1 Data Structure

1.1 Segment Tree

```cpp
struct SegT{
  const int MAXN = 1e5+5;
  int tr[MAXN*4], arr[MAXN], tag[MAXN*4];

  int combine(int a, int b){
      return max(a,b);
  }

  void build(int idx, int l, int r){
      if(l==r){
          tr[idx] = arr[l];
      }else{
          int m = (l+r)/2;
          build(idx*2,l,m);
          build(idx*2+1,m+1,r);
          tr[idx] = combine(tr[idx*2],tr[idx*2+1]);
      }
  }

  void push(int idx){
      if(tag[idx]){
          tr[idx<<1] = max(tr[idx<<1], tag[idx]);
          tr[idx<<1|1] = max(tr[idx<<1|1], tag[idx]);
          tag[idx<<1] = max(tag[idx<<1], tag[idx]);
          tag[idx<<1|1] = max(tag[idx<<1|1], tag[idx]);
          tag[idx] = 0;
      }
  }

  void modify(int ql, int qr, int val, int idx, int l, int r){
      if(l!=r) push(idx); //當節點並非葉節點時，下推標記
      if(ql <= l && r <= qr){
          tr[idx] = max(tr[idx],val);
          tag[idx] = max(tag[idx],val);
          return;
      }
      int m = (l+r)/2;
      if(qr > m) modify(ql, qr, val, idx*2+1, m+1, r);
      if(ql <= m) modify(ql, qr, val, idx*2, l, m);
      tr[idx] = combine(tr[idx<<1],tr[idx<<1|1]);
  }

  int query(int ql, int qr, int idx, int l, int r){
      if(l!=r) push(idx);
      if(ql <= l && r <= qr){
          return tr[idx];
      }
      int m = (l+r)/2;
      if(ql > m){
          return query(ql, qr, idx*2+1, m+1, r);
      }
      if(qr <= m){
          return query(ql, qr, idx*2, l, m);
      }
      return combine(query(ql, qr, idx*2, l, m), query(ql, qr, idx*2+1, m+1, r));
  }
};
```

1.2 Treap

```cpp
struct Treap{
  Treap *l, *r;
  int val, size, sum;
  Treap(int v): l(nullptr), r(nullptr), val(v), size(1), sum(v){}
  void pull();
};

void Treap::pull(){
  size = 1, sum = val;
  if(l!=nullptr) size += l->size, sum += l->sum;
  if(r!=nullptr) size += r->size, sum += r->sum;
}

int sz(Treap *t){
  return (t==nullptr ? 0 : t->size);
}

Treap *merge(Treap *a, Treap *b){
  if(a==nullptr) return b;
  if(b==nullptr) return a;
  if(rand()%(a->size+b->size) <a->size){
    a->r = merge(a->r,b);
    a->pull();
    return a;
  }else{
    b->l = merge(a,b->l);
    b->pull();
    return b;
  }
}

void split(Treap *t, Treap *&a, Treap *&b, int k){
  if(t==nullptr){
    a = b = nullptr;
    return;
  }
  if(sz(t->l) < k){
    a = t;
    split(t->r,a->r,b,k-sz(t->l)-1);
    a->pull();
  }else{
    b = t;
    split(t->l,a,b->l,k);
    b->pull();
  }
}
```

2 Flow

2.1 Dinic

```cpp
struct Dinic {
    const int INF = 1e18;
    struct edge {
        int u, v, cap, flow;
        edge(int u, int v, int cap): u(u), v(v), cap(cap), flow(0) {}
    };
    vector<edge> edges;
    vector<vector<int>> adj;
    vector<int> level, num;
    queue<int> q;
    int n, s, t, cnt = 0; //To maintain the id of edges

    void init(int nn, int ss, int tt) {
        n = nn + 1, s = ss, t = tt;
        adj.resize(n);
        level.resize(n);
        num.resize(n);
    }

    void add_edge(int u, int v, int cap) {
        edges.push_back({u, v, cap});
        edges.push_back({v, u, 0});
        adj[u].push_back(cnt++);
        adj[v].push_back(cnt++);
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);

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

            for (auto eid : adj[u]) {
                edge e = edges[eid];

                //We only pass the edge that has positive capacity
                if (e.cap - e.flow <= 0 || level[e.v] != -1)
                    continue;

                level[e.v] = level[u] + 1;
                q.push(e.v);
            }
        }

        //If we cannot reach t, then there is no Augmenting Path
        return level[t] != -1;
    }

    int dfs(int u, int now) {
        if (now == 0)
            return 0;

        if (u == t)
            return now;

        for (; num[u] < adj[u].size(); num[u]++) {
            edge e = edges[adj[u][num[u]]];

            if (level[e.v] != level[u] + 1 || e.cap - e.flow <= 0)
                continue;

            int f = dfs(e.v, min(now, e.cap - e.flow));

            if (!f)
                continue;

            edges[adj[u][num[u]]].flow += f;
            edges[adj[u][num[u]] ^ 1].flow -= f;
            return f;
        }

        return 0;
    }

    int get_flow() {
        int res = 0, now;

        while (true) {
            if (!bfs())
                break;

            fill(num.begin(), num.end(), 0);

            while (now = dfs(s, INF)) {
                res += now;
            }
        }

        return res;
    }
} flow;
```

2.2 mcmf

```cpp
struct MCMF {
    static const int N = 500, INF = 1e18;
    int n, s, t;
    struct Edge {
        int v, cap, f, cost;
        Edge(int v, int cap, int f, int cost): v(v), cap(cap), f(f), cost(cost) {}
    };
    vector<Edge> edges;
```

```
    vector<vector<int>> adj;

    void init(int n_, int s_, int t_) {
        n = n_ + 1, s = s_, t = t_;
        adj.resize(n);
    }

    void add_edge(int u, int v, int cap, int cost) {
        adj[u].push_back(edges.size());
        edges.push_back({v, cap, 0, cost});
        adj[v].push_back(edges.size());
        edges.push_back({u, 0, 0, -cost});
    }

    int dis[N], p[N], pe[N], inque[N];

    void SPFA() {
        for (int i = 0; i <= n; i++)
            dis[i] = INF, p[i] = -1, pe[i] = -1, inque[i] = 0;

        dis[s] = 0;
        queue<int> q;
        q.push(s);
        inque[s] = 1;

        while (!q.empty()) {
            int u = q.front();
            q.pop();
            inque[u] = false;

            for (auto x : adj[u]) {
                auto e = edges[x];

                if (e.cap > 0 && dis[e.v] > dis[u] + e.cost) {
                    dis[e.v] = dis[u] + e.cost;
                    p[e.v] = u;
                    pe[e.v] = x;

                    if (!inque[e.v])
                        inque[e.v] = true, q.push(e.v);
                }
            }
        }
    }

    pair<int, int> min_cost() {
        int flow = 0, cost = 0;

        while (true) {
            SPFA();

            if (dis[t] == INF)
                break;

            int min_flow = INF;

            int now = t;

            while (now != s) {
                min_flow = min(min_flow, edges[pe[now]].cap);
                now = p[now];
            };

            flow += min_flow, cost += min_flow * dis[t];

            now = t;

            while (now != s) {
                edges[pe[now]].cap -= min_flow;
                edges[pe[now] ^ 1].cap += min_flow;
                now = p[now];
            }
        }

        return {flow, cost};
    }
} flow;
```

3 Graphs

3.1 BCC_Edge

```
void dfs(int u, int a){
  d[u] = l[u] = t++;
  st.emplace(u);
  for(int v : adj[u]){
    if(v == a) continue;
    if(!d[v]){
      dfs(v,u);
      //if(d[u] < l[v]) bridges.emplace_back(u,v);
      l[u] = min(l[u],l[v]);
    }
    l[u] = min(l[u],d[v]);
  }
  if(l[u]==d[u]){
    bccid++;
    int tmp;
    do{
      tmp = st.top(); st.pop();
      bcc[tmp] = bccid;
    }while(tmp!=u);
  }
}
```

3.2 BCC_Vertex

```
const int N = 2e5+5;
vector<int> adj[N], adj2[N], bccids[N];

int low[N], dfn[N], bccid[N], bcccnt, w[N], cutbcc[N], cutid[N];
int cut[N], inst[N], t, tt;

void dfs(int u, int par){
  low[u] = dfn[u] = ++t;
  int cnt = 0;
  st.push(u);
  for(auto v : adj[u]){
    if(v == par) continue;
    if(!dfn[v]){
      dfs(v,u); cnt++;
      low[u] = min(low[u],low[v]);
      if(low[v] >= dfn[u]){
        cut[u] = 1;
        ++bcccnt;
        int x;
        do{
          if(st.empty()) break;
          x = st.top(); st.pop();
          //if(bccid[x]) bcc[bccid[x]].erase(bcc[bccid[x]].find(w[x]));
          bccids[x].push_back(bcccnt);
          bccid[x] = bcccnt;
        }while(x!=v);
        //if(bccid[u]) bcc[bccid[u]].erase(bcc[bccid[u]].find(w[u]));
        bccids[u].push_back(bcccnt);
        bccid[u] = bcccnt;
      }
    }else if(dfn[v] < dfn[u]){
      low[u] = min(low[u],dfn[v]);
    }
  }
  if(par==-1&&cnt < 2) cut[u] = 0;
}
```

3.3 dijkstra

```
priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int,int>>> pq;
pq.push({0,s});
dis[s] = 0;
inq[s] = 1;
while(!pq.empty()){
  auto [ww,u] = pq.top(); pq.pop();
  inq[u] = 0;
  for(auto [v,w] : adj[u]){
    if(dis[v] > dis[u]+ w){
      dis[v] = dis[u]+w;
      if(!inq[v]){
        pq.push({dis[v],v});
        inq[v] = 1;
      }
    }
  }
}
```

3.4 SCC_korasaju

```
const int N = 1e6+5;
vector<int> adj[N], adj2[N];
int vis[N], scc[N], id;
stack<int> st;

void dfs1(int u){
  vis[u] = true;
  for(auto v : adj[u]){
    if(!vis[v]) dfs1(v);
  }
  st.push(u);
}

void dfs2(int u){
  scc[u] = id;
  for(auto v : adj2[u]){
    if(!scc[v]) dfs2(v);
  }
}
```

3.5 SCC_tarjan

```
const int N = 1e6+5;
vector<int> adj[N];
int dfn[N], low[N], inst[N], scc[N], sccid = 0, cnt = 0;
stack<int> st;

void dfs(int u){
  dfn[u] = low[u] = ++cnt;
  st.push(u);
  inst[u] = 1;
  for(auto v : adj[u]){
    if(!dfn[v]){
      dfs(v);
      low[u] = min(low[u],low[v]);
    }else if(inst[v]){
      low[u] = min(low[u],dfn[v]);
    }
  }
  if(low[u]==dfn[u]){
    int x;
    do{
      x = st.top();
      st.pop();
      scc[x] = sccid;
      inst[x] = 0;
    }while(x!=u);
    sccid++;
  }
}
```

4 Number Theory

4.1 FFT

```
typedef complex<double> cp;

const double pi = acos(-1);
const int NN = 131072;

struct FastFourierTransform{
  /*
```

```
    Iterative Fast Fourier Transform

    How this works? Look at this

    0th recursion 0(000)   1(001)   2(010)
          3(011)   4(100)   5(101)   6(110)
          7(111)
    1th recursion 0(000)   2(010)   4(100)
          6(110) | 1(011)   3(011)   5(101)
          7(111)
    2th recursion 0(000)   4(100) | 2(010)
          6(110) | 1(011)   5(101) | 3(011)
          7(111)
    3th recursion 0(000) | 4(100) | 2(010) |
         6(110) | 1(011) | 5(101) | 3(011) |
         7(111)

    All the bits are reversed => We can save
         the reverse of the numbers in an
         array!
*/
int n, rev[NN];
cp omega[NN], iomega[NN];
void init(int n_){
  n = n_;
  for(int i = 0;i < n_;i++){
    //Calculate the nth roots of unity
    omega[i] = cp(cos(2*pi*i/n_),sin(2*pi*
        i/n_));
    iomega[i] = conj(omega[i]);
  }
  int k = __lg(n_);
  for(int i = 0;i < n_;i++){
    int t = 0;
    for(int j = 0;j < k;j++){
      if(i & (1<<j)) t |= (1<<(k-j-1));
    }
    rev[i] = t;
  }
}

void transform(vector<cp> &a, cp* xomega){
  for(int i = 0;i < n;i++)
    if(i < rev[i]) swap(a[i],a[rev[i]]);
  for(int len = 2; len <= n; len <<= 1){
    int mid = len >> 1;
    int r = n/len;
    for(int j = 0;j < n;j += len)
      for(int i = 0;i < mid;i++){
        cp tmp = xomega[r*i] * a[j+mid+i];
        a[j+mid+i] = a[j+i] - tmp;
        a[j+i] = a[j+i] + tmp;
      }
  }
}

void fft(vector<cp> &a){ transform(a,omega
    ); }
void ifft(vector<cp> &a){ transform(a,
    iomega); for(int i = 0;i < n;i++) a[i]
     /= n;}
} FFT;
```

4.2 Linear_Sieve

```
const int N = 1e5+5;
int k, lpf[N];

vector<int> primes;
void init(){
    fill(lpf,lpf+N,1);
    for(int i = 2;i < N;i++){
        if(lpf[i]==1){
            lpf[i] = i;
            primes.push_back(i);
        }
        for(int x : primes){
            if(x*i > N) break;
            lpf[x*i]=x;
            if(x==lpf[i]) break;
        }
    }
}
```

4.3 NTT

```
const int N = 1e5+5, MOD = 998244353, G = 3;

int fastpow(int n, int p){
    int res = 1;
    while(p){
        if(p&1) res = res * n % MOD;
        n = n * n % MOD;
        p >>= 1;
    }
    return res;
}

struct NTT{
    int n, inv, rev[N];
    int omega[N], iomega[N];
    void init(int n_){
        n = 1;
        while(n < n_) n<<=1;
        inv = fastpow(n,MOD-2);
        int k = __lg(n);
        int x = fastpow(G, (MOD-1)/n);
        omega[0] = 1;
        for(int i = 1;i < n;i++)
            omega[i] = omega[i-1] * x % MOD;
        iomega[n-1] = fastpow(omega[n-1],MOD
            -2);
        for(int i = n-2; i >= 0; i--)
            iomega[i] = iomega[i+1] * x %
                MOD;
        for(int i = 0;i < n;i++){
            int t = 0;
            for(int j = 0;j < k;j++)
                if(i&(1<<j)) t |= (1<<k-j-1)
                    ;
            rev[i] = t;
        }
    }
    void transform(int *a, int *xomega){
        for(int i = 0;i < n;i++)
            if(i < rev[i]) swap(a[i],a[rev[i
                ]]);
        for(int len = 2;len <= n;len <<= 1){
            int mid = len>>1;
            int r = n/len;
            for(int j = 0;j < n;j += len){
                for(int i = 0;i < mid;i++){
                    int tmp = xomega[r*i] *
                        a[j+mid+i] % MOD;
                    a[j+mid+i] = (a[j+i] -
                        tmp + MOD) % MOD;
                    a[j+i] = (a[j+i]+tmp)%
                        MOD;
                }
            }
        }
    }
    void dft(int *a){transform(a,omega);}
    void idft(int *a){transform(a,iomega);
        for(int i = 0;i < n;i++) a[i] = a[i
        ]*inv %MOD;}

    int tmp[8][N];

    void copy_(int *a, int *b, int m){
        for(int i = 0;i < m;i++)
            a[i] = b[i];
        for(int i = m;i < n;i++)
            a[i] = 0;
    }

    void copy(int *a, int *b, int m){
        for(int i = 0;i < m;i++)
            a[i] = b[i];
    }

    //B_{k+1} = B_k(2-AB_k) (mod MOD)
    void inverse(int *a, int *b, int m){
        //Uses tmp[0], tmp[1]
        if(m==1){
            b[0] = fastpow(a[0],MOD-2);
            return;
        }
        inverse(a,b,m>>1);
        init(m<<1);
        copy_(tmp[0],a,m); copy_(tmp[1],b,m
            >>1);
        dft(tmp[0]); dft(tmp[1]);
        for(int i = 0;i < n;i++) tmp[0][i] =
            tmp[1][i]*(2-tmp[0][i]*tmp[1][i
            ]%MOD+MOD)%MOD;
        idft(tmp[0]);
        copy(b,tmp[0],m);
    }


    //Q_{k+1} = pow(2,MOD-2)(Q_k + P*pow(Q_k
        ,MOD-2)) (mod MOD)
    void sqrt(int *a, int *b, int m){
        //Uses tmp[2], tmp[3]
        if(m==1){
            b[0] = 1;
            return;
        }
        sqrt(a,b,m>>1);
        for(int i = m;i < m<<1;i++)
            b[i] = 0;
        inverse(b,tmp[2],m);
        init(m<<1);
        for(int i = m;i < m<<1;i++)
            b[i] = tmp[2][i] = 0;
        int inv2 = fastpow(2,MOD-2);
        copy_(tmp[3],a,m);
        dft(tmp[3]); dft(tmp[2]);
        for(int i = 0;i < n;i++)
            tmp[3][i] = tmp[3][i]*tmp[2][i]%
                MOD;
        idft(tmp[3]);
        for(int i = 0;i < m;i++)
            b[i] = (b[i]+tmp[3][i])%MOD*inv2
                %MOD;
    }

    void derivative(int *a, int *b, int m){
        for(int i = 1;i < m;i++) b[i-1] = a[
            i]*i%MOD;
        b[m-1] = 0;
    }

    void integral(int *a, int *b, int m){
        for(int i = m-1;i;i--) b[i] = a[i
            -1]*fastpow(i,MOD-2)%MOD;
        b[0] = 0;
    }

    void ln(int *a, int *b, int m){
        //Uses tmp[4], tmp[5]
        inverse(a,b,m);
        derivative(a,tmp[5],m);

        init(m<<1);
        copy_(tmp[4],b,m), copy_(tmp[5],tmp
            [5],m);
        dft(tmp[4]), dft(tmp[5]);
        for(int i = 0;i < m<<1;i++) tmp[4][i
            ] = tmp[4][i]*tmp[5][i]%MOD;
        idft(tmp[4]);
        integral(tmp[4],b,m);
    }

    void exp(int *a, int *b, int m){
        //Uses tmp[6], tmp[7]
        b[0] = 1;
        for(int i = 4,j = 2;j <= m;j = i, i
            <<=1){
            ln(b,tmp[6],j);
            tmp[6][0] = (a[0]+1-tmp[6][0]+
                MOD)%MOD;
            for(int k = 1; k < j;k++) tmp
                [6][k] = (a[k]-tmp[6][k]+MOD
                )%MOD;

            fill(tmp[6]+j,tmp[6]+i,0);
            dft(b), dft(tmp[6]);
            for(int k = 0; k < i;k++) b[k] =
                b[k]*tmp[6][k]%MOD;
            idft(b);
            fill(b+j,b+i,0);
        }
    }
} NTT;
```

5 String

5.1 AC_Automaton

```cpp
const int N = 1e3+5, MOD = 1e9+9;

int dp[N][N][15];
map<char,int> m;
int arr[N][4], fail[N], cnt[N], pos;

void insert(string &s){
  int now = 0;
  for(auto c : s){
    int &to = arr[now][m[c]];
    now = to ? to : to = ++pos;
  }
  cnt[now] = s.size();
}

void build(){
  queue<int> q;
  for(int i = 0;i < 4;i++)
    if(arr[0][i]) q.push(arr[0][i]);
  while(!q.empty()){
    int now = q.front(); q.pop();
    for(int i = 0;i < 4;i++){
      int &to = arr[now][i], tmp;
      if(to==0) to = arr[fail[now]][i];
      else{
        q.push(to), tmp = fail[now];
        while(tmp && !arr[tmp][i]) cnt[to] =
             max(cnt[to], cnt[arr[tmp][i]]),
            tmp = fail[tmp];
        fail[to] = arr[tmp][i];
        cnt[to] = max(cnt[to],cnt[arr[tmp][i
            ]]);
      }
    }
  }
}

void match(string &s){
  int now = 0;
  for(int i = 0;i < s.size();i++){
    while(now && !arr[now][s[i]-'a']) now =
         fail[now];
    if(arr[now][s[i]-'a']) now = arr[now][s[
        i]-'a'];
    if(cnt[now] > 0){
      //FIND!
    }
  }
}
```

5.2 KMP

```cpp
const int N = 1e6 + 5;
int f[N];

void build_failure(string s) {
    int p = f[0] = -1;

    for (int i = 1; i < s.size(); i++) {
        while (p != -1 && s[p + 1] != s[i])
            p = f[p];

        if (s[p + 1] == s[i])
            p++;

        f[i] = p;
    }
}

int KMP(string s, string t) {
    int cnt = 0;
    int p = -1;

    for (int i = 0; i < t.size(); i++) {
        while (p != -1 && s[p + 1] != t[i])
            p = f[p];

        if (s[p + 1] == t[i])
            p++;

        if (p + 1 == s.size()) {
            cnt++;
        }
    }

    return cnt;
}
```

5.3 Suffix_Array

```cpp
void count_sort(int pos[], int n, vector<int
    > &rank){
  int cnt[n] = {};
  for(int i = 0;i < n;i++) cnt[rank[pos[i
      ]]]++;
  for(int i = 1;i < n;i++) cnt[i] += cnt[i
      -1];
  int ans[n];
  for(int i = n-1;~i;i--){
    ans[cnt[rank[pos[i]]]-1] = pos[i];
    --cnt[rank[pos[i]]];
  }
  for(int i = 0;i < n;i++) pos[i] = ans[i];
}
// 只需要在求完Suffix Array之後順便求LCP就好
    了
void get_suffix(string &s, int pos[], vector
    <int> &lcp){
  s += '$';
  int n = s.size();
  vector<int> rank(n);
  //k = 0
  iota(pos,pos+n,0);
  sort(pos,pos+n,[&](int a, int b){ return s
      [a] < s[b]; });
  for(int i = 0;i < n;i++){
    if(i==0){
      rank[pos[i]] = 0;
    }else{
      rank[pos[i]] = rank[pos[i-1]] + (s[pos
          [i]]!=s[pos[i-1]]);
    }
  }
  //k > 0
  vector<int> new_rank(n);
  for(int k = 0;(1<<k) < n;k++){
    for(int i = 0;i < n;i++)
      pos[i] = (pos[i] - (1<<k)%n + n ) % n;
    count_sort(pos,n,rank);
    new_rank[pos[0]] = 0;
    for(int i = 1;i < n;i++){
      pair<int,int> prev = {rank[pos[i-1]],
          rank[(pos[i-1]+(1<<k))%n]};
      pair<int,int> now = {rank[pos[i]],
          rank[(pos[i]+(1<<k))%n]};
      new_rank[pos[i]] = new_rank[pos[i-1]]
          + (prev!=now);
    }
    rank = new_rank;
  }
  //求LCP Array
  int k = 0;
  for(int i = 0;i < n;i++){
    int pi = rank[i];
    int j = pos[pi-1];
    while(i+k<n && j+k<n && s[i+k]==s[j+k])
        k++;
    lcp[pi] = k;
    k = max(0,k-1);
  }
}
```

5.4 ZValue

```cpp
vector<int> z(const string& s)
{
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 0; i < n; i++) {
        z[i] = max(min(z[i-l],r-i),0);
        while(i+z[i] < n && s[z[i]]==s[i+z[i
            ]])
            l = i, r = i+z[i], z[i]++;
    }
    return z;
}
```

6 Sublime-Text

6.1 SublimeBuild

```
//This is the build file script for sublime-
    text, rename it as XXX.sublime_build

{
"cmd": ["g++", "-std=c++17", "$file", "-o",
    "${file_path}/${file_base_name}"],
"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)
    ?:? (.*)$",
"working_dir": "${file_path}",
"selector": "source.c, source.c++, source.
    cxx, source.cpp",
"variants":
[
{
"name": "Run",
"cmd": ["bash", "-c","g++ -std=c++17 '${file
    }' -I . -o '${file_path}/${
    file_base_name}' && gnome-terminal --
    bash -c \"ulimit -s unlimited; '${
    file_path}/${file_base_name}'; read;exit
    ; exec bash\""]
}
]
}
```