度
16   c = 0; //中央
17   for(int i = 0; i < len - 1; ++i){
18     if(p[i] > m) c = i, m = p[i];
19   }
20   printf("%d\n", m);
21 }
```

- **AC 自动机 + 拓扑优化**

```
1  int cnt = 0;
2  int trie[MAXLEN][26];
3  int fail[MAXLEN];
4  int rd[MAXLEN]; //拓扑必须
5  char c;
6
7
8  void insert(char* s) { //trie插入
9    int id = 0;
10   for (; *s; ++s) {
11     c = *s - 'a';
12     if (!trie[id][c]) trie[id][c] = ++cnt;
13     id = trie[id][c];
14   }
15   //一些操作(在trie上产生节点)
16 }
17
18 void build() {  //记得放在main
19   queue<int> q;
20   for (int i = 0; i < 26; ++i) {
21     if (trie[0][i])q.push(trie[0][i]);
22   }
23   while (!q.empty()) {
24     int u = q.front(); q.pop();
25     for (int i = 0; i < 26; ++i) {
26       if (trie[u][i]) {
27         int t = trie[u][i];
28         fail[t] = trie[fail[u]][i];
29         q.push(t);
30         ++rd[fail[t]]; //拓扑必须
31       }
32       else {
33         trie[u][i] = trie[fail[u]][i];
34       }
```

```
35        }
36     }
37 }
38
39 void query(char* s) {
40    int id = 0;
41    for (int i = 0; s[i]; ++i) {
42       id = trie[id][s[i] - 'a'];
43       //一些懒标操作(复杂度较低)
44       for (int j = id; j; j = fail[j]) {
45          //直接操作
46       }
47    }
48 }
49
50 void topo() {
51    queue<int> q;
52    for (int i = 1; i <= cnt; ++i)if (!rd[i])q.push(i);
53    while (!q.empty()) {
54       int u = q.front(); q.pop();
55       //一些操作(更新当前值)
56       int f = fail[u];
57       --rd[f];
58       //一些操作(懒标回推)
59       if (!rd[f])q.push(f);
60    }
61 }
```

8. 数据结构

- **串列**

```
 1 struct {
 2     int l = 0, r = 0;
 3 } lst[MAXLEN];
 4
 5 void init() {
 6     memset(lst, -1, sizeof(lst));
 7     lst[1].l = 0;
 8     lst[0].r = 1;
 9 }
10
11 void addl(int now, int node) { //加在node的左边
12     lst[now].r = node;
13     lst[lst[node].l].r = now;
14     lst[now].l = lst[node].l;
15     lst[node].l = now;
16 }
17
18 void addr(int now, int node) { //加在node的右边
19     lst[now].l = node;
20     lst[lst[node].r].l = now;
21     lst[now].r = lst[node].r;
22     lst[node].r = now;
23 }
24
25 void remove(int node) {
26     if (!~lst[node].l)return;
27     lst[lst[node].l].r = lst[node].r;
28     lst[lst[node].r].l = lst[node].l;
29     lst[node].l = lst[node].r = -1;
30 }
```

- **稀疏表**

```
 1 int Log2[MAXLEN]; //预处理log2
 2 int st[MAXLEN][20]; //稀疏表
 3 int n, m; //n个节点 m个query
 4
 5 void pre() {
 6   Log2[1] = 0, Log2[2] = 1;
 7   for (int i = 3; i <= MAXLEN; ++i) {
 8     Log2[i] = Log2[i >> 1] + 1;
 9   }
10 }
11
12 void build() {
13   for (int j = 1; j <= 20; ++j) {
14     for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
```

```
15        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
16      }
17    }
18 }
19
20 inline int query(int l, int r) {
21    int lo = Log2[r - l + 1];
22    return min(st[l][lo], st[r - (1 << lo) + 1][lo]);
23 }
```

- **Fenwick**

```
1  #define lowbit(x) (x & -x)
2  int n; //总数量
3  //逆序数对由"大排到小", search(loc - 1)就行了回圈就行了
4
5  namespace fw{
6    int tree[MAXLEN] = {0};
7    void insert(int pos, int val){ //单点插入
8      for(;pos <= n; pos += lowbit(pos))tree[pos] += val;
9    }
10   int search(int pos){ //单点前缀和查询
11     int ans = 0;
12     for(;pos; pos -= lowbit(pos)){
13       ans += tree[pos];
14     }
15     return ans;
16   }
17   int query(int l, int r){  //区间查询
18     return search(r) - search(l - 1);
19   }
20 }
```