

Contents

1. BASIC	2
1.1. <code>pbd.h</code>	2
2. Ds	3
2.1. <code>bst.h</code>	3
2.2. <code>dsu.h</code>	5
2.3. <code>segTree_add.h</code>	6
2.4. <code>segTree_add_setto.h</code>	9
2.5. <code>segTree_MX_MI.h</code>	11
2.6. <code>ST.h</code>	14
2.7. <code>twoDimPrfxSum.h</code>	15
3. GEO	17
3.1. <code>Rotating_Calipers.h</code>	17
4. GRAPH	19
4.1. FLOW	19
4.1.1. <code>max_Flow_print.h</code>	19
4.1.2. <code>min_Cost.h</code>	21
4.2. TREE	24
4.2.1. <code>hld.h</code>	24
4.2.2. <code>lca.h</code>	25
5. MATH	27
5.1. NUMBER_THEORY	27
5.1.1. <code>basic.h</code>	27
5.1.2. <code>Comb.h</code>	28
5.1.3. <code>CRT.h</code>	29
5.1.4. <code>Eular_phi.h</code>	30
5.1.5. <code>Eular_sieve.h</code>	31
5.1.6. <code>factor_pr.h</code>	32
5.2. OTHER	36
5.2.1. <code>Frac.h</code>	36
6. STRING	38
6.1. <code>AC_automaton.h</code>	38
6.2. <code>compress_print.h</code>	40
6.3. <code>get_occr.h</code>	41
6.4. <code>hash_print.h</code>	42
6.5. <code>KMP.h</code>	44
6.6. <code>trie_Tree.h</code>	45

1. BASIC

1.1. pbds.h

```

1  #include <ext/pb_ds/assoc_container.hpp>
2  #include <ext/pb_ds/tree_policy.hpp>
3  using namespace __gnu_pbds;
4  using namespace std;
5  using ord_set = tree<int, null_type, less<int>, rb_tree_tag,
6  tree_order_statistics_node_update>;
7  using ord_mset = tree<int, null_type, less_equal<int>, rb_tree_tag,
8  tree_order_statistics_node_update>;
9  //find_by_order
10 //order_of_key

```

2. Ds

2.1. bst.h

```

1  const int N=1e5+100;
2  struct node{
3      int s[2];
4      int v,p,cnt,sz;
5      void init(int p1,int v1){
6          p=p1;v=v1;
7          cnt=sz=1;
8      }
9  }tr[N];
10 int root=0,idx=0;
11 void pushup(int x){
12     tr[x].sz=tr[x].cnt+tr[tr[x].s[1]].sz+tr[tr[x].s[0]].sz;
13 }
14 void rotate(int x){
15     int y=tr[x].p;
16     int z=tr[y].p;
17     int k=tr[y].s[1]==x;
18     tr[y].s[k]=tr[x].s[k^1];
19     tr[tr[x].s[k^1]].p=y;
20     tr[z].s[tr[z].s[1]==y]=x;
21     tr[x].p=z;
22     tr[y].p=x;
23     tr[x].s[k^1]=y;
24     pushup(y);pushup(x);
25 }
26 void splay(int x,int k){s
27     while(tr[x].p!=k){
28         int y=tr[x].p;
29         int z=tr[y].p;
30         if(z!=k) (tr[y].s[0]==x)^(tr[z].s[0]==y) ? rotate(x) : rotate(y);
31         rotate(x);
32     }
33     if(k==0) root=x;
34 }
35 void find(int v){
36     int x=root;
37     while(tr[x].v!=v && tr[x].s[v>tr[x].v] ) x=tr[x].s[v>tr[x].v];
38     splay(x,0);
39 }
40 int get_pre(int v){
41     find(v);
42     int x=root;
43     if(tr[x].v<v) return x;
44     x=tr[x].s[0];
45     while(tr[x].s[1]) x=tr[x].s[1];
46     splay(x,0);
47     return x;
48 }
49 int get_suc(int v){
50     find(v);
51     int x=root;
52     if(tr[x].v>v) return x;
53     x=tr[x].s[1];

```

```

54     while(tr[x].s[0]) x=tr[x].s[0];
55     splay(x,0);
56     return x;
57 }
58 void del(int v){
59     int pre=get_pre(v);
60     int suc=get_suc(v);
61     splay(pre,0);splay(suc,pre);
62     int d=tr[suc].s[0];
63     if(tr[d].cnt>1){
64         tr[d].cnt--;splay(d,0);
65     }
66     else{
67         tr[suc].s[0]=0;splay(suc,0);
68     }
69 }
70 void insert(int v){
71     int x=root;
72     int p=0;
73     while(x && tr[x].v!=v){
74         p=x;x=tr[x].s[v>tr[x].v];
75     }
76     if(x) tr[x].cnt++;
77     else{
78         x=++idx;
79         tr[p].s[v>tr[p].v]=x;
80         tr[x].init(p,v);
81     }
82     splay(x,0);
83 }
84 int get_rank(int v){
85     insert(v);
86     int res=tr[tr[root].s[0]].sz;
87     del(v);
88     return res;
89 }
90 int get_val(int k){
91     int x=root;
92     while(1){
93         int y=tr[x].s[0];
94         if(tr[x].cnt+tr[y].sz<k){
95             k-=tr[y].sz+tr[x].cnt;
96             x=tr[x].s[1];
97         }
98         else{
99             if(tr[y].sz>=k) x=tr[x].s[0];
100             else break;
101         }
102     }
103     splay(x,0);
104     return tr[x].v;
105 }

```

2.2. dsu.h

```

1  class DSU {
2      std::vector<int> f, siz;
3  public:
4      DSU() {}
5      DSU(int n) {
6          init(n);
7      }
8
9      void init(int n) {
10         f.resize(n);
11         for(int i = 0; i < n; i++) f[i] = i;
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 };

```

2.3. segTree_add.h

```

1 // AC 带懒惰标记线段树
2 template <class TYPE_NAME>
3 class lazyTree
4 {
5     /*
6      * TYPE_NAME 需要支持: + += != 和自定义的合并运算符
7      * 实现了懒惰标记, 仅支持区间批量增加
8      * 区间批量减需要 TYPE_NAME 支持-, 且有 -a = 0 - a
9      * 额外处理了一个单点修改为对应值的函数, 非原生实现, 其复杂度为 4logn
10     * 不提供在线
11     * 不提供持久化
12     */
13 private:
14     vector<TYPE_NAME> d;
15     vector<TYPE_NAME> a;
16     vector<TYPE_NAME> b;
17     int n;
18     const TYPE_NAME INI = 0; // 不会影响合并运算的初始值, 如 max 取 INF, min 取 0,
19     mti 取 1
20     void subbuild(int s, int t, int p)
21     {
22         if (s == t)
23         {
24             d[p] = a[s];
25             return;
26         }
27         int m = s + ((t - s) >> 1); // (s+t)/2
28         subbuild(s, m, p * 2);
29         subbuild(m + 1, t, p * 2 + 1);
30         d[p] = d[p * 2] + d[p * 2 + 1];
31         // 合并运算符 ↑
32     }
33     TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
34     {
35         if (l <= s && t <= r)
36             return d[p];
37         int m = s + ((t - s) >> 1);
38         if (b[p] != 0)
39         {
40             d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应
41             用懒惰标记
42             d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应
43             用懒惰标记
44             b[p * 2] += b[p]; // 下传标记, 无需修改
45             b[p * 2 + 1] += b[p]; // 下传标记, 无需修改
46             b[p] = 0;
47         }
48         TYPE_NAME ans1 = INI;
49         TYPE_NAME ansr = INI;
50         if (l <= m)
51             ans1 = subGetSum(l, r, s, m, p * 2);
52         if (r > m)
53             ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);

```

```

53         return ans1 + ansr;
54         // 合并运算符↑
55     }
56
57     void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
58     {
59         if (l <= s && t <= r)
60         {
61             d[p] += (t - s + 1) * c; // 合并运算符的高阶运算 此处运算为修改整匹配
62             b[p] += c;                // 记录懒惰标记, 无需修改
63             return;
64         }
65         int m = s + ((t - s) >> 1);
66         if (b[p] != 0 && s != t)
67         {
68             d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应
69             d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应
70             b[p * 2] += b[p];                // 下传标记, 无需修改
71             b[p * 2 + 1] += b[p];            // 下传标记, 无需修改
72             b[p] = 0;
73         }
74         if (l <= m)
75             subUpdate(l, r, c, s, m, p * 2);
76         if (r > m)
77             subUpdate(l, r, c, m + 1, t, p * 2 + 1);
78         d[p] = d[p * 2] + d[p * 2 + 1];
79         // 合并运算符 ↑
80     }
81
82     public:
83         lazyTree(TYPE_NAME _n)
84         {
85             n = _n;
86             d.resize(4 * n + 5);
87             a.resize(4 * n + 5);
88             b.resize(4 * n + 5);
89         }
90
91         void build(vector<TYPE_NAME> _a)
92         {
93             a = _a;
94             subbuild(1, n, 1);
95         }
96
97         TYPE_NAME getsum(int l, int r)
98         {
99             return subGetSum(l, r, 1, n, 1);
100         }
101
102         void update(int l, int r, TYPE_NAME c) // 修改区间
103         {
104             subUpdate(l, r, c, 1, n, 1);
105         }
106

```

```
107     void update(int idx, TYPE_NAME tar)
108     { // 修改单点, 未测试
109         TYPE_NAME tmp = getsum(idx, idx);
110         tar -= tmp;
111         subUpdate(idx, idx, tar, 1, n, 1);
112     }
113 };
```


2.4. segTree_add_setto.h

```

1  template <typename T>
2  class SegTreeLazyRangeSet {
3      vector<T> tree, lazy;
4      vector<T> *arr;
5      vector<bool> ifLazy;
6      int n, root, n4, end;
7
8      void maintain(int cl, int cr, int p) {
9          int cm = cl + (cr - cl) / 2;
10         if (cl != cr && ifLazy[p]) {
11             lazy[p * 2] = lazy[p], ifLazy[p*2] = 1;
12             lazy[p * 2 + 1] = lazy[p], ifLazy[p*2+1] = 1;
13             tree[p * 2] = lazy[p] * (cm - cl + 1);
14             tree[p * 2 + 1] = lazy[p] * (cr - cm);
15             lazy[p] = 0;
16             ifLazy[p] = 0;
17         }
18     }
19
20     T range_sum(int l, int r, int cl, int cr, int p) {
21         if (l <= cl && cr <= r) return tree[p];
22         int m = cl + (cr - cl) / 2;
23         T sum = 0;
24         maintain(cl, cr, p);
25         if (l <= m) sum += range_sum(l, r, cl, m, p * 2);
26         if (r > m) sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
27         return sum;
28     }
29
30     void range_set(int l, int r, T val, int cl, int cr, int p) {
31         if (l <= cl && cr <= r) {
32             lazy[p] = val;
33             ifLazy[p] = 1;
34             tree[p] = (cr - cl + 1) * val;
35             return;
36         }
37         int m = cl + (cr - cl) / 2;
38         maintain(cl, cr, p);
39         if (l <= m) range_set(l, r, val, cl, m, p * 2);
40         if (r > m) range_set(l, r, val, m + 1, cr, p * 2 + 1);
41         tree[p] = tree[p * 2] + tree[p * 2 + 1];
42     }
43
44     void build(int s, int t, int p) {
45         if (s == t) {
46             tree[p] = (*arr)[s];
47             return;
48         }
49         int m = s + (t - s) / 2;
50         build(s, m, p * 2);
51         build(m + 1, t, p * 2 + 1);
52         tree[p] = tree[p * 2] + tree[p * 2 + 1];
53     }
54
55     public:
56     explicit SegTreeLazyRangeSet<T>(vector<T> v) {
57         n = v.size();

```

```

58     n4 = n * 4;
59     tree = vector<T>(n4, 0);
60     lazy = vector<T>(n4, 0);
61     ifLazy = vector<bool>(n4,0);
62     arr = &v;
63     end = n - 1;
64     root = 1;
65     build(0, end, 1);
66     arr = nullptr;
67 }
68
69 void show(int p, int depth = 0) {
70     if (p > n4 || tree[p] == 0) return;
71     show(p * 2, depth + 1);
72     for (int i = 0; i < depth; ++i) putchar('\t');
73     printf("%d:%d\n", tree[p], lazy[p]);
74     show(p * 2 + 1, depth + 1);
75 }
76
77 T range_sum(int l, int r) { return range_sum(l, r, 0, end, root); }
78
79 void range_set(int l, int r, T val) { range_set(l, r, val, 0, end, root); }
80 };

```

2.5. segTree_MX_MI.h

```

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

```

```

57         return info[p];
58     }
59     int m = (l + r) / 2;
60     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
61 y);
62 }
63 Info rangeQuery(int l, int r) {
64     return rangeQuery(1, 0, n, l, r);
65 }
66 template<class F>
67 int findFirst(int p, int l, int r, int x, int y, F &&pred) {
68     if (l >= y || r <= x) {
69         return -1;
70     }
71     if (l >= x && r <= y && !pred(info[p])) {
72         return -1;
73     }
74     if (r - l == 1) {
75         return l;
76     }
77     int m = (l + r) / 2;
78     int res = findFirst(2 * p, l, m, x, y, pred);
79     if (res == -1) {
80         res = findFirst(2 * p + 1, m, r, x, y, pred);
81     }
82     return res;
83 }
84 template<class F>
85 int findFirst(int l, int r, F &&pred) {
86     return findFirst(1, 0, n, l, r, pred);
87 }
88 template<class F>
89 int findLast(int p, int l, int r, int x, int y, F &&pred) {
90     if (l >= y || r <= x) {
91         return -1;
92     }
93     if (l >= x && r <= y && !pred(info[p])) {
94         return -1;
95     }
96     if (r - l == 1) {
97         return l;
98     }
99     int m = (l + r) / 2;
100     int res = findLast(2 * p + 1, m, r, x, y, pred);
101     if (res == -1) {
102         res = findLast(2 * p, l, m, x, y, pred);
103     }
104     return res;
105 }
106 template<class F>
107 int findLast(int l, int r, F &&pred) {
108     return findLast(1, 0, n, l, r, pred);
109 }
110 };
111 const int inf = 1E9;
112 struct Info
113 {

```

```

113     int mn {inf}, mnId, mx {-inf}, mxId;
114 } ;
115 Info operator+(Info a, Info b)
116 {
117     if (a.mn > b.mn)
118         a.mn = b.mn, a.mnId = b.mnId;
119     if (a.mx < b.mx)
120         a.mx = b.mx, a.mxId = b.mxId;
121     return a;
122 }

```

2.6. ST.h

```

1  class SparseTable
2  {
3      using func_type = function<int(const int &, const int &)>;
4
5      vector<vector<int>> ST;
6      int len;
7      vector<int> preLog;
8      func_type op;
9      static int default_func(const int &t1, const int &t2) { return max(t1,
10 t2); }
11 public:
12     void build(const vector<int> &v, func_type _func = default_func)
13     {
14         op = _func;
15         len = v.size();
16         int l1 = ceil(log2(len)) + 1;
17         ST.assign(len, vector<int>(l1, 0));
18         for (int i = 0; i < len; ++i)
19         {
20             ST[i][0] = v[i];
21         }
22         for (int j = 1; j < l1; ++j)
23         {
24             int pj = (1 << (j - 1));
25             for (int i = 0; i + pj < len; ++i)
26             {
27                 ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
28             }
29         }
30         preLog.resize(len + 1);
31         for (int i = 1; i <= len; ++i) preLog[i] = floor(log2(i));
32     }
33
34     int getsum(int l, int r)
35     {
36         if (r < l)
37             return 0;
38         l = max(0, l), r = min(len - 1, r);
39         if (r == 0)
40             return 0;
41         int lt = r - l + 1;
42         int q = preLog[lt];
43         return op(ST[l][q], ST[r - (1 << q) + 1][q]);
44     }
45 };

```

2.7. twoDimPrfxSum.h

```

1  class prfx_2{
2  public:
3      vector<vector<int>> mtx;
4      int n,m;
5      public:
6      prfx_2(vector<vector<int>> _mtx){init(_mtx);};
7      prfx_2() { };
8      void init(vector<vector<int>> _mtx)
9      {
10         n = _mtx.size();
11         m = _mtx[0].size();
12         mtx.resize(n+1);
13         for(auto &x:mtx) x.resize(m+1);
14         lop(i,1,n+1)
15             lop(j,1,m+1)
16                 _mtx[i][j] = _mtx[i-1][j] + _mtx[i][j-1] - _mtx[i-1][j-1] +
17             }
18
19         int getsum(int x1,int y1,int x2,int y2)
20         {
21             x1 ++,x2 ++,y1 ++,y2 ++;
22             return _mtx[x2][y2] - _mtx[x1-1][y2] - _mtx[x2][y1-1] + _mtx[x1-1][y1-1];
23         }
24
25         int getsum(pii d1,pii d2)
26         {
27             auto [x1,y1] = d1;
28             auto [x2,y2] = d2;
29             x1 ++,x2 ++,y1 ++,y2 ++;
30             return _mtx[x2][y2] - _mtx[x1-1][y2] - _mtx[x2][y1-1] + _mtx[x1-1][y1-1];
31         }
32
33         vector<int> & operator[](std::size_t i)
34         {
35             return mtx[i];
36         }
37     };
38
39     class conj_diff_2{
40     public:
41         vector<vector<int>> mtx;
42         vector<vector<int>> prf;
43         int n,m;
44         public:
45
46         conj_diff_2(int _n,int _m)
47         {
48             n = _n+1,m = _m+1;
49             vector<vector<int>> initmp(n,vector<int> (m,0));
50             init(initmp);
51         }
52
53         conj_diff_2(vector<vector<int>> _mtx)
54         {
55             this->init(_mtx);
56         }

```

```

57
58     conj_diff_2(){ }
59
60     void init(vector<vector<int>> _mtx)
61     {
62         n = _mtx.size();
63         m = _mtx[0].size();
64         mtx.resize(n+2);
65         for(auto &x:mtx) x.resize(m+2);
66         prf.resize(n+2);
67         for(auto &x:prf) x.resize(m+2);
68         lop(i,1,n+1)
69             lop(j,1,m+1)
70                 prf[i][j] = _mtx[i-1][j-1];
71     }
72
73     void modify(int x1,int y1,int x2,int y2,int k)
74     {
75         x1 ++,x2 ++,y1 ++,y2 ++;
76         mtx[x1][y1] += k;
77         mtx[x2+1][y1] -= k,mtx[x1][y2+1] -= k;
78         mtx[x2+1][y2+1] += k;
79     }
80
81     void modify(pii d1,pii d2,int k)
82     {
83         this->modify(d1.fi,d1.se,d2.fi,d2.se,k);
84     }
85
86     vector<vector<int>> cacu()
87     {
88         auto res = prfx_2(mtx);
89         vector<vector<int>> rst(n,vector<int>(m));
90         lop(i,1,n+1)
91             lop(j,1,m+1)
92                 rst[i-1][j-1] = prf[i][j] + res[i+1][j+1];
93         return rst;
94     }
95
96     vector<int> & operator[](std::size_t i)
97     {
98         return mtx[i];
99     }
100 };

```


3. GEO

3.1. Rotating_Calipers.h

```

1 //Rotating_Calipers
2 template<typename VALUE_TYPE>
3 class Rotating_Calipers
4 {
5 public:
6     using pv = pair<VALUE_TYPE, VALUE_TYPE>;
7     using vec_pv = vector<pair<VALUE_TYPE, VALUE_TYPE>>;
8     vec_pv p;
9
10    static VALUE_TYPE cross(pv p1, pv p2, pv p0)
11    {
12        pv t1 = {p1.fi - p0.fi, p1.se - p0.se},
13        t2 = {p2.fi - p0.fi, p2.se - p0.se};
14        return t1.fi * t2.se - t1.se * t2.fi;
15    }
16
17    static VALUE_TYPE dis(const pv &p1, const pv &p2){
18        return (p1.fi - p2.fi) * (p1.fi - p2.fi) + (p1.se - p2.se) * (p1.se -
19        p2.se);
20    };
21 public:
22
23    Rotating_Calipers() {}
24
25    Rotating_Calipers(vec_pv _A) {
26        build(_A);
27    }
28
29    void build(const vec_pv & _A) {
30        p = ConvexHull(_A);
31    }
32
33    static vec_pv ConvexHull(vec_pv A, VALUE_TYPE flag = 1)
34    {
35        int n = A.size();
36        if (n <= 2) return A;
37        vec_pv ans(n * 2);
38        sort(A.begin(), A.end(),
39        [](pv a, pv b) -> bool {
40            if(fabs(a.fi - b.fi) < 1e-10)
41                return a.se < b.se;
42            else return a.fi < b.fi;});
43        int now = -1;
44        for (int i = 0; i < n; i++)
45        { // 维护下凸包
46            while (now > 0 && cross(A[i], ans[now], ans[now - 1]) < flag)
47                now--;
48            ans[++now] = A[i];
49        }
50        int pre = now;
51        for (int i = n - 2; i >= 0; i--)
52        { // 维护上凸包

```

```

52         while (now > pre && cross(A[i], ans[now], ans[now - 1]) < flag)
now--;
53         ans[++now] = A[i];
54     }
55     ans.resize(now);
56     return ans;
57 }
58
59 VALUE_TYPE getDiameter() {
60     int j = 1;
61     VALUE_TYPE ans = 0;
62     int m = p.size();
63     p.push_back(p[0]);
64     for(int i = 0; i < m; i++)
65     {
66         while( cross(p[i+1], p[j], p[i]) > cross(p[i+1], p[j+1], p[i]) ) j =
(j+1)%m;
67         ans = max(ans, max( dis(p[i], p[j]) , dis(p[i+1], p[j]) ) );
68     }
69     p.pop_back();
70     return ans;
71 }
72
73 VALUE_TYPE getPerimeter() {
74     VALUE_TYPE sum = 0;
75     p.pb(p[0]);
76     for(int i = 0; i < (int)p.size() - 1; i++)
77     {
78         sum += sqrtl(dis(p[i], p[i+1]));
79     }
80     p.pop_back();
81     return sum;
82 }
83
84 };

```

4. GRAPH

4.1. FLOW

4.1.1. max_Flow_print.h

```

1  class maxFlow//AC
2  {
3  private:
4      class edge
5      {
6      public:
7          ll int nxt, cap, flow;
8          pair<int, int> revNodeIdx;
9      public:
10         edge(int _nxt, int _cap)
11         {
12             nxt = _nxt, cap = _cap, flow = 0;
13         }
14         void setRevIdx(int _i, int _j) { revNodeIdx = {_i,_j}; }
15     };
16     vector<vector<edge>> edgeList;
17     vector<int> dep,fir;
18     ll int maxFlowAns;
19     int T, S;
20 public:
21     maxFlow(int _n)
22     {
23         maxFlowAns = 0;
24         S = 1;
25         T = _n;
26         edgeList.resize(_n + 1);
27         dep.resize(_n + 1);
28         fir.resize(_n+1);
29     }
30     void resetTS(int _T, int _S) { T = _T,S = _S; }
31
32     void addedge(int _u, int _v, int _w)
33     {
34         edgeList[_u].push_back(edge(_v, _w));
35         edgeList[_v].push_back(edge(_u, 0));
36         edgeList[_u][edgeList[_u].size() - 1].setRevIdx(_v,
edgeList[_v].size() - 1);
37         edgeList[_v][edgeList[_v].size() - 1].setRevIdx(_u,
edgeList[_u].size() - 1);
38     }
39
40     bool bfs()
41     {
42         queue<int> que;
43         for (auto x = dep.begin(); x != dep.end(); x++) (*x) = 0;
44         dep[S] = 1;
45         que.push(S);
46         while (que.size())
47         {
48             ll int at = que.front();
49             que.pop();

```

```

50         for (int i = 0; i < edgeList[at].size(); i++)
51         {
52             auto tar = edgeList[at][i];
53             if ((!dep[tar.nxt]) && (tar.flow < tar.cap))
54             {
55                 dep[tar.nxt] = dep[at] + 1;
56                 que.push(tar.nxt);
57             }
58         }
59     }
60     return dep[T];
61 }
62
63 ll int dfs(int at, ll int flow)
64 {
65     if ((at == T) || (!flow))
66         return flow; // 到了或者没了
67     ll int ret = 0; // 本节点最大流
68     for (int &i = fir[at]; i < edgeList[at].size(); i++)
69     {
70         auto tar = edgeList[at][i]; // 目前遍历的边
71         int tlFlow = 0; // 目前边的最大流
72         if (dep[at] == dep[tar.nxt] - 1) // 遍历到的边为合法边
73         {
74             tlFlow = dfs(tar.nxt, min((ll)tar.cap - tar.flow, flow -
75 ret));
76             if (!tlFlow)
77                 continue; // 若最大流为 0, 什么都不做
78             ret += tlFlow; // 本节点最大流累加
79             edgeList[at][i].flow += tlFlow; // 本节点实时流量
80             edgeList[tar.revNodeIdx.first][tar.revNodeIdx.second].flow -=
81 tlFlow; // 反向边流量
82             if (ret == flow)
83                 return ret; // 充满了就不继续扫了
84         }
85     }
86     return ret;
87 }
88
89 ll int dinic()
90 {
91     if (maxFlowAns)
92         return maxFlowAns;
93     while (bfs())
94     {
95         for(auto x = fir.begin(); x != fir.end(); x++) (*x) = 0;
96         maxFlowAns += dfs(S, INT_MAX);
97     }
98     return maxFlowAns;
99 }

```

4.1.2. min_Cost.h

```

1  const int INF = 0x3f3f3f3f
2
3  class PD//AC
4  {
5  public:
6      class edge
7      {
8      public:
9          int v, f, c, next;
10         edge(int _v,int _f,int _c,int _next)
11         {
12             v = _v,f = _f,c = _c,next = _next;
13         }
14         edge() { }
15     } ;
16
17     void vecset(int value,vector<int> &arr)
18     {
19         for(int i = 0;i < arr.size();i ++) arr[i] = value;
20         return;
21     }
22
23     class node
24     {
25     public:
26         int v, e;
27     } ;
28
29     class mypair
30     {
31     public:
32         int dis, id;
33
34         bool operator<(const mypair &a) const { return dis > a.dis; }
35
36         mypair(int d, int x)
37         {
38             dis = d;
39             id = x;
40         }
41     };
42
43     vector<int> head,dis,vis,h;
44     vector<edge> e;
45     vector<node> p;
46     int n, m, s, t, cnt = 1, maxf, minc;
47
48     PD(int _n,int _m,int _s,int _t)
49     {
50         n = _n, m = _m,s = _s,t = _t;
51         maxf = 0, minc = 0;
52         head.resize(n+2), dis.resize(n+2),vis.resize(n+2);
53         e.resize(2);
54         h.resize(n+2), p.resize(m+2);
55     }
56
57     void addedge(int u, int v, int f, int c)
58     {

```

```

59     e.push_back(edge(v,f,c,head[u]));
60     head[u] = e.size()-1;
61     e.push_back(edge(u,0,-c,head[v]));
62     head[v] = e.size()-1;
63 }
64
65 bool dijkstra()
66 {
67     priority_queue<mypair> q;
68     vecset(INF,dis);
69     vecset(0,vis);
70     dis[s] = 0;
71     q.push(mypair(0, s));
72     while (!q.empty())
73     {
74         int u = q.top().id;
75         q.pop();
76         if (vis[u])
77             continue;
78         vis[u] = 1;
79         for (int i = head[u]; i; i = e[i].next)
80         {
81             int v = e[i].v, nc = e[i].c + h[u] - h[v];
82             if (e[i].f && dis[v] > dis[u] + nc)
83             {
84                 dis[v] = dis[u] + nc;
85                 p[v].v = u;
86                 p[v].e = i;
87                 if (!vis[v])
88                     q.push(mypair(dis[v], v));
89             }
90         }
91     }
92     return dis[t] != INF;
93 }
94
95 void spfa()
96 {
97     queue<int> q;
98     vecset(63,h);
99     h[s] = 0, vis[s] = 1;
100    q.push(s);
101    while (!q.empty())
102    {
103        int u = q.front();
104        q.pop();
105        vis[u] = 0;
106        for (int i = head[u]; i; i = e[i].next)
107        {
108            int v = e[i].v;
109            if (e[i].f && h[v] > h[u] + e[i].c)
110            {
111                h[v] = h[u] + e[i].c;
112                if (!vis[v])
113                {
114                    vis[v] = 1;
115                    q.push(v);
116                }
117            }
118        }
119    }
120 }

```

```

117         }
118     }
119 }
120
121
122 int pd()
123 {
124     spfa();
125     while (dijkstra())
126     {
127         int minf = INF;
128         for (int i = 1; i <= n; i++)
129             h[i] += dis[i];
130         for (int i = t; i != s; i = p[i].v)
131             minf = min(minf, e[p[i].e].f);
132         for (int i = t; i != s; i = p[i].v)
133         {
134             e[p[i].e].f -= minf;
135             e[p[i].e ^ 1].f += minf;
136         }
137         maxf += minf;
138         minc += minf * h[t];
139     }
140     return 0;
141 }
142
143 pair<int,int> get()
144 {
145     return {maxf,minc};
146 }
147 };

```

4.2. TREE

4.2.1. hld.h

```

1 void dfs1(int o) {
2     son[o] = -1;
3     siz[o] = 1;
4     for (int j = h[o]; j; j = nxt[j])
5         if (!dep[p[j]]) {
6             dep[p[j]] = dep[o] + 1;
7             fa[p[j]] = o;
8             dfs1(p[j]);
9             siz[o] += siz[p[j]];
10            if (son[o] == -1 || siz[p[j]] > siz[son[o]]) son[o] = p[j];
11        }
12 }
13
14 void dfs2(int o, int t) {
15     top[o] = t;
16     cnt++;
17     dfn[o] = cnt;
18     rnk[cnt] = o;
19     if (son[o] == -1) return;
20     dfs2(son[o], t); // 优先对重儿子进行 DFS, 可以保证同一条重链上的点 DFS 序连续
21     for (int j = h[o]; j; j = nxt[j])
22         if (p[j] != son[o] && p[j] != fa[o]) dfs2(p[j], p[j]);
23 }
24
25 int lca(int u, int v) {
26     while (top[u] != top[v]) {
27         if (dep[top[u]] > dep[top[v]])
28             u = fa[top[u]];
29         else
30             v = fa[top[v]];
31     }
32     return dep[u] > dep[v] ? v : u;
33 }
34
35 // st 是线段树结构体
36 int querymax(int x, int y) {
37     int ret = -inf, fx = top[x], fy = top[y];
38     while (fx != fy) {
39         if (dep[fx] >= dep[fy])
40             ret = max(ret, st.query1(1, 1, n, dfn[fx], dfn[x])), x = fa[fx];
41         else
42             ret = max(ret, st.query1(1, 1, n, dfn[fy], dfn[y])), y = fa[fy];
43         fx = top[x];
44         fy = top[y];
45     }
46     if (dfn[x] < dfn[y])
47         ret = max(ret, st.query1(1, 1, n, dfn[x], dfn[y]));
48     else
49         ret = max(ret, st.query1(1, 1, n, dfn[y], dfn[x]));
50     return ret;
51 }

```


4.2.2. lca.h

```

1  class LCA{
2  public:
3      vector<vector<pii>> cnj;
4      vector<int> lg,dep;
5      vector<array<int,32>> fa,wei;
6      int n;
7
8      LCA(int _n) {
9          n = _n;
10         cnj.resize(n+1);
11         lg.resize(n+1),fa.resize(n+1),dep.resize(n+1),wei.resize(n+1);
12         for(int i = 1; i <= n; i++)
13             lg[i] = lg[i-1] + (1 << lg[i-1] == i);
14     }
15
16     void addEdge(int u,int v,int w) {
17         cnj[u].push_back({v,w});
18         cnj[v].push_back({u,w});
19     }
20
21     void build(int rt = 1) {
22         using itf = function<void(int,int)>;
23         itf dfs = [&](int p,int f) -> void {
24             fa[p][0] = f,dep[p] = dep[f] + 1;
25             // wei[p][0] = 0;
26             for(int i = 1;i <= lg[dep[p]];i++) fa[p][i] = fa[fa[p][i-1]]
27 [i-1];
28             for(int i = 1;i <= lg[dep[p]];i++) wei[p][i] = max(wei[p]
29 [i-1],wei[fa[p][i-1]][i-1]);
30             for(auto [x,w]:cnj[p]) if(x == f) continue;
31             else wei[x][0] = w,dfs(x,p);
32         };
33         dfs(rt,0);
34     }
35
36     int get(int x,int y) {
37         if(dep[x] < dep[y]) swap(x,y);
38         while(dep[x] > dep[y]) x = fa[x][lg[dep[x]] - dep[y] - 1];
39         if(x == y) return x;
40         for(int k = lg[dep[x]]-1;k >= 0;k--) if(fa[x][k] != fa[y][k]) x =
41 fa[x][k],y = fa[y][k];
42         return fa[x][0];
43     }
44
45     int getmaxw(int x,int y) {
46         int curmx = 0;
47         if(dep[x] < dep[y]) swap(x,y);
48         while(dep[x] > dep[y]) curmx = max(curmx,wei[x][lg[dep[x]] - dep[y] -
49 1]), x = fa[x][lg[dep[x]] - dep[y] - 1];
50         if(x == y) return curmx;
51         for(int k = lg[dep[x]]-1;k >= 0;k--)
52             if(fa[x][k] != fa[y][k])
53                 curmx = max(curmx,wei[x][k]),x = fa[x][k],
54                 curmx = max(curmx,wei[y][k]),y = fa[y][k];
55         return max({curmx,wei[x][0],wei[y][0]});
56     }
57 }

```

```
53 };
```

5. MATH

5.1. NUMBER_THEORY

5.1.1. basic.h

```

1  __builtin_ffsll(x)
2  // 返回 x 的二进制末尾最后一个 1 的位置
3
4  __builtin_clzll(x)
5  // 返回 x 的二进制的前导 0 的个数。
6
7  __builtin_ctzll(x)
8  // 返回 x 的二进制末尾连续 0 的个数。
9
10 __builtin_clrsbll(x)
11 // 当 x 的符号位为 0 时返回 x 的二进制的前导 0 的个数减一，否则返回 x 的二进制的前导
12 1 的个数减一。
13
14 __builtin_popcountll(x)
15 // 返回 x 的二进制中 1 的个数。
16
17 __builtin_parity(x)
18 // 判断 x 的二进制中 1 的个数的奇偶性。
19
20 int binpow(int x, int y)
21 {
22     int res = 1;
23     while (y > 0)
24     {
25         if (y & 1)
26             res = res * x % mod;
27         x = x * x % mod;
28         y >>= 1;
29     }
30     return res;
31 }
32
33 void exgcd(int a, int b, int& x, int& y) {
34     if (b == 0) {
35         x = 1, y = 0;
36         return;
37     }
38     exgcd(b, a % b, y, x);
39     y -= a / b * x;
40 }
41
42 binpow(x,mod-2)

```

5.1.2. Comb.h

```

1  const int N = 1e6;
2  const int mod = 1e9+7;
3
4  int binpow(int x, int y)
5  {
6      int ans = 1;
7      while (y)
8      {
9          if (y & 1) ans = ans * x % mod;
10         x = x * x % mod;
11         y >>= 1;
12     }
13     return ans;
14 }
15
16 vector<int> fac(N), inv(N);
17
18 void init()
19 {
20     fac[0] = inv[0] = 1;
21     for (int i = 1; i < N; i++) fac[i] = fac[i - 1] * i % mod;
22     inv[N - 1] = binpow(fac[N - 1], mod - 2);
23     for (int i = N - 2; i >= 1; i--)
24     {
25         inv[i] = inv[i + 1] * (i + 1) % mod;
26     }
27 }
28
29 auto C = [&](int x, int y) -> int
30 {
31     return (fac[x] * inv[y] % mod) * inv[x - y] % mod;
32 };

```

5.1.3. CRT.h

```

1  int CRT(vector<int> &r, vector<int> &a)
2  { // % r == a
3      int n = a.size();
4      __int128 k = 1, ans = 0;
5      for (int i = 0; i < n; i++) k *= r[i];
6      for (int i = 0; i < n; i++)
7      {
8          __int128 m = k / r[i];
9          int b, y;
10         exgcd(m, r[i], b, y); // b * m mod r[i] = 1
11         ans = (ans + a[i] * m * b % k) % k;
12     }
13     return (ans % k + k) % k;
14 }
15
16
17
18 int mul(int a, int b, int m) {
19     return (__int128)a * b % m;
20 }
21
22 int exgcd (int a,int b,int &x,int &y) {
23     if (b == 0) { x = 1, y = 0; return a; }
24     int g = exgcd(b, a % b, x, y), tp = x;
25     x = y, y = tp - a / b * y;
26     return g;
27 };
28
29 int EXCRT(vector<int> &a,vector<int> &r) { // % r == a
30     int x, y, k;
31     int n = r.size();
32     int M = a[0], ans = r[0];
33     for (int i = 1; i < n; ++ i) {
34         int ca = M, cb = a[i], cc = (r[i] - ans % cb + cb) % cb;
35         int gcd = exgcd(ca, cb, x, y), bg = cb / gcd;
36         if (cc % gcd != 0) return -1;
37         x = mul(x, cc / gcd, bg);
38         ans += x * M;
39         M *= bg;
40         ans = (ans % M + M) % M;
41     }
42     return (ans % M + M) % M;
43 }

```

5.1.4. Euler_phi.h

```

1  int euler_phi(int n) {
2      int ans = n;
3      for (int i = 2; i * i <= n; i++)
4          if (n % i == 0) {
5              ans = ans / i * (i - 1);
6              while (n % i == 0) n /= i;
7          }
8      if (n > 1) ans = ans / n * (n - 1);
9      return ans;
10 }
```

5.1.5. Euler_sieve.h

```

1  vector<int> init(int n)
2  {
3      vector<int> pri;
4      vector<bool> vis(n, 0);
5      for (int i = 2; i <= n; i++)
6      {
7          if (!vis[i])
8              pri.push_back(i);
9          for (int j = 0; j < pri.size(); j++)
10             {
11                 if (i * pri[j] > n)
12                     break;
13                 vis[pri[j] * i] = 1;
14                 if (i % pri[j] == 0)
15                     break;
16             }
17     }
18     return pri;
19 }
```

5.1.6. factor_pr.h

```

1  #define int long long
2  #define pii pair<int, int>
3  const int INF = 1145141919810LL;
4  using namespace std;
5
6  class Pollard_Rho
7  {
8  private:
9
10     vector<int> B;
11
12     int mul(int a, int b, int m)
13     {
14         int r = a * b - m * (int)(1.L / m * a * b);
15         return r - m * (r >= m) + m * (r < 0);
16     }
17
18     int mypow(int a, int b, int m)
19     {
20         int res = 1 % m;
21         for (; b; b >>= 1, a = mul(a, a, m))
22         {
23             if (b & 1)
24             {
25                 res = mul(res, a, m);
26             }
27         }
28         return res;
29     }
30
31     bool MR(int n)
32     {
33         if (n <= 1)
34             return 0;
35         for (int p : B)
36         {
37             if (n == p)
38                 return 1;
39             if (n % p == 0)
40                 return 0;
41         }
42         int m = (n - 1) >> __builtin_ctz(n - 1);
43         for (int p : B)
44         {
45             int t = m, a = mypow(p, m, n);
46             while (t != n - 1 && a != 1 && a != n - 1)
47             {
48                 a = mul(a, a, n);
49                 t *= 2;
50             }
51             if (a != n - 1 && t % 2 == 0)
52                 return 0;
53         }
54         return 1;
55     }
56
57     inline const int getfecsum(int _n)

```



```

58     {
59         int sum = 0;
60         while (_n)
61         {
62             sum += _n % 10;
63             _n /= 10;
64         }
65         return sum;
66     };
67
68     int PR(int n)
69     {
70         for (int p : B)
71         {
72             if (n % p == 0)
73                 return p;
74         }
75         auto f = [&](int x) -> int
76         {
77             x = mul(x, x, n) + 1;
78             return x >= n ? x - n : x;
79         };
80         int x = 0, y = 0, tot = 0, p = 1, q, g;
81         for (int i = 0; (i & 255) || (g = gcd(p, n)) == 1; i++, x = f(x), y =
f(f(y)))
82         {
83             if (x == y)
84             {
85                 x = tot++;
86                 y = f(x);
87             }
88             q = mul(p, abs(x - y), n);
89             if (q)
90                 p = q;
91         }
92         return g;
93     }
94
95     vector<int> fac(int n)
96     {
97         // if(n == 0)
98         // #define pb emplace_back
99         if (n <= 1)
100             return {};
101         if (MR(n))
102             return {n};
103         int d = PR(n);
104         auto v1 = fac(d), v2 = fac(n / d);
105         auto i1 = v1.begin(), i2 = v2.begin();
106         vector<int> ans;
107         while (i1 != v1.end() || i2 != v2.end())
108         {
109             if (i1 == v1.end())
110             {
111                 ans.pb(*i2++);
112             }
113             else if (i2 == v2.end())
114             {

```

```

115         ans.pb(*i1++);
116     }
117     else
118     {
119         if (*i1 < *i2)
120         {
121             ans.pb(*i1++);
122         }
123         else
124         {
125             ans.pb(*i2++);
126         }
127     }
128 }
129 return ans;
130 }
131
132 public:
133
134 Pollard_Rho(){
135     B = {2, 3, 5, 7, 11, 13, 17, 19, 23};
136 }
137
138 vector<pii> fac_Comp(int n)
139 {
140     auto srt = fac(n);
141     map<int, int> cnt;
142     for (auto x : srt)
143         cnt[x]++;
144     vector<pii> rt;
145     for (auto x : cnt)
146         rt.push_back(x);
147     return rt;
148 }
149
150 vector<int> fac_pri(int n)
151 {
152     return fac(n);
153 }
154
155 vector<int> fac_all(int n)
156 {
157     vector<pii> rt = fac_Comp(n);
158     vector<int> v;
159     function<void(int, int)> dfs = [&](int id, int x)
160     {
161         if (id == rt.size())
162         {
163             v.push_back(x);
164             return;
165         }
166         for(int i = 0; i <= rt[id].se; i++)
167         {
168             dfs(id + 1, x * (mypow(rt[id].fi, i, INF)));
169         }
170     };
171     dfs(0, 1);
172     return v;

```

```
173     }  
174 };
```

5.2. OTHER

5.2.1. Frac.h

```

1  template<class T>
2  struct Frac {
3      T num;
4      T den;
5      Frac(T num_, T den_) : num(num_), den(den_) {
6          if (den < 0) {
7              den = -den;
8              num = -num;
9          }
10     }
11     Frac() : Frac(0, 1) {}
12     Frac(T num_) : Frac(num_, 1) {}
13     explicit operator double() const {
14         return 1. * num / den;
15     }
16     Frac &operator+=(const Frac &rhs) {
17         num = num * rhs.den + rhs.num * den;
18         den *= rhs.den;
19         return *this;
20     }
21     Frac &operator-=(const Frac &rhs) {
22         num = num * rhs.den - rhs.num * den;
23         den *= rhs.den;
24         return *this;
25     }
26     Frac &operator*=(const Frac &rhs) {
27         num *= rhs.num;
28         den *= rhs.den;
29         return *this;
30     }
31     Frac &operator/=(const Frac &rhs) {
32         num *= rhs.den;
33         den *= rhs.num;
34         if (den < 0) {
35             num = -num;
36             den = -den;
37         }
38         return *this;
39     }
40     friend Frac operator+(Frac lhs, const Frac &rhs) {
41         return lhs += rhs;
42     }
43     friend Frac operator-(Frac lhs, const Frac &rhs) {
44         return lhs -= rhs;
45     }
46     friend Frac operator*(Frac lhs, const Frac &rhs) {
47         return lhs *= rhs;
48     }
49     friend Frac operator/(Frac lhs, const Frac &rhs) {
50         return lhs /= rhs;
51     }
52     friend Frac operator-(const Frac &a) {
53         return Frac(-a.num, a.den);
54     }

```

```

55     friend bool operator==(const Frac &lhs, const Frac &rhs) {
56         return lhs.num * rhs.den == rhs.num * lhs.den;
57     }
58     friend bool operator!=(const Frac &lhs, const Frac &rhs) {
59         return lhs.num * rhs.den != rhs.num * lhs.den;
60     }
61     friend bool operator<(const Frac &lhs, const Frac &rhs) {
62         return lhs.num * rhs.den < rhs.num * lhs.den;
63     }
64     friend bool operator>(const Frac &lhs, const Frac &rhs) {
65         return lhs.num * rhs.den > rhs.num * lhs.den;
66     }
67     friend bool operator<=(const Frac &lhs, const Frac &rhs) {
68         return lhs.num * rhs.den <= rhs.num * lhs.den;
69     }
70     friend bool operator>=(const Frac &lhs, const Frac &rhs) {
71         return lhs.num * rhs.den >= rhs.num * lhs.den;
72     }
73     friend std::ostream &operator<<(std::ostream &os, Frac x) {
74         T g = std::gcd(x.num, x.den);
75         if (x.den == g) {
76             return os << x.num / g;
77         } else {
78             return os << x.num / g << "/" << x.den / g;
79         }
80     }
81 };
82
83 using F = Frac<int>;

```

6. STRING

6.1. AC_automaton.h

```

1  struct ACAutomaton
2  {
3      static constexpr int N = 1e6 + 10;
4      int ch[N][26], fail[N], cntNodes;
5      int cnt[N];
6      ACAutomaton()
7      {
8          cntNodes = 1;
9      }
10     void insert(string s)
11     {
12         int u = 1;
13         for (auto c : s)
14         {
15             int &v = ch[u][c - 'a'];
16             if (!v)
17                 v = ++cntNodes;
18             u = v;
19         }
20         cnt[u]++;
21     }
22     void build()
23     {
24         fill(ch[0], ch[0] + 26, 1);
25         queue<int> q;
26         q.push(1);
27         while (!q.empty())
28         {
29             int u = q.front();
30             q.pop();
31             for (int i = 0; i < 26; i++)
32             {
33                 int &v = ch[u][i];
34                 if (!v)
35                     v = ch[fail[u]][i];
36                 else
37                 {
38                     fail[v] = ch[fail[u]][i];
39                     q.push(v);
40                 }
41             }
42         }
43     }
44     LL query(string t)
45     {
46         LL ans = 0;
47         int u = 1;
48         for (auto c : t)
49         {
50             u = ch[u][c - 'a'];
51             for (int v = u; v && ~cnt[v]; v = fail[v])
52             {
53                 ans += cnt[v];

```

```
54         cnt[v] = -1;
55     }
56 }
57 return ans;
58 }
59 };
```

6.2. compress_print.h

```

1  const int N = 1 << 21;
2  static const int mod1 = 1E9 + 7, base1 = 127;
3  static const int mod2 = 1E9 + 9, base2 = 131;
4  vector<int> val1;
5  vector<int> val2;
6  void init(int n = N)
7  {
8      val1.resize(n + 1), val2.resize(n + 2);
9      val1[0] = 1, val2[0] = 1;
10     for (int i = 1; i <= n; i++)
11     {
12         val1[i] = val1[i - 1] * base1 % mod1;
13         val2[i] = val2[i - 1] * base2 % mod2;
14     }
15 }
16
17 string compress(vector<string> in)
18 { // 前后缀压缩
19     vector<int> h1{1};
20     vector<int> h2{1};
21     string ans = "#";
22     for (auto s : in)
23     {
24         s = "#" + s;
25         int st = 0;
26         int chk1 = 0;
27         int chk2 = 0;
28         for (int j = 1; j < s.size() && j < ans.size(); j++)
29         {
30             chk1 = (chk1 * base1 % mod1 + s[j]) % mod1;
31             chk2 = (chk2 * base2 % mod2 + s[j]) % mod2;
32             if ((h1.back() == (h1[ans.size() - 1 - j] * val1[j] % mod1 +
33 % mod1) &&
34             (h2.back() == (h2[ans.size() - 1 - j] * val2[j] % mod2 +
35 % mod2)    )
36                 st = j;
37         }
38         for (int j = st + 1; j < s.size(); j++)
39         {
40             ans += s[j];
41             h1.push_back((h1.back() * base1 % mod1 + s[j]) % mod1);
42             h2.push_back((h2.back() * base2 % mod2 + s[j]) % mod2);
43         }
44     }
45     return ans.substr(1);
46 }

```


6.3. get_occrr.h

```

1  #include <template_overAll.h>
2
3  /*
4   * 找到某一堆短字符串在长字符串中的出现位置
5   *  dira=1 为最早出现的后端点下标  dira=0 为最晚出现的前端点下标
6   *  源字符串 s 长度为|s|, 查找字符串列表中所有字符串长度和为|_s|
7   *  则时间复杂度为 O(max(|_s|log(|_s|),|s|))
8   */
9  class get_occrr
10 {
11 private:
12     string s;
13 public:
14     get_occrr(string _s) { s = _s; }
15     vector<int> locate(vector<string> _s, bool dira = 1)
16     {
17         int n = _s.size();
18         vector<int> occr(n, -1);
19         map<char, vector<pair<int, int>>> gncing;
20         if(dira == 1)
21         {
22             for(int i = 0; i < n; i++)
23                 gncing[_s[i][0]].push_back({i, 0});
24             for(int i = 0; i < s.size(); i++)
25             {
26                 vector<pair<int, int>> gnctmp = gncing[s[i]];
27                 gncing[s[i]].clear();
28                 for(int j = 0; j < gnctmp.size(); j++)
29                 {
30                     if(gnctmp[j].se+1 < _s[gnctmp[j].fi].size())
31                         gncing[_s[gnctmp[j].fi]]
32 [gnctmp[j].se+1].push_back({gnctmp[j].fi, gnctmp[j].se+1});
33                     else occr[gnctmp[j].fi] = i;
34                 }
35             } else {
36                 for(int i = 0; i < n; i++) gncing[_s[i]
37 [_s[i].size()-1]].push_back({i, _s[i].size()-1});
38                 for(int i = s.size()-1; i >= 0; i--)
39                 {
40                     vector<pair<int, int>> gnctmp = gncing[s[i]];
41                     gncing[s[i]].clear();
42                     for(int j = 0; j < gnctmp.size(); j++)
43                     {
44                         if(gnctmp[j].se - 1 >= 0)
45                             gncing[_s[gnctmp[j].fi]]
46 [gnctmp[j].se-1].push_back({gnctmp[j].fi, gnctmp[j].se-1});
47                         else occr[gnctmp[j].fi] = i;
48                     }
49                 }
50             }
51         }
52     }
53     return occr;
54 }
55 };

```

6.4. hash_print.h

```

1  #define int long long
2  const int N = 1 << 21;
3  static const int mod1 = 1E9 + 7, base1 = 127;
4  static const int mod2 = 1E9 + 9, base2 = 131;
5  vector<int> val1;
6  vector<int> val2;
7  using puv = pair<int,int>;
8  void init(int n = N)
9  {
10     val1.resize(n + 1), val2.resize(n + 2);
11     val1[0] = 1, val2[0] = 1;
12     for (int i = 1; i <= n; i++)
13     {
14         val1[i] = val1[i - 1] * base1 % mod1;
15         val2[i] = val2[i - 1] * base2 % mod2;
16     }
17 }
18 class hstring
19 {
20 public:
21     vector<int> h1;
22     vector<int> h2;
23     string s;
24
25     hstring(string s_) : s(s_), h1{0}, h2{0}
26     {
27         build();
28     }
29
30     void build()
31     {
32         for (auto it : s)
33         {
34             h1.push_back((h1.back() * base1 % mod1 + it) % mod1);
35             h2.push_back((h2.back() * base2 % mod2 + it) % mod2);
36         }
37     }
38
39     puv get()
40     { // 输出整串的哈希值
41         return {h1.back(), h2.back()};
42     }
43
44     puv substring(int l, int r)
45     { // 输出子串的哈希值
46         if (l > r) swap(l, r);
47         int ans1 = (mod1 + h1[r + 1] - h1[l] * val1[r - l + 1] % mod1) % mod1;
48         int ans2 = (mod2 + h2[r + 1] - h2[l] * val2[r - l + 1] % mod2) % mod2;
49         return {ans1, ans2};
50     }
51
52     puv modify(int idx, char x)
53     { // 修改 idx 位为 x
54         int n = s.size() - 1;
55         int ans1 = (h1.back() + val1[n - idx] * (x - s[idx]) % mod1) % mod1;
56         int ans2 = (h2.back() + val2[n - idx] * (x - s[idx]) % mod2) % mod2;
57         return {ans1, ans2};

```

```
58     }  
59 };
```

6.5. KMP.h

```

1  #include <template_overAll.h>
2
3  class KMP
4  {
5  private:
6      string s;
7      string inis;
8  public:
9      vector<int> pi;
10     KMP(string _s)
11     {
12         s = _s;
13         inis = s;
14     }
15     void prefix_function()
16     {
17         pi.clear();
18         int n = (int)s.length();
19         pi.resize(n);
20         for (int i = 1; i < n; i++)
21         {
22             int j = pi[i - 1];
23             while (j > 0 && s[i] != s[j])
24                 j = pi[j - 1];
25             if (s[i] == s[j])
26                 j++;
27             pi[i] = j;
28         }
29         return;
30     }
31     vector<int> find_occr(string p)
32     {
33         s = inis;
34         s = p + "#" + s;
35         prefix_function();
36         vector<int> v;
37         for (int i = p.size() + 1; i < s.size(); i++)
38             if (pi[i] == p.size())
39                 v.push_back(i - 2 * p.size());
40         return v;
41     }
42 };

```

6.6. trie_Tree.h

```

1  #include <template_overAll.h>
2
3  class Trie//AC
4  {
5  public:
6      vector<map<char, int>> t;
7      int root = 0;
8      Trie()
9      {
10         t.resize(1);
11     }
12     void addedge(string _s)
13     {
14         int pvidx = root;
15         _s.push_back('-');
16         for (int i = 0; i < _s.size(); i++)
17         {
18             if (t[pvidx].find(_s[i]) != t[pvidx].end())
19             {
20                 pvidx = t[pvidx][_s[i]];
21             }
22             else
23             {
24                 t[pvidx][_s[i]] = t.size();
25                 t.push_back(map<char, int>());
26                 pvidx = t[pvidx][_s[i]];
27             }
28         }
29     }
30     bool ifcmp(string &s)
31     {
32         int pvidx = root;
33         for(int i = 0;i < s.size();i ++)
34         {
35             if(t[pvidx].find(s[i]) != t[pvidx].end()) pvidx = t[pvidx][s[i]];
36             else return 0;
37         }
38         return t[pvidx].find('-') != t[pvidx].end();
39     }
40 };

```

