代码簿

October 1, 2022

1. 杂项

- **VIMRC**

```
set hls
set is
set cb=unnamed
set ts=4
set sw=4
set si
set nu

inoremap { {}<Left>
inoremap {<CR> {<CR>}<Esc>O
inoremap {{ {
inoremap {} {}
```

- **Merge Sort**

```
int arr[MAXLEN]; // 数组
int tmp[MAXLEN]; // 暂存
int ans  = 0; // 逆序对

void mergeSort(int left, int right){
  if(left >= right) return;
  int mid = left + ((right - left) >> 1);
  mergeSort(left, mid);
  mergeSort(mid + 1, right);
  int l = left;
  int r = mid + 1;
  int idx = 0;
  while(l <= mid && r <= right){
    tmp[idx++] = arr[l] <= arr[r] ? arr[l++]
    : ((ans += mid - l + 1 /* 逆序对 */ ), arr[r++]);
  }
  while(l <= mid){
    tmp[idx++] = arr[l++];
  }
  while(r <= right){
    tmp[idx++] = arr[r++];
  }
  for(int i = 0; i < idx; ++i){
    arr[left + i] = tmp[i];
  }
}
```

- **Disjoint Set**

```
int djs[MAXLEN];
//init for(int i = 0; i < size; ++i)djs[i] = i;

int find(int id){
  return djs[id] == id ? id : djs[id] = find(djs[id]);
}
bool uSet(int a, int b){
  int pa = find(a), pb = find(b);
  return pa == pb ? true : (djs[pa] = pb, false);
}

//with dummy head

int djs[MAXLEN];
//init  for(int i = 0; i < 2 * n; ++i) djs[i] = i; for(int i = 0; i < n; ++i) djs[i] = n + i;

int find(int id){
  return djs[id] == id ? id : djs[id] = find(djs[id]);
```

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

2. 图论

- **Eular Path**

```
int cd[MAXLEN], rd[MAXLEN]; //有向图
vector<pair<int, int>> adjs[MAXLEN]; //first for 点, second fir 边
bool visit[MAXLEN]; //边

int cnt = 0; //判断联通(cnt == size)

void dfs(int id){
    for(auto &adj : adjs){
        if(!visit[adj.second]){
            ++cnt;
            visit[adj.second] = true;
            dfs(adj.first);
            //一些操作
        }
    }
}

bool judge(){
    //无向(边为偶数除了端点)
    int cnt = 0;
    int qidian  = 0; // 起点
    for(int i = 0; i < n; ++i){
        if(adjs[i].size() & 1) ++cnt, qidian = i;
    }
    return (cnt == 2 || cnt == 0);

    //有向((出度 - 入度) = 1 && (入度 - 出度) = 1 || 欧拉环)
    int rdc = 0, cdc = 0;
    int qidian = 0;
    for(int i = 0; i < n; ++i){
        if((cd[i] - rd[i] == 1)) ++cdc;
        else if((rd[i] - cd[i]) == 1) ++rdc, qidian = i;
        else if((rd[i] - cd[i]) != 0) return false;
    }

    return ((cdc == 1 && rdc == 1) || (cdc == 0 && rdc == 0));
}
```

- **拓扑排序**

```
vector<int> adjs[MAXLEN];
int rd[MAXLEN]; //入度
int n; //点的数量
vector<int> ans; //结果

void topSort(){
    queue<int> q;
    int cnt= 0;
    for(int i = 0; i < n; ++i) if(!rd[i]) q.push(i), ++cnt, ans.push_back(i);
    while(!q.empty()){
        int f = q.front(); q.pop();
        for(auto adj : adjs[f]){
            --rd[adj];
            if(!rd[adj]) q.push(adj), ++cnt, ans.push_back(adj);
        }
    }
    if(cnt == n) {}//有找到
    else {} //没找到
}
```

- **多元最短路**

```
int v[MAXLEN][MAXLEN];
//初始化成0x3f3f3f3f, 有边的地方为权重, 自己为0

for(int k = 0; k < n; ++k){
    for(int  i = 0; i < n; ++i){
        for(int  j = 0; j < n; ++j){
            v[i][j] = min(v[i][k] + v[k][j], v[i][j]); //最短路径
```

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

- **单元最短路**

```
vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 边权
int dis[MAXLEN];
bool visit[MAXLEN];

void dij(int id){
    memset(dis, 0x3f, sizeof(dis));
    memset(visit, 0, sizeof(visit));
    dis[id] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> q;

    //first for 目前距离 second for 点
    q.push({0, id});
    while(!q.empty()){
        pair<int, int> front = q.top(); q.pop();
        if(visit[front.second]) continue;
        visit[front.second] = true;
        for(auto adj : adjs[front.second]){
            if((adj.second + dis[front.second]) <  dis[adj.first]){
                dis[adj.first] = adj.second + dis[front.second];
                if(visit[adj.first]) continue;
                q.push({dis[adj.first], adj.first});
            }
        }
    }
}
```

- **找负环**

```
vector<pair<int, int>> adjs[MAXLEN]; //first for 点 second for 权重
int dis[MAXLEN]; //最短路径
int m, n; //n 点数 m 边数

bool bellman(){
    memset(dis, 0x3f, 4 * n);
    for(int c = 0; c < n - 1; ++c){
      for(int i = 0; i < n; ++i){
        for(auto adj : adjs[i]){
          dis[adj.first] = min(dis[adj.first], dis[i] + adj.second);
        }
      }
    }
    bool hasAnswer = false;
  for(int i = 0; i < n; ++i){
    for(auto adj : adjs[i]){
      if(dis[adj.first] > dis[i] + adj.second){
        hasAnswer = true;
        goto ans;
      }
    }
  }
    ans:
    return hasAnswer;
}
```

- **MST**

```
// kruskal

pair<int, pii> adjs[MAXLEN]; //first for 权重 //second.first for 起点 //second.second for 终点
int n; //边的数量
int m; //点的数量
int djs[MAXLEN];


int parent(int i){
    return (djs[i] == i) ? i : djs[i] = parent(djs[i]);
}

void kruskal(){
    for(int i = 0; i <= m; ++i){
        djs[i] = i;
    }

```

```
    sort(adjs, adjs + n);
    int cnt = m; //计算边的数量
    for(int i = 0; i < n; ++i){
        int p1 = parent(adjs[i].second.first), p2 = parent(adj[i].second.second);
        if(p1 == p2) continue;
        djs[p1] = p2;
        //一些操作
        if((--cnt) == 1) break;
    }
}

//Prim 普利姆算法

vector<pii> adjs[MAXLEN]; //first for 点, second for 权重
bool visit[MAXLEN];

int n, m;
void prim(){
  priority_queue<pii, vector<pii>, greater<pii>> q;
  int ans = 0; //最小距离
  q.push({0, 1});
  int cnt = n; //是否联通
  while(!q.empty()){
    pii top = q.top();q.pop();
    if(visit[top.second])continue;
    ans += top.first;
    visit[top.second] = true;
    --cnt;
    for(const pii &adj : adjs[top.second]){
      if(!visit[adj.first]){
        q.push({adj.second, adj.first});
      }
    }
  }
  if(cnt){
    printf("orz\n");
  }else{
    printf("%d\n", ans);
  }

}
```

- **最大流 (Dinic)**

```
struct edge
{
    int to;
    long long val;
    int rev;
    edge(int _to, long long _val, int _rev):to(_to), val(_val), rev(_rev){}
};

int n, m, s, t;


vector<edge> adjs[MAXLEN];

int depth[MAXLEN]; //深度

bool bfs(){ //分层 + 确定没有回去找
    memset(depth, 0, sizeof(depth));
    depth[s] = 1;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int f = q.front(); q.pop();
        for(edge &adj : adjs[f]){
            if(!depth[adj.to] && adj.val){
                depth[adj.to] = depth[f] + 1;
                q.push(adj.to);
            }
        }
    }
    return depth[t];
}

long long dfs(int u = s, long long in = 0x3f3f3f3f3f3f3f3f){ //多路增广
    if(u == t) return in;
    if(in == 0) return 0;
    long long out = 0;
```

```
    for(edge &adj : adjs[u]){
        if((depth[adj.to] == (depth[u] + 1)) && adj.val){
            long long dist = dfs(adj.to, min(in, adj.val));
            adj.val -= dist;
            adjs[adj.to][adj.rev].val += dist;
            in -= dist;
            out += dist;
        }
    }
    if(!out) depth[u] = 0;
    return out;
}

long long ans = 0;

void dinic(){
    while(bfs()){
        while(long long d = dfs()){
            ans += d;
        }
    }
}
```

- **费用流 (Dinic)**

```
struct edge
{
    int to;
    int val;
    int rev;
    int cost;
    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
};

int n, m, s, t;

vector<edge> adjs[MAXLEN];

int dis[MAXLEN];
bool visit[MAXLEN];



long long zuidaliu = 0, zuixiaofeiyong = 0; //最大流and最小费用

bool spfa(int id = t){ //BellMan-Ford
  memset(visit, 0, sizeof(visit));
  memset(dis, 0x3f, sizeof(dis));
  visit[id]  = true;
  dis[id] = 0;
  deque<int> q;
  q.push_front(id);

  while(!q.empty()){
    int front = q.front(); q.pop_front();

    for(edge &adj : adjs[front]){
      if(adjs[adj.to][adj.rev].val > 0 && dis[adj.to]  > dis[front] - adj.cost ){
        dis[adj.to] = dis[front] - adj.cost;
        if(!visit[adj.to]){
          visit[adj.to] = true;
          if(!q.empty() && dis[adj.to] < dis[q.front()]) q.push_front(adj.to); //SLF优化
          else q.push_back(adj.to);
        }
      }
    }
    visit[front] = false;
  }
  return dis[s] != 0x3f3f3f3f;
}

int dfs(int u = s, int flow =  0x3f3f3f3f){ //多路增广
  if(u == t){
    visit[u] = true;
    return flow;
  }
  int in = flow;
  int out = 0;
  visit[u] = true;
```

```
  for(edge &adj : adjs[u]){

    if(adj.val >0 && !visit[adj.to] && dis[adj.to] == (dis[u] - adj.cost)){
      int dis = dfs(adj.to, min(in, adj.val));
      if(dis > 0){
        in -= dis;
        adj.val -= dis;
        out += dis;
        zuixiaofeiyong += dis * adj.cost;
        adjs[adj.to][adj.rev].val += dis;
      }
      if(!in) break;

    }
  }

  return out;
}



void dinic(){
  while(spfa()){
    while(int d = dfs()){
      memset(visit, 0, sizeof(visit));
      zuidaliu += d;
    }
  }
}
```

3. 树问题

- **树直径**

```
//DFS两次

int d[MAXN];
vector<int> adjs[MAXN];

int last; //最远点，  d[last] 为数的直径 不可做负边

int bfs(int u, int p){
    queue<pair<int, int>> q;
    q.push({u, p});
    while(!q.empty()){
        auto [u, p] = q.front(); q.pop();
        for(const int adj : adjs[u]){
            if(adj == p)
                continue;
            d[adj] = d[u] + 1;
            last = adj;
            q.push({adj, u});
        }
    }
}

int farPoint(int u, int p){
    last = u;
    bfs(u, p);
    d[last] = 0;
    bfs(last, p);
    return d[last];
}

//树形DP，  可做负边

int n, d = 0;
int d1[N], d2[N];
vector<int> adjs[N];

void dfs(int u, int p) {
  d1[u] = d2[u] = 0;
  for (int v : adjs[u]) {
    if (v == p)
        continue;
    dfs(v, u);
    int t = d1[v] + 1; //可改权重
    if (t > d1[u])
      d2[u] = d1[u], d1[u] = t;
```

```
    else if (t > d2[u])
      d2[u] = t;
  }
  d = max(d, d1[u] + d2[u]);
}
```

- **LCA**

```
//倍增

int dep[MAXN];
int pa[MAXN][32];
vector<int> ch[MAXN];


void dfs(int u, int p){ // Pre-process
    dep[u] = dep[p] + 1;
    pa[u][0] = p;

    for(int i = 1; i < 32; ++i)
        pa[u][i] = pa[pa[u][i - 1]][i - 1];

    for(const int v : ch[u]){
        if(v == p)
            continue;
        dfs(v, u);
    }
}

int lca(int a, int b){
    if(dep[a] > dep[b])
        swap(a, b);

    int diff = dep[b] - dep[a];
    int i = 0;

    //调整高度
    for(; diff; ++i, diff >>= 1)
        if(diff & 1)
            b = pa[b][i];

    if(a == b)
        return a;

    i = 0;

    for(i = 31; i >= 0; --i)
        if(pa[a][i] != pa[b][i])
            a = pa[a][i], b = pa[b][i];

    return pa[a][0];

}
```

4. 计算几何

- **凸包**

```
//Andrew


// 点，上凸包，下凸包

using pdd = pair<double, double>
pdd ps[MAXN], u[MAXN], d[MAXN];

int ui = 0, di = 0;

inline double cross(const pdd &p1, const pdd &p2, const pdd &p3){
    return (p2.first - p1.first) * (p3.second - p1.second) - (p2.second - p1.second) * (p3.first - p1.
    first);
}


void andrew(){
    sort(ps, ps + n);
    for(int i = 0; i < n; ++i){
        while(ui >= 2 && cross(u[ui - 2], u[ui - 1], ps[i]) <= 0)
            --ui;
```

```
        while(di >= 2 && cross(d[di - 2], d[di - 1], ps[i]) >= 0)
            --di;

        u[ui++] = ps[i];
        d[di++] = ps[i];
    }

}
```

5. 数论

- **Miller Robin(确认素数)**

```
// allow condition 1, 1, 1, 1, .....
//                 X, X, X, X, X, p - 1, 1, 1, 1, ....
//                 p - 1, 1, 1, 1, 1, 1, .....
//                 0 (cannot check) multiply of p

bool is_prime(ll p){
    if(p < 3 || p % 2 == 0)
        return p == 2;
    ll u = p - 1;
    int t = 0;
    while(!(u & 1))
        u >>= 1, ++t;

    const static ll ud[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

    for(const ll a : ud){
        ll v = qpow(a, u, p);
        if(v <= 1 || v == p - 1)
            continue;

        for(int j = 1; j <= t; ++j){
            v = qmul(v, v, p);
            if(v == p - 1 && j != t){
                v = 1;
                break;
            }
            if(v == 1)
                return false;
        }
        if(v != 1)
            return false;
    }
    return true;
}
```

- **Pollard-Rho(确认因数)**

```
// remember srand(time(NULL))

ll PR(ll n){
    if(n == 4)
        return 2;

    if(is_prime(n))
        return n;

    while(1){
        ll c = (rand() % (n - 1)) + 1;
        auto f = [&](const ll x){
            return ((lll) x * x + c) % n;
        };
        ll t = 0, r = 0, p = 1, q;
        do{
            for(int i = 0; i < 128; ++i){
                t = f(t), r = f(f(r));
                if(t == r || (q = (lll) p * abs(t - r)  % n) == 0)
                    break;
                p = q;
            }
            ll d = __gcd(p, n);
            if(d > 1)
                return d;
        }while(t != r);
    }
}
```

- **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;

```

```
    for(int i = 2; i <= n; ++i){
        if(!vis[i])
            primes[tot++] = i, phi[i] = i - 1;
        for(int j = 0; j < tot && i * primes[j] <= n; ++j){
            vis[i * primes[j]]  = true;
            if(i % primes[j])
                phi[i * primes[j]] = phi[i] * (primes[j] - 1);
            else{
                phi[i * primes[j]] = primes[j] * phi[i];
                break;
            }
        }
    }
}
```

- **扩展欧几里得**

```
// ax+by=gcd(a, b) if gcd(a, b) == 1
// ax同余c(mod b) <=> ax + by = c
// c 同余 a / b (mod M) = bx 同余 a (mod M)

ll Exgcd(ll a, ll b, ll &x, ll &y){
    if(!b)
        return x = 1, y = 0, a;
    ll d = Exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b) * y;
    return d;
}

//线性

void inv(){
    inv[1] = 1;
    for(int i = 2; i <= n; ++i)
      inv[i] = (ll)(p - p / i) * inv[p % i] % p;
}


//随机线性

int arr[n + 1], ji[n + 1], rev[n + 1], inv[n + 1]; // 数组, 前缀积, 前缀积逆元

ji[0] = 1;

for(int i = 1; i <= n; ++i)
    ji[i] = ji[i - 1] * arr[i] % p;

rev[n] = (exgcd(ji[n], p), x);

for(int i = n; i >= 1; --i)
    rev[i - 1] = (ll)rev[i] * arr[i] % p;

for(int i = 1; i <= n; ++i)
    inv[i] = (ll)rev[i] * ji[i - 1] % p;
```

- **中国剩余定理**

```
// r互素

int a[MAXN], r[MAXN];
int n = 1, ans = 0;


for(int i = 0; i < k; ++i)
  n *= r[i];

for(int i = 0; i < k; ++i){
  ll m = n / r[i];
  exgcd(m, r[i]);
  ans = (ans + ((((lll)a[i] * m) % n) * x % n)) % n;
}


// r非互素

ll  M = r[0],  ans = a[0];

for(int i = 1; i < k; ++i){
```

```
  ll da = (((a[i] - ans) % r[i]) + r[i]) % r[i];
  ll g = exgcd(M, r[i]);
  if(da % g)
    return -1;
  auto tmp = (lll)M * x * (da / g);
  M = M * (r[i] / g);
  ans = (((ans + tmp) % M) + M) % M;
}
```

- **exBSGS**

```
// a^x 同于 b (mod p)

int a, b, p, res;
unordered_map<int, int> mp;

bool bsgs(){
    a %= p, b %= p;
    if(b == 1 || p == 1){
      res = 0;
      return true;
    }
    ll ax =1;
    int k = 0, g;
    while((g = exgcd(a, p)) != 1){
      if(b % g)
        return false;
      ++k;
      b /= g, p /= g;
      ax = ((ll)ax * (a / g)) % p;
      if(ax == b){
        res = k;
        return true;
      }
    }

    exgcd(ax , p);
    b = ((ll)b * (((x % p) + p) % p)) % p;

    int lim = ceil(sqrt(p));

    int cur = 1;

    mp.clear();

    for(int i = 0; i < lim; ++i){
      mp[(ll)cur * b % p] = i;
      cur = (ll)cur * a % p;
    }
    a = cur, cur = 1;
    if(!a){
      if(b)
        return false;
      res = k + 1;
      return true;
    }

    for(int i = 0; i <= lim; ++i){
      if(mp.count(cur) && i * lim - mp[cur] >= 0){
        res = i * lim - mp[cur] + k;
        return true;
      }
      cur = (ll)cur * a % p;
    }
    return false;
```

- **卢卡斯**

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

- **高斯消去**

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

```
    for(int j = i + 1; j < n; ++j)
      if(arr[r][i] < arr[j][i])
        r = j;
    if(r != i)
      swap(arr[r], arr[i]);

    if(!arr[i][i])
      return false;

    int rev = (exgcd(arr[i][i], MOD), (x % MOD + MOD) % MOD);

    for(int k = 0; k < n; ++k){
      if(k == i)
        continue;
      int times = arr[k][i] * rev % MOD;
      for(int j = i; j < (n << 1); ++j)
        arr[k][j] = ((arr[k][j] - times * arr[i][j]) % MOD + MOD) % MOD;

    }

    for(int j = i; j < (n << 1); ++j)
      arr[i][j] = arr[i][j] * rev % MOD;
  }
  return true;
}
```

- **线性基**

```
// XOR 最大值

ll p[x];

bool insert(ll x) {
    for(int i = 63; i + 1; --i) {
        if(!(x & (1ll << i)))
            continue;

        if(!p[i]) {
            p[i] = x;
            return true;
        }
        x^= p[i];
    }
    return false;
}
```

6. 动态规划

- **区间覆盖**

```
pii intv[MAXLEN];
int n; //区间数量
int end; //终点


int solve(){
    sort(intv, intv + n);
    int tmp = 0 /* 暂存 */ , r = 0 /* 右边更新 */;
    int sum = 0; //答案
    int i = 0;
    while(i < n && r < end){
        if(intv[i].first > r) break;
        while(i < n && intv[i].first <= r  && tmp < end){
            tmp = max(intv[i].second, tmp);
            ++i;
        }
        r = tmp;
        ++sum;
    }
    return r >= end : sum ? 0; // 没有则回传0
}
```

- **LIS**

```
int tmp[MAXLEN]; //暂存
int arr[MAXLEN]; //待处理数组
int dp[MAXLEN]; //各个结尾的值

void LIS(){
```

```
    for(int i = 0; i <= n; ++i) tmp[i] = INT_MAX;
    tmp[0] = 0;
    for(int i = 0; i < n; ++i){
        int pos = lower_bound(tmp, tmp + n, arr[i]) - tmp; //lower_bound找< // upper_bound找<=
        tmp[pos] = arr[i], dp[i] = pos;
    }
}
```

7. 字串

- **KMP**

```
int f[MAXLEN]; // failure Function 1-index

void kmp(char s[], char p[]){
  int i = 0, j = -1;
  int n = strlen(s), m = strlen(p);f[0]=-1;
  while(i < m){
    if(j == -1 || p[i] == p[j]){
      f[++i] = ++j;
    }else{
      j = f[j];
    }
  }
  i = 0; j = 0;
  while(i <= n){
    if(j == -1 || s[i] == p[j]){
      ++j, ++i;
      if(j == m){
        //一些操作
      }

    }else{
      j = f[j];
    }
  }
}
```

- **马拉车**

```
char s[MAXLEN]; //字串 记得填充$#^(aaa-> $#a#a#a#^)
int p[MAXLEN]; //回文长度

int len = 0 //s的长度

void manache(){
  int i_r = 0, c = 0, r = 0; //i_r 反射点, c 中心点 r 右端点
  for(int i = 1; i < len; ++i){
    i_r = 2 * c - i;
    if(r > i) p[i] = min(r - i, p[i_r]);
    else p[i] = 0;
    while(s[i + 1 + p[i]] == s[i - 1 - p[i]]) ++p[i];
    if(i + p[i] > r) c = i, r = i + p[i];
  }
  int m = 0; // 长度
  c = 0; //中央
  for(int i = 0; i < len - 1; ++i){
    if(p[i] > m) c = i, m = p[i];
  }
  printf("%d\n", m);
}
```

- **AC 自动机 + 拓扑优化**

```
int cnt = 0;
int trie[MAXLEN][26];
int fail[MAXLEN];
int rd[MAXLEN]; //拓扑必须
char c;


void insert(char* s) { //trie插入
  int id = 0;
  for (; *s; ++s) {
    c = *s - 'a';
    if (!trie[id][c]) trie[id][c] = ++cnt;
    id = trie[id][c];
  }
  //一些操作(在trie上产生节点)
```

```
}

void build() {  //记得放在main
  queue<int> q;
  for (int i = 0; i < 26; ++i) {
    if (trie[0][i])q.push(trie[0][i]);
  }
  while (!q.empty()) {
    int u = q.front(); q.pop();
    for (int i = 0; i < 26; ++i) {
      if (trie[u][i]) {
        int t = trie[u][i];
        fail[t] = trie[fail[u]][i];
        q.push(t);
        ++rd[fail[t]]; //拓扑必须
      }
      else {
        trie[u][i] = trie[fail[u]][i];
      }
    }
  }
}

void query(char* s) {
  int id = 0;
  for (int i = 0; s[i]; ++i) {
    id = trie[id][s[i] - 'a'];
    //一些懒标操作(复杂度较低)
    for (int j = id; j; j = fail[j]) {
      //直接操作
    }
  }
}

void topo() {
  queue<int> q;
  for (int i = 1; i <= cnt; ++i)if (!rd[i])q.push(i);
  while (!q.empty()) {
    int u = q.front(); q.pop();
    //一些操作(更新当前值)
    int f = fail[u];
    --rd[f];
    //一些操作(懒标回推)
    if (!rd[f])q.push(f);
  }
}
```

8. 数据结构

- **串列**

```
struct {
    int l = 0, r = 0;
} lst[MAXLEN];

void init() {
    memset(lst, -1, sizeof(lst));
    lst[1].l = 0;
    lst[0].r = 1;
}

void addl(int now, int node) { //加在node的左边
    lst[now].r = node;
    lst[lst[node].l].r = now;
    lst[now].l = lst[node].l;
    lst[node].l = now;
}

void addr(int now, int node) { //加在node的右边
    lst[now].l = node;
    lst[lst[node].r].l = now;
    lst[now].r = lst[node].r;
    lst[node].r = now;
}

void remove(int node) {
    if (!~lst[node].l)return;
    lst[lst[node].l].r = lst[node].r;
    lst[lst[node].r].l = lst[node].l;
```

```
    lst[node].l = lst[node].r = -1;
}
```

- **稀疏表**

```
int Log2[MAXLEN]; //预处理log2
int st[MAXLEN][20]; //稀疏表
int n, m; //n个节点 m个query

void pre() {
  Log2[1] = 0, Log2[2] = 1;
  for (int i = 3; i <= MAXLEN; ++i) {
    Log2[i] = Log2[i >> 1] + 1;
  }
}

void build() {
  for (int j = 1; j <= 20; ++j) {
    for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
      st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
}

inline int query(int l, int r) {
  int lo = Log2[r - l + 1];
  return min(st[l][lo], st[r - (1 << lo) + 1][lo]);
}
```

- **Fenwick**

```
#define lowbit(x) (x & -x)
int n; //总数量
//逆序数对由"大排到小", search(loc - 1)就行了回圈就行了

namespace fw{
  int tree[MAXLEN] = {0};
  void insert(int pos, int val){ //单点插入
    for(;pos <= n; pos += lowbit(pos))tree[pos] += val;
  }
  int search(int pos){ //单点前缀和查询
    int ans = 0;
    for(;pos; pos -= lowbit(pos)){
      ans += tree[pos];
    }
    return ans;
  }
  int query(int l, int r){  //区间查询
    return search(r) - search(l - 1);
  }
}
```

- **线段树**

```
ll vals[MAXN << 2] = {0};
ll tag[MAXN << 2] = {0};
ll arr[MAXN];


void build(int u, int l, int r) {
  if(l == r)
    return void(vals[u] = arr[l]);
  int mid = l + (r - l) / 2;

  build(u << 1, l, mid);
  build(u << 1 | 1, mid + 1, r);
  vals[u] = vals[u << 1] + vals[u << 1 | 1];
}

void push_down(int u, int l, int r) { //拿parent的tag来更新
  vals[u] += tag[u >> 1] * (r - l + 1);
  tag[u] += tag[u >> 1];
}

void insert(int u, int l, int r, int ul, int ur, int val) {

  if(l >= ul && r <= ur)
    return void((vals[u] += (r - l + 1) * val, tag[u] += val)); //标记添加在当前点上

  int mid = l + (r - l) / 2;
```

```
27
28    push_down(u << 1, l, mid);
29    push_down(u << 1 | 1, mid + 1, r);
30
31    tag[u] = 0;
32
33    if(ul <= mid)
34      insert(u << 1, l, mid, ul, ur, val);
35    if(mid < ur)
36      insert(u << 1 | 1, mid + 1, r, ul, ur, val);
37
38    vals[u] = vals[u << 1] + vals[u << 1 | 1]; //用child的值来算加总
39  }
40
41  ll query(int u, int l, int r, int ul, int ur) {
42    if(l >= ul && r <= ur)
43      return vals[u];
44
45    int mid = l + (r - l) / 2;
46    push_down(u << 1, l, mid);
47    push_down(u << 1 | 1, mid + 1, r);
48    tag[u] = 0;
49
50    ll sum = 0;
51
52    if(ul <= mid)
53      sum += query(u << 1, l, mid, ul, ur);
54    if(mid < ur)
55      sum += query(u << 1 | 1, mid + 1, r, ul, ur);
56    return sum;
57  }
```