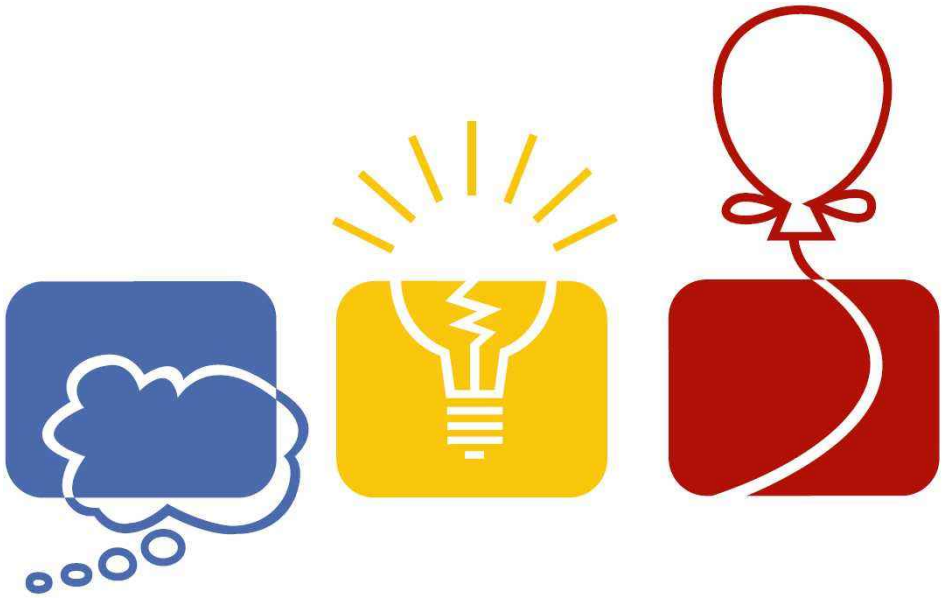

ACM/ICPC Template Manaual

ZheJiang GongShang University

Happy Otaku

November 8, 2019

Contents

0 头文件 **1**

1 字串符 **2**
1.1 KMP . . . 2

2 动态规划 **4**
2.1 01Bag . . . 4
2.2 BagProblem . . . 4
2.3 FullBag . . . 4
2.4 MultiBag . . . 5
2.5 Maze01 . . . 6

3 数据结构 **8**
3.1 BTree . . . 8
3.2 pbds-bbtree . . . 10
3.3 树状数组 . . . 10
3.4 二维树状数组 . . . 12
3.5 线段树 . . . 12
3.6 二维线段树 . . . 15
3.7 树状数组求逆序对 . . . 17
3.8 ST . . . 17
3.9 笛卡尔树 . . . 19
3.10 DancingLinks . . . 19
3.11 静态主席树 . . . 21
3.12 动态主席树 . . . 22
3.13 伸展树 Splay . . . 25

4 图论 **30**
4.1 Graph . . . 30
4.2 Dijkstra . . . 30
4.3 spfa . . . 31
4.4 Dinic . . . 32
4.5 hungry . . . 33
4.6 MinSpanTree . . . 34
4.7 MinCostMaxFlow . . . 35
4.8 ISAP . . . 38
4.9 树链剖分 . . . 40
4.10 倍增 LCA . . . 41
4.11 Tarjan . . . 42
4.12 支配树 . . . 43
4.13 Hopcroft-Karp . . . 44

5 博弈 **46**
5.1 GameProblem . . . 46

6 分治 **47**
6.1 IntegerFastPower . . . 47
6.2 MatrixFastPower . . . 47
6.3 三维 CDQ . . . 48
6.4 树分治-点分治 . . . 48

7 数论 **52**
7.1 线性基 . . . 52
7.2 FWT . . . 54
7.3 FFT . . . 54
7.4 圆上整点 . . . 55

8 其他 **57**
8.1 BigInteger 57
8.2 FastIO 63
8.3 InputOutputSpeedUp 66
8.4 gcd 66
8.5 myItoa 67
8.6 Permutation 67
8.7 prime 67
8.8 Hash 68

0 头文件

```cpp
// 巨菜的ACMer-Happy233

#include <bits/stdc++.h>

using namespace std;

//-----
typedef double db;
typedef long long ll;
typedef unsigned int ui;
typedef vector<int> vi;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define fi first
#define se second
#define pw(x) (1ll << (x))
#define bt(x, k) (((x) >> k) & 1)
#define sz(x) ((int)(x).size())
#define all(x) (x).begin(),(x).end()
#define rep(i, l, r) for(int i=(l);i<(r);++i)
#define per(i, l, r) for(int i=(r)-1;i>=(l);--i)
#define mst(t, v, n) memset(t, v, sizeof(decltype(*(t))) * (n))
#define sf(x) scanf("%d", &(x))
#ifndef ACM_LOCAL
#define endl '\n'
#endif

int main() {
#ifdef ACM_LOCAL
    freopen("./data/std.in", "r", stdin);
    // freopen("./data/std.out", "w", stdout);
#else
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
#endif

#ifdef ACM_LOCAL
    auto start = clock();
#endif
    int t = 1;
//    cin >> t;
    while (t--)
        solve();
#ifdef ACM_LOCAL
    auto end = clock();
    cerr << "Run Time: " << double(end - start) / CLOCKS_PER_SEC << "s" << endl;
#endif
    return 0;
}
```

1 字串符

1.1 KMP

```
template<class elemType>
inline void kmp_nxt(elemType &T, vector<int> &nxt) {
    nxt[0] = -1;
    for (int i = 1; i < T.size(); i++) {
        int j = nxt[i - 1];
        while (j >= 0 && T[i - 1] != T[j]) j = nxt[j];
        if (j >= 0 && T[i - 1] == T[j]) nxt[i] = j + 1;
        else nxt[i] = 0;
    }
}

template<class elemType>
inline int kmp_count(elemType &S, elemType &T) {
    vector<int> nxt(T.size());
    kmp_nxt(T, nxt);
    int index, count = 0;
    for (index = 0; index < S.size(); ++index) {
        int pos = 0;
        int iter = index;
        while (pos < T.size() && iter < S.size()) {
            if (S[iter] == T[pos]) {
                ++iter;
                ++pos;
            } else {
                if (pos == 0) ++iter;
                else pos = nxt[pos - 1] + 1;
            }
        }
        if (pos == T.size() && (iter - index) == T.size()) ++count;
    }
    return count;
}

template<class elemType>
inline void kmp_next(elemType T[], int count, vector<int> &nxt) {
    nxt[0] = -1;
    for (int i = 1; i < count; i++) {
        int j = nxt[i - 1];
        while (j >= 0 && T[i - 1] != T[j]) j = nxt[j];
        if (j >= 0 && T[i - 1] == T[j]) nxt[i] = j + 1;
        else nxt[i] = 0;
    }
}

template<class elemType>
inline int kmp_count(elemType S[], int c1, elemType T[], int c2) {
    vector<int> nxt(c2);
    kmp_nxt(T, c2, nxt);
    int index, count = 0;
    for (index = 0; index < c1; ++index) {
        int pos = 0;
        int iter = index;
        while (pos < c2 && iter < c1) {
            if (S[iter] == T[pos]) {
                ++iter;
```

```
56                 ++pos;
57             }
58             else {
59                 if (pos == 0) ++iter;
60                 else pos = nxt[pos - 1] + 1;
61             }
62         }
63         if (pos == c2 && (iter - index) == c2) ++count;
64     }
65     return count;
66 }
```

2 动态规划

2.1 01Bag

```
void dp(int n, int m) {
    // n=物品个数
    for (int i = 0; i < n; i++) {
        // m=背包最大容量
        for (int j = m; j >= wei[i]; j--)
            // wei=大小 val=价值
            f[j] = max(f[j], f[j - wei[i]] + val[i]);
    }
}
```

2.2 BagProblem

```
#define N 1000
// val=价值 wei=重量 num=数量
int val[N], wei[N], num[N], f[N];
// n=种类个数 m=背包最大值

// 01背包
void dp1(int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = m; j >= wei[i]; j--)
            f[j] = max(f[j], f[j - wei[i]] + val[i]);
    }
}

// 完全背包
void dp2(int n, int m) {
    //初始化看要求
    for (int i = 0; i <= m; i++) {
        f[i] = INF;
    }
    f[0] = 0;
    //若要求恰好装满背包，那在初始化时除了f[0]=0其它f[1..V]均=-∞
    //若没要求背包装满，只希望价格大，初始化时应将f[0..V]=0)
    for (int i = 0; i < n; i++)
        for (int j = wei[i]; j <= m; j++)
            f[j] = max(f[j], f[j - wei[i]] + val[i]);
}

// 多重背包
void dp3(int n, int m) {
    for (int i = 0; i < n; i++)
        for (int k = 0; k < num[i]; k++)
            for (int j = m; j >= wei[i]; j--)
                f[j] = max(f[j], f[j - wei[i]] + val[i]);
}
```

2.3 FullBag

```
/*
完全背包问题的特点是，每种物品可以无限制的重复使用，可以选择放或不放。
完全背包问题描述:
有N物品和一个容量为V的背包。第i件物品的重量是wei[i]，价值是val[i]。
```

```
*/

#include <cstdio>
#define INF 0x3fffffff
#define N 10047
int f[N],val[N],wei[N];
int min(int a,int b)
{
    return x<y?x:y;
}
int main()
{
    int t,i,j,k,E,F,m,n;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d",&E,&F);
        int c = F-E;
        for(i = 0 ; i <= c ; i++)
            f[i]=INF;
        scanf("%d",&n);
        for(i = 0 ; i < n ; i++)
        {
            scanf("%d%d",&val[i],&wei[i]);//val[i]为面额，wei[i]为重量
        }
        f[0]=0;//因为此处假设的是小猪储钱罐 恰好装满 的情况
        //注意初始化（要求恰好装满背包，那么在初始化时除了f[0]为0其它f[1..V]均设为-∞，
        //这样就可以保证最终得到的f[N]是一种恰好装满背包的最优解。
        //如果并没有要求必须把背包装满，而是只希望价格尽量大，初始化时应该将f[0..V]全部设为0）
        for(i =0 ; i < n ; i++)
        {
            for(j = wei[i] ; j <= c ; j++)
            {
                f[j] = min(f[j],f[j-wei[i]]+val[i]);//此处求的是最坏的情况所以用min，确定最少的钱,当然最后就用max了，HEHE
            }
        }
        if(f[c] == INF)
            printf("This is impossible.\n");
        else
            printf("The minimum amount of money in the piggy-bank is %d.\n",f[c]);
    }
    return 0;
}
//此代码为HDU1114;
```

2.4 MultiBag

```
//多重背包(MultiplePack)：有N种物品和一个容量为V的背包。
//第i种物品最多有n[i]件可用，每件费用是c[i]，价值是w[i]。
//求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，
//且价值总和最大。
//HDU 2191

#include <cstdio>
#include <cstring>
#define N 247
int max(int x,int y){
```

```
    return x>y?x:y;
}
int main()  {
    int t,n,m,i,j,k;
    int w[N],pri[N],num[N],f[N];
    while(~scanf("%d",&t)){
        while(t--){
            memset(f,0,sizeof(f));
            scanf("%d%d",&n,&m);//n为总金额，m为大米种类
            for(i = 0 ; i < m ; i++){
                scanf("%d%d%d",&pri[i],&w[i],&num[i]);//num[i]为每种大米的袋数
            }
            for(i = 0 ; i < m ; i++){
                for(k = 0 ; k < num[i] ; k++){
                    for(j = n ; j >= pri[i]; j--){
                        f[j] = max(f[j],f[j-pri[i]]+w[i]);
                    }
                }
            }
            printf("%d\n",f[n]);
        }
    }
    return 0;
}
```

2.5 Maze01

```
struct Maze01 {
    // 原始01矩阵 1-n 1-m
    bool a[N][N];
    // 以(i,j)向上最大高度的最大子矩阵的左右高
    int l[N][N], r[N][N], h[N][N];
    // 最大连续前缀1的左端位置，如果(i,j)为0则ml=j+1
    int ml[N][N];
    // 矩阵大小
    int n, m;

    void prework() {
        for (int i = 1; i <= m; i++) {
            l[0][i] = 0;
            r[0][i] = m;
        }
        for (int i = 1; i <= n; i++) {
            int maxl = 1;
            int maxr = m;
            for (int j = 1; j <= m; j++) {
                if (a[i][j] == 0) {
                    maxl = j + 1;
                    h[i][j] = l[i][j] = 0;
                } else {
                    h[i][j] = h[i - 1][j] + 1;
                    l[i][j] = max(maxl, l[i - 1][j]);
                }
                ml[i - 1][j] = maxl;
                ml[i][j] = j + 1;
            }
            for (int j = m; j >= 1; --j) {
                if (a[i][j] == 0) {
```

```
                    maxr = j - 1;
                    r[i][j] = m;
                } else {
                    r[i][j] = min(maxr, r[i - 1][j]);
                }
            }
        }
    }

    // 单调栈
    pii st[N];

    int calc() {
        prework();
        int ans = 0;
        for (int i = 1; i <= n; i++) {
            int top = 0;
            for (int j = 1; j <= m; j++) {
                if (h[i][j] == 0) continue;
                if (top == 0 || st[top] != make_pair(l[i][j], r[i][j])) {
                    st[++top] = make_pair(l[i][j], r[i][j]);
                }
                while (top && st[top].second == j) {
                    int pos = st[top--].first;
                    if (pos < ml[i][j]) {
                        // l[i][j]-r[i][j]为底，h[i][j]为高
                        // 可以得到所有的唯一子矩阵，不存在maze1完全属于maze2
                        ans++;
                    }
                }
            }
        }
        return ans;
    }
} dp;
```

3 数据结构

3.1 BTree

```
template<class T>

struct TreeNode {
    T value;
    TreeNode *left;
    TreeNode *right;
};

template<class T>
TreeNode<T> *createTree(const T *pre, const T *in, const int len) {
    TreeNode<T> *t = NULL;
    if (len > 0) {
        t = new TreeNode<T>;
        t->value = pre[0];
        int index;
        for (index = 0; index < len; index++) {
            if (in[index] == pre[0]) {
                break;
            }
        }
        if (index == len) {
            index = -1;
        }
        t->left = createTree(pre + 1, in, index);
        t->right = createTree(pre + index + 1, in + index + 1, len - index - 1);
    }
    return t;
}

template<class T>
int preOrder(TreeNode<T> *root, queue<T> &out) {
    if (root) {
        int count = 1;
        out.push(root->value);
        count += preOrder(root->left, out);
        count += preOrder(root->right, out);
        return count;
    } else {
        return 0;
    }
}

template<class T>
int inOrder(TreeNode<T> *root, queue<T> &out) {
    if (root) {
        int count = 1;
        count += inOrder(root->left, out);
        out.push(root->value);
        count += inOrder(root->right, out);
        return count;
    } else {
        return 0;
    }
}

```

```
56  template<class T>
57  void postOrder(TreeNode<T> *root, queue<T> &out) {
58      if (root) {
59          postOrder(root->left, out);
60          postOrder(root->right, out);
61          out.push(root->value);
62      } else {
63          return;
64      }
65  }
66
67  template<class T>
68  T *convertQueueToArray(queue<T> &out, int len) {
69      T *list = new T[len];
70      int now = 0;
71      while (!out.empty() && now < len) {
72          list[now] = out.front();
73          out.pop();
74          now++;
75      }
76      return list;
77  }
78
79  template<class T>
80  void destroyTree(TreeNode<T> *root) {
81      if (root) {
82          destroyTree(root->left);
83          destroyTree(root->right);
84          delete root;
85      } else return;
86  }
87
88  template<class T>
89  void insertIntoBSTree(TreeNode<T> *root, const T &value) {
90      if (!root) {
91          return;
92      }
93      if (value < root->value) {
94          if (root->left) {
95              insertIntoTree(root->left, value);
96          } else {
97              root->left = new TreeNode<T>;
98              root->left->value = value;
99              root->left->left = NULL;
100             root->left->right = NULL;
101         }
102     } else if (value > root->value) {
103         if (root->right) {
104             insertIntoTree(root->right, value);
105         } else {
106             root->right = new TreeNode<T>;
107             root->right->value = value;
108             root->right->left = NULL;
109             root->right->right = NULL;
110         }
111     }
112 }
113
114 template<class T>
```

```
TreeNode<T> *createBSTree(T *list, int len) {
    if (len < 1) {
        return NULL;
    }
    TreeNode<T> *root = new TreeNode<char>;
    root->value = list[0];
    root->left = NULL;
    root->right = NULL;
    for (int i = 1; i < len; i++) {
        insertIntoBSTree(root, list[i]);
    }
    return root;
}
```

3.2 pbds-bbtree

```
// RBTree 红黑树
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
// 红黑树
__gnu_pbds::tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> t;
// null_type无映射(低版本g++为null_mapped_type)
// 类似multiset
__gnu_pbds::tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update> t;
find_by_order(size_t order);
// 结点更新
tree_order_statistics_node_update
insert(p);
erase(it);
// 求k在树中是第几大:
order_of_key(p);
// 找到第order小的迭代器
find_by_order(order);
// 前驱
lower_bound(p);
// 后驱
upper_bound(p);
// 合并
a.join(b);
// 分割 key小于等于v的元素属于a，其余的属于b
a.split(v, b);

// 优先队列
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp>
// 操作类似于stl的优先队列
typedef __gnu_pbds::priority_queue<node, greater<node>, __gnu_pbds::thin_heap_tag> heap
    ;
heap::point_iterator; // 指向元素的指针
```

3.3 树状数组

```
const int N = 1000005;
struct BITree {
    int n;
```

```
    ll c[N];

    void init(int _n) {
        n = _n + 1;
        memset(c, 0, sizeof(ll) * n);
    }

    void change(int pos, ll v) {
        for (int i = pos; i < n; i += i & (-i))
            c[i] += v;
    }

    ll query(int x) {
        ll ans = 0;
        for (int i = x; i > 0; i -= i & (-i))
            ans += c[i];
        return ans;
    }

    void update(int l, int r, ll v) {
        change(l, v);
        change(r + 1, -v);
    }
} tree;

// 区间更新区间查询
struct BITree {
    int n;
    ll c[N], d[N];

    void init(int _n) {
        n = _n + 1;
        memset(c, 0, sizeof(ll) * n);
        memset(d, 0, sizeof(ll) * n);
    }

    void change(int pos, ll v) {
        for (int i = pos; i < n; i += i & (-i))
            c[i] += v, d[i] += v * (pos - 1);
    }

    ll query(int x) {
        ll ans = 0;
        for (int i = x; i > 0; i -= i & (-i))
            ans += x * c[i] - d[i];
        return ans;
    }

    void update(int l, int r, ll v) {
        change(l, v);
        change(r + 1, -v);
    }

    ll sum(int l, int r) {
        return query(r) - query(l - 1);
    }
} tree;
```

3.4 二维树状数组

```
const int N = 2005;

inline int lowbit(const int &x) {
        return x & -x;
    }

struct TdBITree {
    int n, m;
    ll c[N][N];

    void init(int n, int m) {
        this->n = n;
        this->m = m;
        memset(c, 0, sizeof(c))
    }

    void init(int n, int m, ll v) {
        this->n = n;
        this->m = m;
        rep(x, 1, N) {
            rep(y, 1, N) {
                c[x][y] = (x * y + (x - lowbit(x)) * (y - lowbit(y)) - x * (y - lowbit(
    y)) - (x - lowbit(x)) * y) * v;
            }
        }
    }

    void change(int x, int y, ll v) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j))
                c[i][j] += v;
    }

    ll query(int x, int y) {
        ll ans = 0;
        for (int i = x; i >= 1; i -= lowbit(i))
            for (int j = y; j >= 1; j -= lowbit(j))
                ans += c[i][j];
        return ans;
    }

    ll solve(int x1, int y1, int x2, int y2) {
        return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) + query(x1 - 1, y1
     - 1);
    }
};
```

3.5 线段树

```
// hdu 6562
struct TreeNode {
    int l, r;
    int lson, rson;
    ll sum;
    ll len;
    ll laz1, laz2, laz3;
```

```
8
9       inline void init(int a, int b, int ls, int rs) {
10          lson = ls;
11          rson = rs;
12          l = a, r = b;
13          sum = 0;
14          len = 1;
15          laz1 = laz2 = 0;
16          laz3 = 1;
17      }
18
19      inline int mid() {
20          return (l + r) >> 1;
21      }
22
23      inline int width() {
24          return r - l + 1;
25      }
26
27      inline void add(ll val) {
28          sum = (sum * 10 + val * len * 10 + val * width()) % MOD;
29          len = (len * 100) % MOD;
30          // 右懒惰
31          laz1 = (laz1 * 10 + val) % MOD;
32          // 左懒惰
33          laz2 = (laz2 + val * laz3) % MOD;
34          // 长度懒惰
35          laz3 = (laz3 * 10) % MOD;
36      }
37  };
38
39  struct SegTree {
40      int tot;
41      TreeNode node[N << 1];
42
43      inline void init() {
44          tot = 0;
45      }
46
47      inline void up(int k) {
48          TreeNode &nd = node[k];
49          nd.sum = (node[nd.lson].sum + node[nd.rson].sum) % MOD;
50          nd.len = (node[nd.lson].len + node[nd.rson].len) % MOD;
51      }
52
53      inline void push(int k) {
54          TreeNode &nd = node[k];
55          if (nd.laz3 == 1) return;
56          TreeNode &lson = node[nd.lson];
57          TreeNode &rson = node[nd.rson];
58
59          lson.sum = (nd.laz1 * lson.width() + lson.sum * nd.laz3 + nd.laz2 * lson.len %
        MOD * nd.laz3) % MOD;
60          lson.len = (lson.len * nd.laz3 % MOD * nd.laz3) % MOD;
61          lson.laz1 = (lson.laz1 * nd.laz3 + nd.laz1) % MOD;
62          lson.laz2 = (nd.laz2 * lson.laz3 + lson.laz2) % MOD;
63          lson.laz3 = (lson.laz3 * nd.laz3) % MOD;
64
```

```
65          rson.sum = (nd.laz1 * rson.width() + rson.sum * nd.laz3 + nd.laz2 * rson.len %
        MOD * nd.laz3) % MOD;
66          rson.len = (rson.len * nd.laz3 % MOD * nd.laz3) % MOD;
67          rson.laz1 = (rson.laz1 * nd.laz3 + nd.laz1) % MOD;
68          rson.laz2 = (nd.laz2 * rson.laz3 + rson.laz2) % MOD;
69          rson.laz3 = (rson.laz3 * nd.laz3) % MOD;
70
71          nd.laz1 = nd.laz2 = 0;
72          nd.laz3 = 1;
73      }
74
75      void build(int k, int l, int r) {
76          TreeNode &nd = node[k];
77          nd.init(l, r, tot + 1, tot + 2);
78          tot += 2;
79          if (l == r) {
80              return;
81          }
82          int mid = nd.mid();
83          build(nd.lson, nd.l, mid);
84          build(nd.rson, mid + 1, nd.r);
85          up(k);
86      }
87
88      void change(int k, int l, int r, ll val) {
89          TreeNode &nd = node[k];
90          if (nd.l == l && nd.r == r) {
91              nd.add(val);
92              return;
93          }
94          push(k);
95          int mid = nd.mid();
96          if (r <= mid) {
97              change(nd.lson, l, r, val);
98          } else if (l > mid) {
99              change(nd.rson, l, r, val);
100         } else {
101             change(nd.lson, l, mid, val);
102             change(nd.rson, mid + 1, r, val);
103         }
104         up(k);
105     }
106
107     ll query(int k, int l, int r) {
108         TreeNode &nd = node[k];
109         if (nd.l == l && nd.r == r) {
110             return nd.sum;
111         }
112         push(k);
113         int mid = nd.mid();
114         ll ans = 0;
115         if (r <= mid) {
116             ans += query(nd.lson, l, r);
117         } else if (l > mid) {
118             ans += query(nd.rson, l, r);
119         } else {
120             ans += query(nd.lson, l, mid);
121             ans += query(nd.rson, mid + 1, r);
122         }
```

```
        return ans % MOD;
    }
} tree;
```

3.6 二维线段树

```

const int N = 1005;

struct SegTree {

    inline int son(int k, int x) {
        return (k << 2) - 2 + x;
    }

    struct node {
        int l, r;

        node() = default;

        node(int a, int b) : l(a), r(b) {}

        inline int mid() {
            return (l + r) >> 1;
        }

        inline node left() {
            return node(l, mid());
        }

        inline node right() {
            return node(mid() + 1, r);
        }

        inline bool in(int x) {
            return x >= l && x <= r;
        }

        inline bool more() {
            return l < r;
        }

        bool operator==(const node &t) {
            return l == t.l && r == t.r;
        }
    };

    ll c[N << 2][N << 2];
    ll ans[N << 4];
    ll laz[N << 4];

    inline void up(int k, bool x, bool y) {
        int s = (k << 2) - 2;
        ll t = 0;
        if (x) t += ans[s] + ans[s + 1] + laz[s] + laz[s + 1];
        if (y) t += ans[s + 2] + ans[s + 3] + laz[s + 2] + laz[s + 3];
        ans[k] = t;
    }
```

```
53
54      inline void push(int k) {
55          int s = (k << 2) - 2;
56          laz[s] += laz[k];
57          laz[s + 1] += laz[k];
58          laz[s + 2] += laz[k];
59          laz[s + 3] += laz[k];
60          ans[k] += laz[k];
61          laz[k] = 0;
62      }
63
64      void build(node x, node y, int k) {
65          laz[k] = 0;
66          if (x.more() && y.more()) {
67              ans[k] = c[x.l][y.l];
68              return;
69          }
70          ans[k] = 0;
71          bool ax = false;
72          bool ay = false;
73          if (x.more()) {
74              build(x.left(), y, son(k, 0));
75              build(x.right(), y, son(k, 1));
76          }
77          if (y.more()) {
78              build(x, y.left(), son(k, 2));
79              build(x, y.right(), son(k, 3));
80          }
81          up(k, x.more(), y.more());
82      }
83
84      void change(node x, node y, int k, node l, node r, ll v) {
85          if (x == l && y == r) {
86              laz[k] += v;
87              return;
88          }
89          push(k);
90          if (x.more()) {
91              if (l.r <= x.mid()) {
92                  change(x.left(), y, son(k, 0), l, r, v);
93              } else if (l.l > x.mid()) {
94                  change(x.right(), y, son(k, 1), l, r, v);
95              } else {
96                  change(x.left(), y, son(k, 0), node(l.l, x.mid()), r, v);
97                  change(x.right(), y, son(k, 1), node(x.mid() + 1, l.r), r, v);
98              }
99          }
100         if (y.more()) {
101             if (r.l <= y.mid()) {
102                 change(x, y.left(), son(k, 2), l, r, v);
103             } else if (r.r > y.mid()) {
104                 change(x, y.right(), son(k, 3), l, r, v);
105             } else {
106                 change(x, y.left(), son(k, 2), l, node(r.l, y.mid()), v);
107                 change(x, y.right(), son(k, 3), l, node(y.mid() + 1, r.r), v);
108             }
109         }
110         up(k, x.more(), y.more());
111     }
```

```

    ll query(node x, node y, int k, node l, node r) {
        if (x == l && y == r) {
            return ans[k] + laz[k];
        }
        push(k);
        ll t = 0;
        if (x.more()) {
            if (l.r <= x.mid()) {
                t += query(x.left(), y, son(k, 0), l, r);
            } else if (l.l > x.mid()) {
                t += query(x.right(), y, son(k, 1), l, r);
            } else {
                t += query(x.left(), y, son(k, 0), node(l.l, x.mid()), r);
                t += query(x.right(), y, son(k, 1), node(x.mid() + 1, l.r), r);
            }
        }
        if (y.more()) {
            if (r.l <= y.mid()) {
                t += query(x, y.left(), son(k, 2), l, r);
            } else if (r.r > y.mid()) {
                t += query(x, y.right(), son(k, 3), l, r);
            } else {
                t += query(x, y.left(), son(k, 2), l, node(r.l, y.mid()));
                t += query(x, y.right(), son(k, 3), l, node(y.mid() + 1, r.r));
            }
        }
        return t;
    }
};
```

3.7 树状数组求逆序对

```
BITree t;
int n;
pii a[N];

void solve() {
    t.init(n);
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        a[i] = make_pair(x, i);
    }
    sort(a + 1, a + n + 1);
    ll ans = 0;
    for (int i = 1; i <= n; i++) {
        t.change(a[i].second, 1);
        ans += (i - t.query(a[i].second));
    }
    cout << ans << endl;
}
```

3.8 ST

```
// 只需要取值
struct ST {
```

```
3       int ck[N];
4       int dp[20][N];
5
6       void init(int n, int squ[]) {
7           ++n;
8           ck[0] = -1;
9           for (int i = 1; i <= n; i++) {
10              ck[i] = ck[i - 1] + ((i & (i - 1)) == 0 ? 1 : 0);
11          }
12          for (int i = 0; i < n; i++) {
13              dp[0][i] = squ[i];
14          }
15          for (int k = 1; k <= ck[n]; k++) {
16              int dk = k - 1;
17              for (int i = 0; i + (1 << k) - 1 < n; i++) {
18                  dp[k][i] = max(dp[dk][i], dp[dk][i + (1 << dk)]);
19              }
20          }
21      }
22
23      int query(int l, int r) {
24          if (l > r) swap(l, r);
25          int k = ck[r - l + 1];
26          return max(dp[k][l], dp[k][r - (1 << k) + 1]);
27      }
28  };
29
30  // 可得到下标
31  struct ST {
32      int ck[N];
33      int rmq[N];
34      int dp[20][N];
35
36      void init(int n, int squ[]) {
37          ++n;
38          ck[0] = -1;
39          for (int i = 1; i <= n; i++) {
40              ck[i] = ck[i - 1] + ((i & (i - 1)) == 0 ? 1 : 0);
41          }
42          memcpy(rmq, squ, sizeof(int) * n);
43          for (int i = 0; i < n; i++) {
44              dp[0][i] = i;
45          }
46          for (int k = 1; k <= ck[n]; k++) {
47              int dk = k - 1;
48              for (int i = 0; i + (1 << k) - 1 < n; i++) {
49                  int a = dp[dk][i];
50                  int b = dp[dk][i + (1 << dk)];
51                  dp[k][i] = rmq[a] < rmq[b] ? a : b;
52              }
53          }
54      }
55
56      int query(int l, int r) {
57          if (l > r) swap(l, r);
58          int k = ck[r - l + 1];
59          int a = dp[k][l];
60          int b = dp[k][r - (1 << k) + 1];
61          return rmq[a] < rmq[b] ? a : b;
```

```
    }
};
```

3.9 笛卡尔树

```
// 笛卡尔树，静态建树，区间最值跳转
struct CartesianTree {
    int rt; // 根节点
    pii ch[N]; // 左右儿子
    int st[N]; // 单调栈

    void build(int n, int p[]) {
        rt = 0;
        int t = 0;
        for (int i = 1; i <= n; i++) {
            ch[i] = {0, 0};
            // 决定了大于还是小于
            while (t && p[st[t]] > p[i]) --t;
            if (t) {
                // 上一个点的右儿子作为自己的左儿子
                // 成为上一个点的右儿子
                ch[i].first = ch[st[t]].second;
                ch[st[t]].second = i;
            } else { // 自己作为根节点
                ch[i].first = rt;
                rt = i;
            }
            st[++t] = i;
        }
    }
} dika;
```

3.10 DancingLinks

```
// Dancing Links
struct DLX {
    int n, m, size;
    int U[MaxNode], D[MaxNode], L[MaxNode], R[MaxNode], Row[MaxNode], Col[MaxNode];
    int H[MaxN], S[MaxM];
    int ansd, ans[MaxN];

    void init(int _n, int _m) {
        n = _n;
        m = _m;
        for (int i = 0; i <= m; i++) {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i - 1;
            R[i] = i + 1;
        }
        R[m] = 0;
        L[0] = m;
        size = m;
        for (int i = 0; i <= n; i++) {
            H[i] = -1;
        }
    }
```

```
24
25      void Link(int r, int c) {
26          ++S[Col[++size] = c];
27          Row[size] = r;
28          D[size] = D[c];
29          U[D[c]] = size;
30          U[size] = c;
31          D[c] = size;
32          if (H[r] < 0) {
33              H[r] = L[size] = R[size] = size;
34          } else {
35              R[size] = R[H[r]];
36              L[R[H[r]]] = size;
37              L[size] = H[r];
38              R[H[r]] = size;
39          }
40      }
41
42      void remove(int c) {
43          L[R[c]] = L[c];
44          R[L[c]] = R[c];
45          for (int i = D[c]; i != c; i = D[i]) {
46              for (int j = R[i]; j != i; j = R[j]) {
47                  U[D[j]] = U[j];
48                  D[U[j]] = D[j];
49                  --S[Col[j]];
50              }
51          }
52      };
53
54      void resume(int c) {
55          for (int i = U[c]; i != c; i = U[i])
56              for (int j = L[i]; j != i; j = L[j])
57                  ++S[Col[U[D[j]] = D[U[j]] = j]];
58          L[R[c]] = R[L[c]] = c;
59      }
60
61      bool Dance(int d) {
62          if (R[0] == 0) {
63              for (int i = 0; i < d; i++) {
64                  printf("%d%c", ans[i], " \n"[i == d - 1]);
65              }
66              return true;
67          }
68          int c = R[0];
69          for (int i = R[0]; i != 0; i = R[i]) if (S[i] < S[c]) c = i;
70          remove(c);
71          for (int i = D[c]; i != c; i = D[i]) {
72              ans[d] = Row[i];
73              for (int j = R[i]; j != i; j = R[j])remove(Col[j]);
74              if (Dance(d + 1))return true;
75              for (int j = L[i]; j != i; j = L[j])resume(Col[j]);
76          }
77          resume(c);
78          return false;
79      }
80  };
```

3.11 静态主席树

```
// m=update count,MAXN>=m*log(n)
const int N = int(2e5 + 10);
const int MAXN = int(1e7 + 10);

struct PSegTree {
    const int *a;
    pii ran; // Tree Range
    int c[MAXN];
    int tot = 0;
    int lson[MAXN], rson[MAXN];

    int build(int l, int r) {
        int k = ++tot;
        c[k] = 0;
        if (l == r) {
            return k;
        }
        int mid = (l + r) >> 1;
        lson[k] = build(l, mid);
        rson[k] = build(mid + 1, r);
        return k;
    }

    int init(int l, int r, const int num[]) {
        tot = 0;
        a = num;
        ran = {l, r};
        return build(l, r);
    }

    int update(int rt, int p, int v) {
        int k = ++tot, rst = k;
        int l, r;
        tie(l, r) = ran;
        // calc
        c[k] = c[rt] + v;
        while (l < r) {
            int mid = (l + r) >> 1;
            // 下面的逗号表达式顺序不能换
            if (p <= mid) {
                // go left
                rson[k] = rson[rt], rt = lson[rt], k = lson[k] = ++tot;
                r = mid;
            } else {
                // go right
                lson[k] = lson[rt], rt = rson[rt], k = rson[k] = ++tot;
                l = mid + 1;
            }
            // calc
            c[k] = c[rt] + v;
        }
        return rst;
    }

    // r1=right_root,r2=left_root,kth number
    int query(int r1, int r2, int k) {
        int l, r;
```

```
        tie(l, r) = ran;
        while (l < r) {
            int mid = (l + r) >> 1;
            int cnt = c[lson[r1]] - c[lson[r2]];
            if (cnt >= k) {
                r1 = lson[r1], r2 = lson[r2];
                r = mid;
            } else {
                k -= cnt;
                r1 = rson[r1], r2 = rson[r2];
                l = mid + 1;
            }
        }
        return l;
    }

    // r1=right_root,r2=left_root, sum of num<=k
    int sum(int r1, int r2, int k) {
        int l, r;
        tie(l, r) = ran;
        int ans = 0;
        while (l < r) {
            int mid = (l + r) >> 1;
            int cnt = c[lson[r1]] - c[lson[r2]];
            if (k <= mid) {
                r1 = lson[r1], r2 = lson[r2];
                r = mid;
            } else {
                ans += cnt;
                r1 = rson[r1], r2 = rson[r2];
                l = mid + 1;
            }
        }
        ans += c[r1] - c[r2];
        return ans;
    }
} tree;
```

3.12 动态主席树

```
// m: update count,MAXN>=m*log(n)^2
const int N = int(2e5 + 10);
const int MAXN = int(4e7 + 10);
const int LN = 40;

struct PSegTree {
    const int *a;
    pii ran;
    int n;
    int c[MAXN];
    int tot = 0;
    int lson[MAXN], rson[MAXN];
    // t: static root, s: dynamic root
    int t[N], s[N];

    int build(int l, int r) {
        int k = ++tot;
        c[k] = 0;
```

```
19          if (l == r) {
20              return k;
21          }
22          int mid = (l + r) >> 1;
23          lson[k] = build(l, mid);
24          rson[k] = build(mid + 1, r);
25          return k;
26      }
27
28      // SegTree Range and n points, num can be nullptr
29      int init(int l, int r, int _n, const int num[]) {
30          tot = 0;
31          a = num;
32          ran = {l, r};
33          n = _n;
34          int rt = build(l, r);
35          for (int i = 0; i <= n; i++) t[i] = s[i] = rt;
36          return rt;
37      }
38
39      // update the root in k
40      void update(int k[], int rt[], int cnt, int p, int v) {
41          // calc
42          for (int i = 0; i < cnt; i++)
43              c[k[i]] = c[rt[i]] + v;
44          int l, r;
45          tie(l, r) = ran;
46          while (l < r) {
47              int mid = (l + r) >> 1;
48              // 下面的逗号表达式顺序不能换
49              if (p <= mid) {
50                  // go left
51                  for (int i = 0; i < cnt; i++) {
52                      rson[k[i]] = rson[rt[i]], rt[i] = lson[rt[i]], k[i] = lson[k[i]] =
    ++tot;
53                  }
54                  r = mid;
55              } else {
56                  // go right
57                  for (int i = 0; i < cnt; i++) {
58                      lson[k[i]] = lson[rt[i]], rt[i] = rson[rt[i]], k[i] = rson[k[i]] =
    ++tot;
59                  }
60                  l = mid + 1;
61              }
62              // calc
63              for (int i = 0; i < cnt; i++)
64                  c[k[i]] = c[rt[i]] + v;
65          }
66      }
67
68      // build static tree
69      inline void change(int pos, int p, int v) {
70          if (v == 0) { // no change
71              t[pos] = t[pos - 1];
72          } else {
73              // use int as the int[]
74              // must use variable because I use the pointer
75              int rt = t[pos - 1];
```

```
            int k = t[pos] = ++tot;
            update(&k, &rt, 1, p, v);
        }
    }

    int use1[LN], use2[LN];

    // edit dynamic tree
    inline void add(int pos, int p, int v) {
        // memory reuse
        int *k = use1, *rt = use2;
        int cnt = 0;
        for (int i = pos; i <= n; i += (i & -i), cnt++) {
            rt[cnt] = s[i], s[i] = k[cnt] = ++tot;
        }
        update(k, rt, cnt, p, v);
    }

    // calc lson value in use
    inline int sum(int use[], int cnt) {
        int ans = 0;
        for (int i = 0; i < cnt; i++)
            ans += c[lson[use[i]]];
        return ans;
    }

    // calc value in use
    inline int calc(int use[], int cnt) {
        int ans = 0;
        for (int i = 0; i < cnt; i++)
            ans += c[use[i]];
        return ans;
    }

    // ans=p1-p2
    int querySum(int p1, int p2, int k) {
        int r1 = t[p1], r2 = t[p2];
        int cnt1 = 0, cnt2 = 0;
        // calc root in need
        for (int i = p1; i; i -= (i & -i)) use1[cnt1++] = s[i];
        for (int i = p2; i; i -= (i & -i)) use2[cnt2++] = s[i];
        int l, r;
        tie(l, r) = ran;
        int ans = 0;
        while (l < r) {
            int mid = (l + r) >> 1;
            int cnt = c[lson[r1]] - c[lson[r2]] + sum(use1, cnt1) - sum(use2, cnt2);
            if (k <= mid) {
                // go left
                r1 = lson[r1], r2 = lson[r2];
                for (int i = 0; i < cnt1; i++) use1[i] = lson[use1[i]];
                for (int i = 0; i < cnt2; i++) use2[i] = lson[use2[i]];
                r = mid;
            } else {
                // go right
                ans += cnt;
                r1 = rson[r1], r2 = rson[r2];
                for (int i = 0; i < cnt1; i++) use1[i] = rson[use1[i]];
                for (int i = 0; i < cnt2; i++) use2[i] = rson[use2[i]];
```

```
                l = mid + 1;
            }
        }
        int cnt = c[r1] - c[r2] + calc(use1, cnt1) - calc(use2, cnt2);
        ans += cnt;
        return ans;
    }

    // query k
    int query(int p1, int p2, int k) {
        int r1 = t[p1], r2 = t[p2];
        int cnt1 = 0, cnt2 = 0;
        // calc root in need
        for (int i = p1; i; i -= (i & -i)) use1[cnt1++] = s[i];
        for (int i = p2; i; i -= (i & -i)) use2[cnt2++] = s[i];
        int l, r;
        tie(l, r) = ran;
        while (l < r) {
            int mid = (l + r) >> 1;
            int cnt = c[lson[r1]] - c[lson[r2]] + sum(use1, cnt1) - sum(use2, cnt2);
            if (cnt >= k) {
                // go left
                r1 = lson[r1], r2 = lson[r2];
                for (int i = 0; i < cnt1; i++) use1[i] = lson[use1[i]];
                for (int i = 0; i < cnt2; i++) use2[i] = lson[use2[i]];
                r = mid;
            } else {
                // go right
                k -= cnt;
                r1 = rson[r1], r2 = rson[r2];
                for (int i = 0; i < cnt1; i++) use1[i] = rson[use1[i]];
                for (int i = 0; i < cnt2; i++) use2[i] = rson[use2[i]];
                l = mid + 1;
            }
        }
        return l;
    }
} tree;
```

3.13 伸展树 Splay

```
const int N = int(1e6 + 100);

struct Node;
Node *tail, *null;
// 回收栈
Node *st[N], **stop;

struct Node {
    // 内存池
    static Node pool[N];

    Node *ch[2], *fa;
    bool rev, edit;
    int size, val;
    ll sum;
    ll dpl, dpr, dpm; // 左端开始最大，右端开始最大，实际最大

```

```
18      // 初始化内存池
19      static void init() {
20          null = tail = pool;
21          null->clear();
22          stop = st;
23      }
24
25      // 清空当前
26      inline void clear(int _val = 0, int _size = 0) {
27          edit = rev = false;
28          sum = val = _val;
29          size = _size;
30          fa = ch[0] = ch[1] = null;
31          dpl = dpr = dpm = _val;
32      }
33
34      inline void push_up() {
35          if (this == null) return;
36          size = ch[0]->size + ch[1]->size + 1;
37          sum = ch[0]->sum + ch[1]->sum + val;
38          Node *x = ch[0], *y = ch[1];
39          if (size == 1) {
40              dpl = dpr = dpm = val;
41          } else if (y == null) {
42              dpl = max(x->dpl, x->sum + val);
43              dpr = max(1ll * val, val + x->dpr);
44              dpm = max(x->dpm, val + max(x->dpr, 0ll));
45          } else if (x == null) {
46              dpl = max(1ll * val, val + y->dpl);
47              dpr = max(y->dpr, y->sum + val);
48              dpm = max(y->dpm, val + max(y->dpl, 0ll));
49          } else {
50              dpl = max(x->dpl, max(x->sum + val, x->sum + val + y->dpl));
51              dpr = max(y->dpr, max(y->sum + val, y->sum + val + x->dpr));
52              dpm = max(max(x->dpm, y->dpm), val + max(x->dpr, 0ll) + max(y->dpl, 0ll));
53          }
54      }
55
56      // 设置儿子
57      inline void setc(Node *p, int d) { ch[d] = p, p->fa = this; }
58
59      // 获取方向
60      inline bool d() { return fa->ch[1] == this; }
61
62      inline void update_rev() {
63          if (this == null) return;
64          swap(ch[0], ch[1]);
65          swap(dpl, dpr);
66          rev ^= 1;
67      }
68
69      inline void update_val(int v) {
70          if (this == null) return;
71          val = v;
72          sum = 1ll * size * v;
73          edit = 1;
74          dpl = dpr = dpm = max(1ll * val, sum);
75      }
76
```

```
77      inline void push_down() {
78          if (this == null) return;
79          if (rev) {
80              ch[0]->update_rev(), ch[1]->update_rev();
81              rev = 0;
82          }
83          if (edit) {
84              ch[0]->update_val(val), ch[1]->update_val(val);
85              edit = 0;
86          }
87      }
88
89      inline bool isroot() {
90          return fa == null || (this != fa->ch[0] && this != fa->ch[1]);
91      }
92  };
93
94  Node Node::pool[N];
95
96  // 获取第k个
97  Node *get_kth(Node *x, int k) {
98      while (x->ch[0]->size + 1 != k) {
99          x->push_down();
100         if (x->ch[0]->size >= k) {
101             x = x->ch[0];
102         } else {
103             k -= x->ch[0]->size + 1;
104             x = x->ch[1];
105         }
106     }
107     return x;
108 }
109
110 void rotate(Node *x) {
111     Node *f = x->fa, *ff = f->fa;
112     f->push_down();
113     x->push_down();
114     int c = x->d(), cc = f->d();
115     f->setc(x->ch[!c], c);
116     x->setc(f, !c);
117     if (ff->ch[cc] == f) ff->setc(x, cc);
118     else x->fa = ff;
119     f->push_up();
120 }
121
122 void splay(Node *&root, Node *x, Node *goal) {
123     for (Node *f; (f = x->fa) != goal; rotate(x)) {
124         if (f->fa == goal) continue;
125         f->fa->push_down();
126         f->push_down();
127         x->push_down();
128         rotate(x->d() == f->d() ? f : x);
129     }
130     x->push_up();
131     if (goal == null) root = x;
132 }
133
134 int a[N];
135
```

```
136 // 获取有效节点
137 inline Node *getNode() { return stop == st ? ++tail : *--stop; }
138
139 Node *build(int l, int r) {
140     int mid = (l + r) >> 1;
141     Node *root = getNode();
142     root->clear();
143     root->val = a[mid];
144     root->size = 1;
145     if (l < mid) root->setc(build(l, mid - 1), 0);
146     if (r > mid) root->setc(build(mid + 1, r), 1);
147     root->push_up();
148     return root;
149 }
150
151 // l->root, r->root.ch[1]
152 inline Node *make(Node *&root, int l, int r) {
153     Node *x = get_kth(root, l);
154     splay(root, x, null);
155     Node *y = get_kth(root, r);
156     splay(root, y, root);
157     return y;
158 }
159
160 // 插入p子树
161 inline void insert(Node *&root, int l, int r, Node *p) {
162     Node *x = make(root, l, r);
163     x->setc(p, 0);
164     x->push_up();
165     x->fa->push_up();
166 }
167
168 // 回收内存
169 inline void clear(Node *&x) {
170     if (x == null) return;
171     auto now = stop;
172     *stop++ = x;
173     while (now != stop) {
174         x = *now++;
175         if (x->ch[0] != null) *stop++ = x->ch[0];
176         if (x->ch[1] != null) *stop++ = x->ch[1];
177     }
178 }
179
180 // 释放子树
181 inline void del(Node *&root, int l, int r) {
182     Node *x = make(root, l, r);
183     clear(x->ch[0]);
184     x->ch[0] = null;
185     x->push_up();
186     x->fa->push_up();
187 }
188
189 // 转向
190 inline void reverse(Node *&root, int l, int r) {
191     Node *x = make(root, l, r);
192     x->ch[0]->update_rev();
193     x->push_up();
194     x->fa->push_up();
```

```
}

// set value
inline void make_same(Node *&root, int l, int r, int val) {
    Node *x = make(root, l, r);
    x->ch[0]->update_val(val);
    x->push_up();
    x->fa->push_up();
}

// 求和
inline ll sum(Node *&root, int l, int r) {
    Node *x = make(root, l, r);
    return x->ch[0]->sum;
}

// 求最大子串和
inline ll maxsum(Node *&root, int l, int r) {
    Node *x = make(root, l, r);
    return x->ch[0]->dpm;
}

void debug(Node *root) {
    function<void(Node *)> dfs = [&dfs](Node *x) {
        if (x == null) return;
        x->push_down();
        dfs(x->ch[0]);
        cout << x->val << ' ';
        dfs(x->ch[1]);
    };
    dfs(root);
    cout << endl;
}
```

4 图论

4.1 Graph

```
#define forg(i, h, eg) for(int i = (h); ~i; i = (eg[i]).nxt)

struct Edge {
    int e, nxt;
    ll v;
    Edge() = default;
    Edge(int a, ll b, int c = 0) : e(a), v(b), nxt(c) {}

    bool operator<(const Edge &a) const {
        return (a.v == v ? e < a.e : v < a.v);
    }
};

const ll INF = ll(1e11);
const int N = int(1e5 + 10);
const int M = int(3e5 + 10);

struct Graph {
    Edge eg[M];
    int head[N];
    int cnt;

    void init(int n) {
        memset(head, -1, sizeof(int) * ++n);
        cnt = 0;
    }

    inline void addEdge(int x, int y, ll v = 0) {
        eg[cnt] = Edge(y, v, head[x]);
        head[x] = cnt++;
    }
} gh;
```

4.2 Dijkstra

```
int dist[N];
int path[N];

void bfs(int s, int n) {
    n++;
    rep(i, 0, n) dist[i] = INF;
    memset(path, -1, sizeof(int) * n);
    dist[s] = 0;
    path[s] = s;
    // 注意优先队列默认less运算，但选择最大的作为top，注意cmp！！！
    priority_queue<Edge, vector<Edge>, greater<Edge>> q;
    q.push(Edge(s, dist[s]));
    while (!q.empty()) {
        Edge f = q.top();
        q.pop();
        for (int i = gh.head[f.e]; ~i; i = gh.eg[i].nxt) {
            Edge &t = gh.eg[i];
            if (dist[t.e] > f.v + t.v) {
                dist[t.e] = f.v + t.v;
```

```
                path[t.e] = f.e;
                q.push(Edge(t.e, dist[t.e]));
            }
        }
    }
}

#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp>
typedef __gnu_pbds::priority_queue<Edge, greater<Edge>> heap;
// 使用该模板，需要注意因为使用了greater，所以需要重载大于运算
// 默认pairing_heap_tag
// push O(1), pop O(logn) modify O(logn) erase O(logn) join O(1)
// 可选thin_heap_tag
// push O(1), pop O(logn) modify O(1) erase O(logn) join O(n)

heap::point_iterator its[N];
int cnt[N];

void bfs(int s, int n) {
    n++;
    rep(i, 0, n) dist[i] = INF;
    memset(cnt, 0, sizeof(int) * n);
    dist[s] = 0;
    cnt[s] = 1;
    heap q;
    its[s] = q.push(Edge(s, dist[s]));
    while (!q.empty()) {
        Edge f = q.top();
        q.pop();
        for (int i = gh.head[f.e]; ~i; i = gh.eg[i].nxt) {
            Edge &t = gh.eg[i];
            its[t.e] = 0;
            int v = f.v + t.v;
            if (dist[t.e] > v) {
                dist[t.e] = v;
                if (its[t.e] != 0) {
                    q.modify(its[t.e], Edge(t.e, dist[t.e]));
                } else {
                    its[t.e] = q.push(Edge(t.e, dist[t.e]));
                }
                cnt[t.e] = cnt[f.e];
            } else if (dist[t.e] == v) {
                (cnt[t.e] += cnt[f.e]) %= 100003;
            }
        }
    }
}
```

4.3 spfa

```
vector<int> dist;
vector<vector<node>> eg;
vector<int> path;

bool spfa(int n, int start) {
    dist.assign(n, INF);
    dist[start] = 0;
```

```
    deque<int> q;
    q.push_back(start);
    path.assign(n, -1);
    vector<int> cnt(n, 0);
    vector<bool> flag(n, false);
    cnt[start] = flag[start] = true;
    while (!q.empty()) {
        const int now = q.front();
        q.pop_front();
        flag[now] = false;
        for (auto i: eg[now]) {
            if (dist[i.x] > dist[now] + i.d) {
                dist[i.x] = dist[now] + i.d;
                path[i.x] = now;
                if (!flag[i.x]) {
                    if (n == ++cnt[i.x]) return false;
                    //队列非空且优于队首 (SLF)
                    if (!q.empty() && dist[i.x] < dist[q.front()]) {
                        q.push_front(i.x);
                    } else {
                        q.push_back(i.x);
                    }
                    flag[i.x] = true;
                }
            }
        }
    }
    return true;
}
```

4.4 Dinic

```
struct Dinic {
    Graph gh;
    // 点的范围[0, n)
    int n;
    // 弧优化
    int cur[N], dis[N];

    Dinic(){};

    // 设置N
    void init(int _n) {
        n = _n;
        gh.init(n);
    }

    // 加流量
    void addFlow(int x, int y, ll f) {
        gh.addEdge(x, y, f);
        gh.addEdge(y, x, 0);
    }

    bool bfs(int s, int e) {
        memset(dis, -1, sizeof(int) * n);
        int q[N];
        int l, r;
        l = r = 0;
```

```
        dis[s] = 0;
        q[r++] = s;
        while (l < r) {
            int f = q[l++];
            for (int i = gh.head[f]; ~i; i = gh.eg[i].nxt) {
                if (gh.eg[i].v > 0 && dis[gh.eg[i].e] == -1) {
                    dis[gh.eg[i].e] = dis[f] + 1;
                    q[r++] = gh.eg[i].e;
                }
            }
        }
        return dis[e] > 0;
    }

    ll dfs(int s, int e, ll mx) {
        if (s == e || mx == 0) {
            return mx;
        }
        ll flow = 0;
        for (int &k = cur[s]; ~k; k = gh.eg[k].nxt) {
            auto &eg = gh.eg[k];
            ll a;
            if (eg.v > 0 && dis[eg.e] == dis[s] + 1 && (a = dfs(eg.e, e, min(eg.v, mx))
    )) {
                eg.v -= a;
                gh.eg[k ^ 1].v += a;
                flow += a;
                mx -= a;
                if (mx <= 0) break;
            }
        }
        return flow;
    }

    ll max_flow(int s, int e) {
        ll ans = 0;
        while (bfs(s, e)) {
            memcpy(cur, gh.head, sizeof(int) * n);
            ans += dfs(s, e, INF);
        }
        return ans;
    }
} dinic;
```

4.5 hungry

```
#define N 105
#define M 10005
int n, m, k;
pii eg[M * 2];
int result[N * 2];
int head[N * 2];
int cnt = 0;

void addEdge(int x, int y) {
    eg[cnt].first = y;
    eg[cnt].second = head[x];
    head[x] = cnt++;
```

```
}

bool vis[M * 2] = {false};

int dfs(int x) {
    for (int i = head[x]; ~i; i = eg[i].second) {
        int y = eg[i].first;
        if (!vis[y]) {
            vis[y] = true;
            if (result[y] == -1 || dfs(result[y])) {
                result[y] = x;
                return 1;
            }
        }
    }
    return 0;
}

int MaxMatch() {
    int ans = 0;
    memset(result, -1, sizeof(result));
    rep(i, 1, n + 1) {
        memset(vis, 0, sizeof(vis));
        ans += dfs(i);
    }
    return ans;
}

void solve() {
    scanf("%d%d", &m, &k);
    memset(head, -1, sizeof(head));
    cnt = 0;
    rep(i, 0, k) {
        int x, y;
        scanf("%d%d", &x, &y);
        addEdge(x, y);
    }
    int ans = MaxMatch();
    printf("%d\n", ans);
}
```

4.6 MinSpanTree

```
/*
* Prim 求 MST
* 耗费矩阵 cost[][]，标号从 0 开始，0~n-1
* 返回最小生成树的权值，返回 -1 表示原图不连通
 */
const int INF = 0x3f3f3f3f;
const int N = 110;
bool vis[N];
int lowc[N]; //点是 0 n-1
int prim(int cost[][N], int n) {
    int ans = 0;
    memset(vis, false, sizeof(vis));
    vis[0] = true;
    for (int i = 1; i < n; i++)lowc[i] = cost[0][i];
    for (int i = 1; i < n; i++) {
```

```
        int minc = INF;
        int p =-1;
        19
        for (int j = 0; j < n; j++)
            if (!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        if (minc == INF)return -1;//原图不连通
        ans += minc;
        vis[p] = true;
        for (int j = 0; j < n; j++)
            if (!vis[j] && lowc[j] > cost[p][j])
                lowc[j] = cost[p][j];
    }
    return ans;
}
```

4.7 MinCostMaxFlow

```
struct Edge {
    int e, nxt;
    ll flow, cost;

    Edge() {};

    Edge(int a, ll b, ll c, int d = 0) : e(a), flow(b), cost(c), nxt(d) {}
};

const ll INF = 1000000;
const int N = int(1e5 + 10);
const int M = int(1e5 + 10);

//前向星
struct Graph {
    Edge eg[M];
    int head[N];
    int cnt;

    void init(int n) {
        memset(head, -1, sizeof(int) * ++n);
        cnt = 0;
    }

    inline void addEdge(int x, int y, ll v, ll c) {
        eg[cnt] = Edge(y, v, c, head[x]);
        head[x] = cnt++;
    }
};

struct MinCostMaxFlow {
    Graph gh;
    // 点的范围[0, n)
    int n;

    // 设置N
    void init(int _n) {
        n = _n + 1;
```

```
39          gh.init(n);
40      }
41
42      // 加流量，反向是负的花费
43      void addFlow(int x, int y, ll f, ll c) {
44          // printf("%d->%d: %lld\t%lld\n", x, y, f, c); fflush(stdout);
45          gh.addEdge(x, y, f, c);
46          gh.addEdge(y, x, 0, -c);
47      }
48
49      // 该pre存的是边
50      int pre[N];
51      int dis[N];
52      bool vis[N];
53
54      bool spfa(int s, int e) {
55          queue<int> q;
56          for (int i = 0; i < n; i++) {
57              dis[i] = INF;
58              vis[i] = false;
59              pre[i] = -1;
60          }
61          dis[s] = 0;
62          vis[s] = true;
63          q.push(s);
64          while (!q.empty()) {
65              int u = q.front();
66              q.pop();
67              vis[u] = false;
68              for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
69                  Edge &eg = gh.eg[i];
70                  if (eg.flow > 0 && dis[eg.e] > dis[u] + eg.cost) {
71                      dis[eg.e] = dis[u] + eg.cost;
72                      pre[eg.e] = i;
73                      if (!vis[eg.e]) {
74                          vis[eg.e] = true;
75                          q.push(eg.e);
76                      }
77                  }
78              }
79          }
80          return pre[e] != -1;
81      }
82
83      pll cal(int s, int e) {
84          ll flow = 0, cost = 0;
85          while (spfa(s, e)) {
86              ll f = INF;
87              for (int i = pre[e]; ~i; i = pre[gh.eg[i ^ 1].e]) {
88                  f = min(f, gh.eg[i].flow);
89              }
90              for (int i = pre[e]; ~i; i = pre[gh.eg[i ^ 1].e]) {
91                  gh.eg[i].flow -= f;
92                  gh.eg[i ^ 1].flow += f;
93                  cost += gh.eg[i].cost;
94              }
95              flow += f;
96          }
97          return make_pair(flow, cost);
```

```
    }

} network;

// vector图存
struct MinCostMaxFlow {
    vector<Edge> g[N];
    // 点的范围[0, n)
    int n = 0;

    // 设置N
    void init(int _n) {
        rep(i, 0, n) {
            g[i].clear();
        }
        n = _n + 1;
    }

    // 加流量，反向是负的花费
    void addFlow(int x, int y, int f, int c) {
        g[x].push_back(Edge(y, f, c, g[y].size()));
        g[y].push_back(Edge(x, 0, -c, g[x].size() - 1));
    }

    // 该pre存的是(点,边)
    pii pre[N];
    int dis[N];
    bool vis[N];
    int h[N];

    int cnt = 0;

    bool bfs(int s, int e) {
        priority_queue<pii, vector<pii>, greater<pii>> q;
        for (int i = 0; i < n; i++) {
            dis[i] = INF;
            vis[i] = false;
            pre[i] = pii(-1, -1);
        }
        dis[s] = 0;
        q.push(pii(0, s));
        while (!q.empty()) {
            pii f = q.top();
            int u = f.second;
            q.pop();
            if (f.first != dis[u]) continue;
            for (int i = 0; i < sz(g[u]); i++) {
                auto &eg = g[u][i];
                if (eg.flow == 0) continue;
                int v = eg.e;
                int cost = eg.cost + dis[u] + h[u] - h[v];
                if (dis[v] > cost) {
                    cnt++;
                    dis[v] = cost;
                    pre[v] = pii(u, i);
                    q.push(pii(dis[v], v));
                }
            }
        }
```

```
157         for (int i = 0; i < n; i++) {
158             h[i] += dis[i];
159         }
160         return pre[e].second != -1;
161     }
162 
163     pii cal(int s, int e, int limit) {
164         int flow = 0, cost = 0;
165         memset(h, 0, sizeof(int) * n);
166         cnt = 0;
167         while (limit) {
168             if (!bfs(s, e)) break;
169             int f = INF;
170             for (int i = e; ~pre[i].second; i = pre[i].first) {
171                 f = min(f, g[pre[i].first][pre[i].second].flow);
172             }
173             for (int i = e; ~pre[i].second; i = pre[i].first) {
174                 g[pre[i].first][pre[i].second].flow -= f;
175                 g[i][g[pre[i].first][pre[i].second].nxt].flow += f;
176             }
177             cost += f * h[e];
178             flow += f;
179             limit -= f;
180         }
181         return make_pair(flow, cost);
182     }
183 
184 } network;
```

4.8 ISAP

```
1  struct ISAP {
2      Graph gh;
3      // 点的范围[0, n)
4      int n;
5      // 弧优化
6      int cur[N], dis[N];
7      ISAP() {};
8      // 设置N
9      void init(int _n) {
10         n = _n;
11         gh.init(n);
12     }
13 
14     // 加流量
15     inline void addFlow(int x, int y, ll f) {
16         gh.addEdge(x, y, f);
17         gh.addEdge(y, x, 0);
18     }
19 
20     int dep[N]; // 记录距离标号
21     int gap[N]; // gap常数优化
22     int q[N]; // 数组模拟队列
23 
24     void bfs(int s, int e) {
25         memset(dep, -1, sizeof(int) * n);
26         memset(gap, 0, sizeof(int) * n);
27         gap[0] = 1;
```

```
28          dep[e] = 0;
29          int l = 0, r = 0;
30          q[r++] = e;
31          while (l < r) {
32              int u = q[l++];
33              for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
34                  int v = gh.eg[i].e;
35                  if (~dep[v]) continue;
36                  q[r++] = v;
37                  dep[v] = dep[u] + 1;
38                  gap[dep[v]]++;
39              }
40          }
41      }
42
43      ll st[N]; // 栈优化
44
45      ll max_flow(int s, int e) {
46          bfs(s, e);
47          memcpy(cur, gh.head, sizeof(int) * n);
48          int top = 0;
49          int u = s;
50          ll ans = 0;
51          while (dep[s] < N) {
52              if (u == e) {
53                  ll mf = INF;
54                  int sel = 0;
55                  for (int i = 0; i < top; i++) {
56                      if (mf > gh.eg[st[i]].v) {
57                          mf = gh.eg[st[i]].v;
58                          sel = i;
59                      }
60                  }
61
62                  for (int i = 0; i < top; i++) {
63                      gh.eg[st[i]].v -= mf;
64                      gh.eg[st[i] ^ 1].v += mf;
65                  }
66                  ans += mf;
67                  top = sel;
68                  u = gh.eg[st[top] ^ 1].e;
69                  continue;
70              }
71              bool flag = false;
72              int v = 0;
73              for (int i = cur[u]; ~i; i = gh.eg[i].nxt) {
74                  v = gh.eg[i].e;
75                  if (gh.eg[i].v > 0 && dep[v] + 1 == dep[u]) {
76                      flag = true;
77                      cur[u] = i;
78                      break;
79                  }
80              }
81              if (flag) {
82                  st[top++] = cur[u];
83                  u = v;
84                  continue;
85              }
86              int mind = N;
```

```
            for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
                if (gh.eg[i].v > 0 && dep[gh.eg[i].e] < mind) {
                    mind = dep[gh.eg[i].e];
                    cur[u] = i;
                }
            }
            gap[dep[u]]--; // 当前层无法连通，降层
            if (!gap[dep[u]]) return ans; // 断层结束运算
            dep[u] = mind + 1; // 进入更高层
            gap[dep[u]]++;
            if (u != s)  u = gh.eg[st[--top] ^ 1].e;
        }
        return ans;
    }
} isap;
```

4.9 树链剖分

```
struct TreeChain {
    int top[N]; // 链条顶端点ID
    int fa[N]; // 父亲节点
    int son[N]; // 重儿子
    int deep[N]; // 深度
    int num[N]; // 儿子节点数（包括自己）


    int p[N]; // 在线段树中的ID，
    int fp[N]; // 线段树中ID对应的点
    int tot;

    void dfs(int u, int pre, int d) {
        num[u] = 1;
        deep[u] = d;
        fa[u] = pre;
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].e;
            if (v == pre) continue;
            dfs(v, u, d + 1);
            num[u] += num[v];
            if (son[u] == -1 || num[v] > num[son[u]]) {
                son[u] = v;
            }
        }
    }

    void getpos(int u, int sp) {
        top[u] = sp;
        p[u] = tot++;
        fp[p[u]] = u;
        if (son[u] == -1) return;
        getpos(son[u], sp);
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].e;
            if (v == son[u] || v == fa[u]) continue;
            getpos(v, v);
        }
    }

```

```
    void build(int start, int root, int n) {
        memset(son, -1, sizeof(int) * ++n);
        tot = start; // start是线段树中的ID起始数值
        dfs(root, 0, 0);
        getpos(root, root);
    }
} treec;

// 树状数组，如果无需在线查询可以使用差分树
BITree tree;

// 点权修改
void change1(int u, int v, ll val) {
    int f1 = treec.top[u];
    int f2 = treec.top[v];
    while (f1 != f2) {
        if (treec.deep[f1] < treec.deep[f2]) {
            swap(f1, f2);
            swap(u, v);
        }
        tree1.update(treec.p[f1], treec.p[u], val);
        u = treec.fa[f1];
        f1 = treec.top[u];
    }
    if (treec.deep[u] > treec.deep[v]) {
        swap(u, v);
    }
    tree1.update(treec.p[u], treec.p[v], val);
}

// 边权修改
void change2(int u, int v, ll val) {
    int f1 = treec.top[u];
    int f2 = treec.top[v];
    while (f1 != f2) {
        if (treec.deep[f1] < treec.deep[f2]) {
            swap(f1, f2);
            swap(u, v);
        }
        tree2.update(treec.p[f1], treec.p[u], val);
        u = treec.fa[f1];
        f1 = treec.top[u];
    }
    if (treec.deep[u] > treec.deep[v]) {
        swap(u, v);
    }
    tree2.update(treec.p[treec.son[u]], treec.p[v], val);
}
```

4.10 倍增 LCA

```
const int MAX_DEP = 20;

// 倍增2^k的父亲
int fa[N][MAX_DEP];
int dep[N];

// 倍增LCA
```

```cpp
int lca(int u, int v) {
    if (dep[u] > dep[v]) {
        swap(u, v);
    }
    int hu = dep[u], hv = dep[v];
    int tu = u, tv = v;
    for (int det = hv - hu, i = 0; det; det >>= 1, i++) {
        if (det & 1)
            tv = fa[tv][i];
    }
    if (tu == tv) {
        return tu;
    }
    for (int i = MAX_DEP - 1; i >= 0; i--) {
        if (fa[tu][i] == fa[tv][i]) {
            continue;
        }
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}

// 动态更新节点的父亲属性
void lineFa(int u, int v) {
    fa[u][0] = v;
    for (int i = 1; i < MAX_DEP; i++) {
        v = fa[u][i] = fa[v][i - 1];
    }
}
```

4.11 Tarjan

```cpp
int dfn[N], low[N], st[N], belong[N], num[N];
bool inst[N];
int idx, top, scc;

void tarjan(int u) {
    dfn[u] = low[u] = ++idx;
    st[top++] = u;
    inst[u] = true;
    for (int i = gh.head[u]; i != -1; i = gh.eg[i].nxt) {
        int v = gh.eg[i].e;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (inst[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    int v;
    if (dfn[u] == low[u]) {
        scc++;
        do {
            v = st[--top];
            inst[v] = false;
            belong[v] = scc;
            num[scc]++;
```

```
        } while (u != v);
    }
}
```

4.12 支配树

```
const int MAX_DEP = 20;

// 注意0,1点的边界问题
struct DominatorTree {
    int deg[N]; // 入度
    int dep[N]; //
    int dfn[N];
    int st[N];
    int tot;

    // 拓扑序，要保证root是入度为0
    void bfs(Graph &gh, int root) {
        queue<int> q;
        q.push(root);
        tot = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            dfn[u] = ++tot;
            st[tot] = u;
            forg(i, gh.head[u], gh.eg) {
                int v = gh.eg[i].e;
                if ((--deg[v]) == 0) {
                    q.push(v);
                }
            }
        }
    }

    // 倍增2^k的父亲
    int fa[N][MAX_DEP];

    // 倍增LCA
    int lca(int u, int v) {
        if (dep[u] > dep[v]) {
            swap(u, v);
        }
        int hu = dep[u], hv = dep[v];
        int tu = u, tv = v;
        for (int det = hv - hu, i = 0; det; det >>= 1, i++) {
            if (det & 1)
                tv = fa[tv][i];
        }
        if (tu == tv) {
            return tu;
        }
        for (int i = MAX_DEP - 1; i >= 0; i--) {
            if (fa[tu][i] == fa[tv][i]) {
                continue;
            }
            tu = fa[tu][i];
            tv = fa[tv][i];
```

```
        }
        return fa[tu][0];
    }

    // 动态更新节点的父亲属性
    void lineFa(int u, int v) {
        fa[u][0] = v;
        for (int i = 1; i < MAX_DEP; i++) {
            v = fa[u][i] = fa[v][i - 1];
        }
    }

    // 建树，op是gh的反向图，用来寻找其父亲
    void build(Graph &gh, Graph &op, int n, int root) {
        memcpy(deg, gh.deg, sizeof(int) * (n + 1));
        bfs(gh, root);
        for (int k = 1; k <= tot; k++) {
            int u = st[k], fath = -1;
            dep[u] = 0;
            for (int i = op.head[u]; ~i; i = op.eg[i].nxt) {
                int v = op.eg[i].e;
                if (dfn[v] > dfn[u]) continue;
                fath = (fath == -1 ? v : lca(fath, v));
            }
            if (fath == -1) fath = u;
            lineFa(u, fath);
            dep[u] = dep[fath] + 1;
        }
    }
} dtree;
```

4.13 Hopcroft-Karp

```
int dis;
// linkx: x链接的y, linky: y链接的x
int linkx[N], linky[N];
int dx[N], dy[N];
bool vis[N];

bool searchP(int n) {
    queue<int> q;
    dis = INF;
    mst(dx, -1, n);
    mst(dy, -1, n);
    for (int i = 0; i < n; i++) {
        if (linkx[i] == -1) {
            q.push(i);
            dx[i] = 0;
        }
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        if (dx[u] > dis) break;
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].e;
            if (dy[v] == -1) {
                dy[v] = dx[u] + 1;
```

```
                if (linky[v] == -1) {
                    dis = dy[v];
                } else {
                    dx[linky[v]] = dy[v] + 1;
                    q.push(linky[v]);
                }
            }
        }
    }
    return dis != INF;
}

bool dfs(int u) {
    for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
        int v = gh.eg[i].e;
        if (!vis[v] && dy[v] == dx[u] + 1) {
            vis[v] = true;
            if (linky[v] != -1 && dy[v] == dis) continue;
            if (linky[v] == -1 || dfs(linky[v])) {
                linky[v] = u;
                linkx[u] = v;
                return true;
            }
        }
    }
    return false;
}

int MaxMatch(int n) {
    int rst = 0;
    mst(linkx, -1, n);
    mst(linky, -1, n);
    while (searchP(n)) {
        mst(vis, false, n);
        for (int i = 0; i < n; i++) {
            if (linkx[i] == -1 && dfs(i)) {
                rst++;
            }
        }
    }
    return rst;
}
```

5 博弈

5.1 GameProblem

```cpp
// 巴什博奕，是否先手必胜
inline bool bash_game(int n, int m) {
    //一堆东西，n个物品,最多选m个
    return n % (m + 1);
}

// 威佐夫博弈，是否先手必胜
// 有两堆各若干的物品，两人轮流从其中一堆取至少一件物品，至多不限，或从两堆中同时取相同件物品，规定最后取完者胜利。
inline bool wythoff_game(int n, int m) {
    if (n > m) {
        swap(n, m);
    }
    int temp = floor((n2 - n1) * (1 + sqrt(5.0)) / 2.0);
    return temp != n1;
}
// SG函数
#define N 1001
//f[]: 可以取走的石子个数
//sg[]:0~n的SG函数值
int f[N], sg[N], mex[N];

void getSG(int n) {
    int i, j;
    memset(sg, 0, sizeof(sg));
    for (i = 1; i <= n; i++) {
        memset(mex, 0, sizeof(mex));
        for (j = 1; f[j] <= i; j++)
            mex[sg[i - f[j]]] = 1;
        for (j = 0; j <= n; j++) { //求mes{}中未出现的最小的非负整数
            if (mex[j] == 0) {
                sg[i] = j;
                break;
            }
        }
    }
}

// Auti-nim 反尼姆游戏
// 当先拿完所有石子时候输
// 当如下条件时，先手必胜
// □ : 所有堆的石子数均=1，且有偶数堆。
// □ : 至少有一个堆的石子数>1，且石子堆的异或和≠0。
```

6 分治

6.1 IntegerFastPower

```
ll fpow(ll x, ll k) {
    ll base = x, r = 1;
    for (; k; k >>= 1) {
        if (k & 1) r = r * base;
        base = base * base;
    }
    return r;
}
```

6.2 MatrixFastPower

```
#define MAX_N 10
#define mod_num 9973

struct Mat {
    long long mat[MAX_N][MAX_N];
    long long n;
    Mat() {
        memset(mat, 0, sizeof(mat));
        n = 0;
    }
    Mat(long long n) {
        memset(mat, 0, sizeof(mat));
        this->n = n;
    }
    void init() {
        for (int i = 0; i < n; ++i) {
            mat[i][i] = 1;
        }
    }
    Mat(const long long ** list, long long n) {
        this->n = n;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                mat[i][j] = list[i][j];
            }
        }
    }
};

Mat operator * (Mat a, Mat b) {
    long long n = a.n;
    Mat c(n);
    memset(c.mat, 0, sizeof(c.mat));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                c.mat[i][j] += (a.mat[i][k] * b.mat[k][j]) % mod_num;
                c.mat[i][j] %= mod_num;
            }
        }
    }
    return c;
}
```

```

Mat operator ^ (Mat a, int k) {
    long long n = a.n;
    Mat c(n);
    c.init();
    for (; k; k >>= 1) {
        if (k & 1)   c = c * a;
        a = a * a;
    }
    return c;
}
```

6.3 三维 CDQ

```
struct node {
    // time: 时间 | id: 0,1 是否修改 | f: 正负 | x是下标 | y是权值
    int time, id, f, x, y;
};

bool cmp(const node &a, const node &b) {
    return a.x < b.x;
}

int ans[N];
node p[N << 2], et[N << 2];

void cdq(int l, int r) {
    if (l + 1 == r) return;
    int mid = (l + r) >> 1;
    cdq(l, mid), cdq(mid, r);
    int t = l;
    for (int i = mid; i < r; i++) {
        // if edit continue
        if (p[i].id)continue;
        for (; t < mid && p[t].x <= p[i].x; t++) {
            if (p[t].id) tree.change(p[t].y, p[t].f);;
        }
        int f = p[i].f;
        int cnt = tree.query(p[i].y);
        ans[p[i].time] += f * cnt;
    }
    // 逆操作p[t].y
    while (--t >= l) {
        if (p[t].id) tree.change(p[t].y, -p[t].f);
    }
    // 归并排序
    int t1 = l, t2 = mid, k = 0;
    while (t1 < mid && t2 < r) {
        et[k++] = p[t1].x < p[t2].x ? p[t1++] : p[t2++];
    }
    copy(p + t1, p + mid, et + k);
    copy(p + t2, p + r, et + k);
    copy(et, et + (r - l), p + l);
}
```

6.4 树分治-点分治

```cpp
// 题意：n个节点的树，存在边权，范围1e18
// 求任意两点之间点集的子集中两点之间路径异或和为0的个数
// u<v,u'<v',(u',v') ∈  path(u,v),求path(u', v')异或和==0

struct Edge {
    int to, nxt;
    ll w;
};
const int N = int(1e5 + 10);
const int M = N << 1;

struct Grahp {
    int head[N];
    Edge eg[M];
    int tot;

    void init(int n) {
        memset(head, -1, sizeof(int) * ++n);
    }

    inline void addEdge(int u, int v, ll w) {
        eg[tot] = {v, head[u], w};
        head[u] = tot++;
    }
} gh;

bool vis[N];
// q队列，fa祖先，sz是子树大小，smx是子树最大
int q[N], fa[N], sz[N], smx[N];

int froot(int s) {
    int l, r, mn = N, rt = 0;
    q[l = r = 1] = s;
    while (l <= r) {
        int u = q[l++];
        sz[u] = 1;
        smx[u] = 0;
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].to;
            if (v == fa[u] || vis[v]) continue;
            fa[v] = u;
            q[++r] = v;
        }
    }
    // 反向遍历所有点算size
    while (--l) {
        int u = q[l];
        int mx = max(smx[u], r - sz[u]);
        if (mx < mn) mn = mx, rt = u;
        if (l == 1) break; // 根节点没有fa
        sz[fa[u]] += sz[u];
        smx[fa[u]] = max(smx[fa[u]], sz[u]);
    }
    return rt;
}

// sons子树方向节点个数，val根到该节点异或和，gc边后继方向的节点个数
int sons[N], gc[M];
ll val[N];
```

```
ll ans = 0;
int n;

const int MOD = int(1e9 + 7);

ll nums[N];
int cnt[N];

void go(int s, int rt) {
    fa[s] = rt;
    val[s] = 0;
    int l, r;
    // 不计算s
    q[l = r = 0] = s;
    int m = 0;
    while (l <= r) {
        int u = q[l++];
        nums[m++] = val[u];
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].to;
            if (v == fa[u] || vis[v]) continue;
            fa[v] = u;
            q[++r] = v;
            val[v] = val[u] ^ gh.eg[i].w;
            // 这个点方向后面有多少点
            sons[v] = gc[i];
        }
    }
    sort(nums, nums + m);
    m = unique(nums, nums + m) - nums;
    mst(cnt, 0, m);
    // 遍历分支
    for (int j = gh.head[s]; ~j; j = gh.eg[j].nxt) {
        // 分支的根
        int du = gh.eg[j].to;
        if (vis[du]) continue;
        q[l = r = 1] = du;
        while (l <= r) {
            int u = q[l++];
            int k = lower_bound(nums, nums + m, val[u]) - nums;
            (ans += 1ll * sons[u] * cnt[k] % MOD) %= MOD;
            if (val[u] == 0) {
                (ans += 1ll * sons[u] * (n - gc[j]) % MOD) %= MOD;
            }
            for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
                int v = gh.eg[i].to;
                if (v == fa[u] || vis[v]) continue;
                q[++r] = v;
            }
        }
        // 增加这个方向的值
        while (--l) {
            int u = q[l];
            int k = lower_bound(nums, nums + m, val[u]) - nums;
            (cnt[k] += sons[u]) %= MOD;
        }
    }
}

```

```cpp
void work(int u) {
    // 换根
    u = froot(u);
    vis[u] = true;
    go(u, 0);
    for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
        int v = gh.eg[i].to;
        if (vis[v]) continue;
        work(v);
    }
}

// 预处理边后继节点个数
int pdfs(int u, int f) {
    int fg_id = -1;
    int s = 1;
    for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
        int v = gh.eg[i].to;
        if (v == f) { // 记录父边ID
            fg_id = i;
            continue;
        }
        int c = pdfs(v, u);
        gc[i] = c;
        s += c;
    }
    // 存在父边
    if (~fg_id) gc[fg_id] = n - s;
    return s;
}

void solve() {
    while (cin >> n) {
        gh.init(n);
        for (int i = 2; i <= n; i++) {
            int u, v;
            ll w;
            u = i;
            cin >> v >> w;
            gh.addEdge(u, v, w);
            gh.addEdge(v, u, w);
        }
        mst(vis, false, n + 1);
        pdfs(1, 0);
        ans = 0;
        work(1);
        cout << ans << endl;
    }
}
```

7 数论

7.1 线性基

```
#define rep(i, l, r) for(int i=(l);i<(r);++i)
#define per(i, l, r) for(int i=(r)-1;i>=(l);--i)
#define pw(x) (1ll << (x))
#define bt(x, i) ((x >> i) & 1)
const int LN = 61;
struct LB {
    ll d[LN] = {0}, p[LN] = {0};
    int g[LN] = {0};
    int cnt = 0;

    LB() = default;

    ll &operator[](int pos) {
        return d[pos];
    }

    const ll &operator[](int pos) const {
        return d[pos];
    }

    void insert(ll val, int pos) {
        per(i, 0, LN) {
            if (val & pw(i)) {
                if (!d[i]) {
                    d[i] = val;
                    g[i] = pos;
                    return;
                }
                // 贪心保留最右
                if (pos > g[i]) {
                    swap(pos, g[i]);
                    swap(val, d[i]);
                }
                val ^= d[i];
            }
        }
    }

    ll query_max(int l) {
        ll ret = 0;
        per(i, 0, LN) {
            if (g[i] >= l)
                ret = max(ret, ret ^ d[i]);
        }
        return ret;
    }

    ll query_max() {
        ll ret = 0;
        per(i, 0, LN) {
            ret = max(ret, ret ^ d[i]);
        }
        return ret;
    }

```

```
56      ll query_min() {
57          rep(i, 0, LN) {
58              if (d[i]) return d[i];
59          }
60          return 0;
61      }
62
63      bool test(ll x) const {
64          per(i, 0, LN) {
65              if (bt(x, i)) {
66                  if (!d[i]) return false;
67                  x ^= d[i];
68              }
69          }
70          return true;
71      }
72
73      void rebuild() {
74          per(i, 0, LN) {
75              per(j, 0, i) {
76                  if (d[i] & (1LL << j)) d[i] ^= d[j];
77              }
78          }
79          rep(i, 0, LN) {
80              if (d[i]) p[cnt++] = d[i];
81          }
82      }
83
84      ll kth_query(ll k) {
85          int ret = 0;
86          if (k >= pw(cnt)) return -1;
87          per(i, 0, LN) {
88              if (bt(k, 1)) ret ^= p[i];
89          }
90          return ret;
91      }
92  };
93  // 求并集
94  LB operator+(const LB &n1, const LB &n2) {
95      LB ret = n1;
96      per(i, 0, LN)
97          if (n2.d[i])
98              ret.insert(n1.d[i], n1.g[i]);
99      return ret;
100 }
101 // 求交集
102 LB operator^(const LB &n1, const LB &n2) {
103     LB ans = {}, c = n2, d = n2;
104     rep(i, 0, LN) {
105         ll x = n1[i];
106         if (!x) continue;
107         int p = i;
108         ll T = 0;
109         per(j, 0, p + 1) {
110             if (bt(x, j)) {
111                 if (c[j]) {
112                     x ^= c[j];
113                     T ^= d[j];
114                 } else {
```

```
                    p = j;
                    break;
                }
            }
        }
        if (!x) {
            ans[i] = T;
        } else {
            c[p] = x;
            d[p] = T;
        }
    }
    return ans;
}
```

7.2 FWT

```
void FWT(int p[], int n) { // 如果要取模运算记得-x+MOD,*inv
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += (i << 1)) {
            for (int k = 0; k < i; k++) {
                // or{
                p[i + j + k] += +p[j + k];
                // and
                p[j + k] += p[i + j + k];
                // xor
                tie(p[j + k], p[i + j + k]) = make_pair(p[j + k] + p[i + j + k], p[j +
    k] - p[i + j + k]);
            }
        }
    }
}

void IFWT(int p[], int n) { // 如果要取模运算记得-x+MOD,*inv
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += (i << 1)) {
            for (int k = 0; k < i; k++) {
                // or
                p[i + j + k] -= p[j + k];
                // and
                p[j + k] -= p[i + j + k];
                // xor
                tie(p[j + k], p[i + j + k]) = make_pair(p[j + k] + p[i + j + k], p[j +
    k] - p[i + j + k]);
                p[j + k] = p[j + k] / 2, p[i + j + k] = p[i + j + k] / 2;
            }
        }
    }
}
```

7.3 FFT

```
inline int lowbit(int x) { return x & -x; }

int calc(int n) {
    int k = 0;
    while ((1 << k) < n) k++;
```

```
    return k;
}

// FFT
const double pi = acos(-1.0);

const int N = (1 << 20);
using Complex = complex<double>;

void change(Complex p[], int n) {
    int k = calc(n);
    n = 1 << k;
    vector<int> r(n, 0);
    for (int i = 0; i < n; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (k - 1));
    for (int i = 0; i < n; i++) if (i < r[i]) swap(p[i], p[r[i]]);
}

void FFT(Complex p[], int n, int type) {
    change(p, n);
    for (int mid = 1; mid < n; mid <<= 1) { //待合并区间的长度的一半
        Complex wn(cos(pi / mid), type * sin(pi / mid)); //单位根
        for (int R = mid << 1, j = 0; j < n; j += R) { //R是区间的长度，j表示前已经到哪个位置了
            Complex w(1, 0); //幂
            for (int k = 0; k < mid; k++, w = w * wn) { //枚举左半部分
                Complex x = p[j + k], y = w * p[j + mid + k]; //蝴蝶效应
                p[j + k] = x + y;
                p[j + mid + k] = x - y;
            }
        }
    }
}
```

7.4 圆上整点

```
// x^2+y^2=r^2 --> y^2=(r-x)*(r+x)
// d = gcd(r-x,r+x), n=(r-x)/d, m=(r+x)/d, y^2=d^2*m*n
// 因为y^2和d^2为完全平方数，所以 n*m 为完全平方数
// 又因为n,m互质，所以 n=u^2, m =v^2
// 所以 r-y=d*u^2, r+y=d*v^2
// 可得 x=d*(v^2-u^2), y=d*u*v

ll gcd(ll a, ll b) { return !b ? a : gcd(b, a % b); }

inline int work(vector<pll> &p, ll d, ll t) { // d * t == 2 * r
    int sum = 0;
    for (ll u = 1; 2 * u * u < t; u++) {
        ll v = ll(sqrt(t - u * u));
        if (v == u || v * v + u * u != t || gcd(u, v) != 1) continue;
        ll x = d * (v * v - u * u) / 2;
        ll y = d * u * v;
        p.push_back({x, y});
        sum++;
    }
    return sum;
}

int calc(vector<pll> &p, ll r) {
```

```
24      int sum = 0;
25      r <<= 1;
26      for (ll i = 1; i * i <= r; i++) { // sqrt 枚举因子
27          if (r % i) continue;
28          sum += work(p, i, r / i);
29          if (i * i < r) sum += work(p, r / i, i);
30      }
31      for (int i = 0, cnt = p.size(); i < cnt; i++) {
32          int x = p[i].fi, y = p[i].se;
33          p.push_back({x, -y}), p.push_back({-x, y}), p.push_back({-x, -y});
34      }
35      sum <<= 2;
36      r >>= 1;
37      sum += 4;
38      p.push_back({r, 0}), p.push_back({-r, 0}), p.push_back({0, r}), p.push_back({0, -r
        });
39      sort(p.begin(), p.end());
40      return sum;
41  }
```

8 其他

8.1 BigInteger

```
// base and base_digits must be consistent
constexpr int base = 1000000000;
constexpr int base_digits = 9;

struct bigint {
    // value == 0 is represented by empty z
    vector<int> z; // digits

    // sign == 1 <==> value >= 0
    // sign == -1 <==> value < 0
    int sign;

    bigint() : sign(1) {}

    bigint(ll v) { *this = v; }

    bigint &operator=(ll v) {
        sign = v < 0 ? -1 : 1;
        v *= sign;
        z.clear();
        for (; v > 0; v = v / base) z.push_back((int) (v % base));
        return *this;
    }

    bigint(const string &s) { read(s); }

    bigint &operator+=(const bigint &other) {
        if (sign == other.sign) {
            for (int i = 0, carry = 0; i < other.z.size() || carry; ++i) {
                if (i == z.size())
                    z.push_back(0);
                z[i] += carry + (i < other.z.size() ? other.z[i] : 0);
                carry = z[i] >= base;
                if (carry)
                    z[i] -= base;
            }
        } else if (other != 0 /* prevent infinite loop */) {
            *this -= -other;
        }
        return *this;
    }

    friend bigint operator+(bigint a, const bigint &b) { return a += b; }

    bigint &operator-=(const bigint &other) {
        if (sign == other.sign) {
            if (sign == 1 && *this >= other || sign == -1 && *this <= other) {
                for (int i = 0, carry = 0; i < other.z.size() || carry; ++i) {
                    z[i] -= carry + (i < other.z.size() ? other.z[i] : 0);
                    carry = z[i] < 0;
                    if (carry)
                        z[i] += base;
                }
                trim();
            } else {
```

```
56                 *this = other - *this;
57                 this->sign = -this->sign;
58             }
59         } else {
60             *this += -other;
61         }
62         return *this;
63     }
64
65     friend bigint operator-(bigint a, const bigint &b) {
66         return a -= b;
67     }
68
69     bigint &operator*=(int v) {
70         if (v < 0) sign = -sign, v = -v;
71         for (int i = 0, carry = 0; i < z.size() || carry; ++i) {
72             if (i == z.size()) z.push_back(0);
73             ll cur = (ll) z[i] * v + carry;
74             carry = (int) (cur / base);
75             z[i] = (int) (cur % base);
76         }
77         trim();
78         return *this;
79     }
80
81     bigint operator*(int v) const { return bigint(*this) *= v; }
82
83     friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
84         int norm = base / (b1.z.back() + 1);
85         bigint a = a1.abs() * norm;
86         bigint b = b1.abs() * norm;
87         bigint q, r;
88         q.z.resize(a.z.size());
89
90         for (int i = (int) a.z.size() - 1; i >= 0; i--) {
91             r *= base;
92             r += a.z[i];
93             int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()] : 0;
94             int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size() - 1] : 0;
95             int d = (int) (((ll) s1 * base + s2) / b.z.back());
96             r -= b * d;
97             while (r < 0) r += b, --d;
98             q.z[i] = d;
99         }
100
101         q.sign = a1.sign * b1.sign;
102         r.sign = a1.sign;
103         q.trim();
104         r.trim();
105         return {q, r / norm};
106     }
107
108     friend bigint sqrt(const bigint &a1) {
109         bigint a = a1;
110         while (a.z.empty() || a.z.size() % 2 == 1) a.z.push_back(0);
111
112         int n = a.z.size();
113
114         int firstDigit = (int) ::sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
```

```
115         int norm = base / (firstDigit + 1);
116         a *= norm;
117         a *= norm;
118         while (a.z.empty() || a.z.size() % 2 == 1) a.z.push_back(0);
119 
120         bigint r = (ll) a.z[n - 1] * base + a.z[n - 2];
121         firstDigit = (int) ::sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
122         int q = firstDigit;
123         bigint res;
124 
125         for (int j = n / 2 - 1; j >= 0; j--) {
126             for (;; --q) {
127                 bigint r1 = (r - (res * 2 * base + q) * q) * base * base +
128                             (j > 0 ? (ll) a.z[2 * j - 1] * base + a.z[2 * j - 2] : 0);
129                 if (r1 >= 0) {
130                     r = r1;
131                     break;
132                 }
133             }
134             (res *= base) += q;
135 
136             if (j > 0) {
137                 int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size() + 2] : 0;
138                 int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size() + 1] : 0;
139                 int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] : 0;
140                 q = (int) (((ll) d1 * base * base + (ll) d2 * base + d3) / (firstDigit
    * 2));
141             }
142         }
143 
144         res.trim();
145         return res / norm;
146     }
147 
148     bigint operator/(const bigint &v) const {
149         return divmod(*this, v).first;
150     }
151 
152     bigint operator%(const bigint &v) const {
153         return divmod(*this, v).second;
154     }
155 
156     bigint &operator/=(int v) {
157         if (v < 0) sign = -sign, v = -v;
158         for (int i = (int) z.size() - 1, rem = 0; i >= 0; --i) {
159             ll cur = z[i] + rem * (ll) base;
160             z[i] = (int) (cur / v);
161             rem = (int) (cur % v);
162         }
163         trim();
164         return *this;
165     }
166 
167     bigint operator/(int v) const {
168         return bigint(*this) /= v;
169     }
170 
171     int operator%(int v) const {
172         if (v < 0) v = -v;
```

```
173         int m = 0;
174         for (int i = (int) z.size() - 1; i >= 0; --i)
175             m = (int) ((z[i] + m * (ll) base) % v);
176         return m * sign;
177     }
178 
179     bigint &operator*=(const bigint &v) {
180         return *this = *this * v;;
181     }
182 
183     bigint &operator/=(const bigint &v) {
184         return *this = *this / v;
185     }
186 
187     bool operator<(const bigint &v) const {
188         if (sign != v.sign)
189             return sign < v.sign;
190         if (z.size() != v.z.size())
191             return z.size() * sign < v.z.size() * v.sign;
192         for (int i = (int) z.size() - 1; i >= 0; i--)
193             if (z[i] != v.z[i])
194                 return z[i] * sign < v.z[i] * sign;
195         return false;
196     }
197 
198     bool operator>(const bigint &v) const { return v < *this; }
199 
200     bool operator<=(const bigint &v) const { return !(v < *this); }
201 
202     bool operator>=(const bigint &v) const { return !(*this < v); }
203 
204     bool operator==(const bigint &v) const { return !(*this < v) && !(v < *this); }
205 
206     bool operator!=(const bigint &v) const { return *this < v || v < *this; }
207 
208     void trim() {
209         while (!z.empty() && z.back() == 0) z.pop_back();
210         if (z.empty()) sign = 1;
211     }
212 
213     bool isZero() const {
214         return z.empty();
215     }
216 
217     friend bigint operator-(bigint v) {
218         if (!v.z.empty()) v.sign = -v.sign;
219         return v;
220     }
221 
222     bigint abs() const {
223         return sign == 1 ? *this : -*this;
224     }
225 
226     ll longValue() const {
227         ll res = 0;
228         for (int i = (int) z.size() - 1; i >= 0; i--)
229             res = res * base + z[i];
230         return res * sign;
231     }
```

```
232
233     friend bigint gcd(const bigint &a, const bigint &b) {
234         return b.isZero() ? a : gcd(b, a % b);
235     }
236
237     friend bigint lcm(const bigint &a, const bigint &b) {
238         return a / gcd(a, b) * b;
239     }
240
241     void read(const string &s) {
242         sign = 1;
243         z.clear();
244         int pos = 0;
245         while (pos < s.size() && (s[pos] == '-' || s[pos] == '+')) {
246             if (s[pos] == '-') sign = -sign;
247             ++pos;
248         }
249         for (int i = (int) s.size() - 1; i >= pos; i -= base_digits) {
250             int x = 0;
251             for (int j = max(pos, i - base_digits + 1); j <= i; j++)
252                 x = x * 10 + s[j] - '0';
253             z.push_back(x);
254         }
255         trim();
256     }
257
258     friend istream &operator>>(istream &stream, bigint &v) {
259         string s;
260         stream >> s;
261         v.read(s);
262         return stream;
263     }
264
265     friend ostream &operator<<(ostream &stream, const bigint &v) {
266         if (v.sign == -1)
267             stream << '-';
268         stream << (v.z.empty() ? 0 : v.z.back());
269         for (int i = (int) v.z.size() - 2; i >= 0; --i)
270             stream << setw(base_digits) << setfill('0') << v.z[i];
271         return stream;
272     }
273
274     static vector<int> convert_base(const vector<int> &a, int old_digits, int
    new_digits) {
275         vector<ll> p(max(old_digits, new_digits) + 1);
276         p[0] = 1;
277         for (int i = 1; i < p.size(); i++)
278             p[i] = p[i - 1] * 10;
279         vector<int> res;
280         ll cur = 0;
281         int cur_digits = 0;
282         for (int v : a) {
283             cur += v * p[cur_digits];
284             cur_digits += old_digits;
285             while (cur_digits >= new_digits) {
286                 res.push_back(int(cur % p[new_digits]));
287                 cur /= p[new_digits];
288                 cur_digits -= new_digits;
289             }
```

```
290         }
291         res.push_back((int) cur);
292         while (!res.empty() && res.back() == 0) res.pop_back();
293         return res;
294     }
295 
296     typedef vector<ll> vll;
297 
298     static vll karatsubaMultiply(const vll &a, const vll &b) {
299         int n = a.size();
300         vll res(n + n);
301         if (n <= 32) {
302             for (int i = 0; i < n; i++)
303                 for (int j = 0; j < n; j++)
304                     res[i + j] += a[i] * b[j];
305             return res;
306         }
307 
308         int k = n >> 1;
309         vll a1(a.begin(), a.begin() + k);
310         vll a2(a.begin() + k, a.end());
311         vll b1(b.begin(), b.begin() + k);
312         vll b2(b.begin() + k, b.end());
313 
314         vll a1b1 = karatsubaMultiply(a1, b1);
315         vll a2b2 = karatsubaMultiply(a2, b2);
316 
317         for (int i = 0; i < k; i++) a2[i] += a1[i];
318         for (int i = 0; i < k; i++) b2[i] += b1[i];
319 
320         vll r = karatsubaMultiply(a2, b2);
321         for (int i = 0; i < a1b1.size(); i++) r[i] -= a1b1[i];
322         for (int i = 0; i < a2b2.size(); i++) r[i] -= a2b2[i];
323 
324         for (int i = 0; i < r.size(); i++) res[i + k] += r[i];
325         for (int i = 0; i < a1b1.size(); i++) res[i] += a1b1[i];
326         for (int i = 0; i < a2b2.size(); i++) res[i + n] += a2b2[i];
327         return res;
328     }
329 
330     bigint operator*(const bigint &v) const {
331         vector<int> a6 = convert_base(this->z, base_digits, 6);
332         vector<int> b6 = convert_base(v.z, base_digits, 6);
333         vll a(a6.begin(), a6.end());
334         vll b(b6.begin(), b6.end());
335         while (a.size() < b.size()) a.push_back(0);
336         while (b.size() < a.size()) b.push_back(0);
337         while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
338         vll c = karatsubaMultiply(a, b);
339         bigint res;
340         res.sign = sign * v.sign;
341         for (int i = 0, carry = 0; i < c.size(); i++) {
342             ll cur = c[i] + carry;
343             res.z.push_back((int) (cur % 1000000));
344             carry = (int) (cur / 1000000);
345         }
346         res.z = convert_base(res.z, 6, base_digits);
347         res.trim();
348         return res;
```

```
    }
};
```

8.2 FastIO

```
/*
 * FastIO
 * 代码模板 !
 * 如有雷同 !
 * 纯属巧合 !
 */
namespace FastIO {
#define BUF_SIZE 10000000
#define OUT_SIZE 10000000
#define ll long long
    //fread->read
    bool IOerror = 0;

    inline char nc() {
        static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf + BUF_SIZE;
        if (p1 == pend) {
            p1 = buf;
            pend = buf + fread(buf, 1, BUF_SIZE, stdin);
            if (pend == p1) {
                IOerror = 1;
                return -1;
            }
            //{printf("IO error!\n");system("pause");for (;;);exit(0);}
        }
        return *p1++;
    }

    inline bool blank(char ch) { return ch == ' ' || ch == '\n' || ch == '\r' || ch ==
    '\t'; }

    inline void read(int &x) {
        bool sign = 0;
        char ch = nc();
        x = 0;
        for (; blank(ch); ch = nc());
        if (IOerror)return;
        if (ch == '-')sign = 1, ch = nc();
        for (; ch >= '0' && ch <= '9'; ch = nc())x = x * 10 + ch - '0';
        if (sign)x = -x;
    }

    inline void read(ll &x) {
        bool sign = 0;
        char ch = nc();
        x = 0;
        for (; blank(ch); ch = nc());
        if (IOerror)return;
        if (ch == '-')sign = 1, ch = nc();
        for (; ch >= '0' && ch <= '9'; ch = nc())x = x * 10 + ch - '0';
        if (sign)x = -x;
    }

    inline void read(double &x) {
```

```
53          bool sign = 0;
54          char ch = nc();
55          x = 0;
56          for (; blank(ch); ch = nc());
57          if (IOerror)return;
58          if (ch == '-')sign = 1, ch = nc();
59          for (; ch >= '0' && ch <= '9'; ch = nc())x = x * 10 + ch - '0';
60          if (ch == '.') {
61              double tmp = 1;
62              ch = nc();
63              for (; ch >= '0' && ch <= '9'; ch = nc())tmp /= 10.0, x += tmp * (ch - '0')
    ;
64          }
65          if (sign)x = -x;
66      }
67
68      inline void read(char *s) {
69          char ch = nc();
70          for (; blank(ch); ch = nc());
71          if (IOerror)return;
72          for (; !blank(ch) && !IOerror; ch = nc())*s++ = ch;
73          *s = 0;
74      }
75
76      inline void read(char &c) {
77          for (c = nc(); blank(c); c = nc());
78          if (IOerror) {
79              c = -1;
80              return;
81          }
82      }
83
84      //fwrite->write
85      struct Ostream_fwrite {
86          char *buf, *p1, *pend;
87          Ostream_fwrite() {
88              buf = new char[OUT_SIZE];
89              p1 = buf;
90              pend = buf + OUT_SIZE;
91          }
92          void out(char ch) {
93              if (p1 == pend) {
94                  fwrite(buf, 1, OUT_SIZE, stdout);
95                  p1 = buf;
96              }
97              *p1++ = ch;
98          }
99          void print(int x) {
100             static char s[15], *s1;
101             s1 = s;
102             if (!x)*s1++ = '0';
103             if (x < 0)out('-'), x = -x;
104             while (x)*s1++ = x % 10 + '0', x /= 10;
105             while (s1-- != s)out(*s1);
106         }
107         void println(int x) {
108             static char s[15], *s1;
109             s1 = s;
110             if (!x)*s1++ = '0';
```

```
111             if (x < 0)out('-'), x = -x;
112             while (x)*s1++ = x % 10 + '0', x /= 10;
113             while (s1-- != s)out(*s1);
114             out('\n');
115         }
116         void print(ll x) {
117             static char s[25], *s1;
118             s1 = s;
119             if (!x)*s1++ = '0';
120             if (x < 0)out('-'), x = -x;
121             while (x)*s1++ = x % 10 + '0', x /= 10;
122             while (s1-- != s)out(*s1);
123         }
124         void println(ll x) {
125             static char s[25], *s1;
126             s1 = s;
127             if (!x)*s1++ = '0';
128             if (x < 0)out('-'), x = -x;
129             while (x)*s1++ = x % 10 + '0', x /= 10;
130             while (s1-- != s)out(*s1);
131             out('\n');
132         }
133         void print(double x, int y) {
134             static ll mul[] = {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000,
    100000000,
135                                1000000000, 10000000000LL, 100000000000LL, 1000000000000
    LL, 10000000000000LL,
136                                100000000000000LL, 1000000000000000LL, 10000000000000000
    LL, 100000000000000000LL};
137             if (x < -1e-12)out('-'), x = -x;
138             x *= mul[y];
139             ll x1 = (ll) floor(x);
140             if (x - floor(x) >= 0.5)++x1;
141             ll x2 = x1 / mul[y], x3 = x1 - x2 * mul[y];
142             print(x2);
143             if (y > 0) {
144                 out('.');
145                 for (size_t i = 1; i < y && x3 * mul[i] < mul[y]; out('0'), ++i);
146                 print(x3);
147             }
148         }
149         void println(double x, int y) {
150             print(x, y);
151             out('\n');
152         }
153         void print(char *s) { while (*s)out(*s++); }
154         void println(char *s) {
155             while (*s)out(*s++);
156             out('\n');
157         }
158         void flush() {
159             if (p1 != buf) {
160                 fwrite(buf, 1, p1 - buf, stdout);
161                 p1 = buf;
162             }
163         }
164         ~Ostream_fwrite() { flush(); }
165     } Ostream;
166     inline void print(int x) { Ostream.print(x); }
```

```
    inline void println(int x) { Ostream.println(x); }
    inline void print(char x) { Ostream.out(x); }
    inline void println(char x) {
        Ostream.out(x);
        Ostream.out('\n');
    }
    inline void print(ll x) { Ostream.print(x); }
    inline void println(ll x) { Ostream.println(x); }
    inline void print(double x, int y) { Ostream.print(x, y); }
    inline void println(double x, int y) { Ostream.println(x, y); }
    inline void print(char *s) { Ostream.print(s); }
    inline void println(char *s) { Ostream.println(s); }
    inline void println() { Ostream.out('\n'); }
    inline void flush() { Ostream.flush(); }
};
using namespace FastIO;
```

8.3 InputOutputSpeedUp

```


template <class T>
inline bool read(T &x) {
    x = 0;
    char c = getchar();
    if(c == EOF) return false;
    bool f = false;
    for (; !isdigit(c); c = getchar()) f ^= (c == '-');
    for (; isdigit(c); c = getchar()) x = x * 10 + (c - '0');
    x = f ? -x : x;
    return true;
}

template <class T>
inline void write(T x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    T y = 1;
    int len = 1;
    for (; y <= x / 10; y *= 10) ++len;
    for (; len; --len, x %= y, y /= 10) putchar(x / y + '0');
}
```

8.4 gcd

```
ll gcd(ll x, ll y) { // 循环版
    ll t;
    while (y){
        t = x % y;
        x = y;
        y = t;
    }
    return  x;
}

```

```
ll gcd(ll a, ll b) { // 递归版
    return b == 0 ? a : gcd(b, a % b);
}

// 扩展欧几里得
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    ll q = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return q;
}
```

8.5 myItoa

```
char * myItoa(int value, char* result, int base = 10);

char * myItoa(int value, char* result, int base) {
    // check that the base if valid

    if (base < 2 || base > 16) { *result = 0; return result; }
    char* out = result;
    int quotient = abs(value);
    do {
        const int tmp = quotient / base;
        *out = "0123456789abcdef"[quotient - (tmp*base)];
        ++out;
        quotient = tmp;
    } while (quotient);
    // Apply negative sign
    if (value < 0) *out++ = '-';
    std::reverse(result, out);
    *out = 0;
    return result;
}
```

8.6 Permutation

```
// 错排问题
// D(n) = n! [(-1)^2/2! + ⋯ + (-1)^(n-1)/(n-1)! + (-1)^n/n!].
long long table[1000] = {0, 0, 1};
void init() {
    for (int i = 3; i <= 20; i++) {
        table[i] = (i - 1) * (table[i - 1] + table[i - 2]);
    }
}
```

8.7 prime

```
// 普通素数筛
const int PMAX = 1000000;

int prime_count = 0;
bool prime_list[PMAX] = { false };//元素值为0代表是素数
```

```
int prime_table[PMAX] = { 0 };

void initPrime() {
    for (int i = 2; i < PMAX; i++) {
        if (!prime_list[i])
            prime_table[prime_count++] = i;
        for (int j = 0, e = (PMAX - 1) / i;
             j < prime_count && prime_table[j] <= e; j++) {
            prime_list[i * prime_table[j]] = true;
            if (i % prime_table[j] == 0) break;
        }
    }
}


// 可以得到其中一个质因子的素数筛
const int PMAX = 1000005;

int prime_count = 0;
int prime_list[PMAX] = {0}; //元素值为0代表是素数
int prime_table[PMAX] = {0};

void initPrime() {
    for (int i = 2; i < PMAX; i++) {
        if (!prime_list[i])
            prime_list[i] = prime_table[prime_count++] = i;
        for (int j = 0, e = (PMAX - 1) / i, now;
             j < prime_count && prime_table[j] <= e; j++) {
            prime_list[i * now] = now;
            if (i % now == 0) break;
        }
    }
}
```

8.8 Hash

```
struct Hash {
    int num[N];
    int tot;

    void init() { tot = 0; }

    void insert(int x) { num[tot++] = x; }

    void build() {
        sort(num, num + tot);
        tot = unique(num, num + tot) - num;
    }

    inline int operator[](int x) { return lower_bound(num, num + tot, x) - num; }
} hs;
```