

# Contents

<b>1</b>	<b>Reminder</b>	<b>1</b>
1.1	Observations and Tricks	1
1.2	Bug List	1
<b>2</b>	<b>Init (Linux)</b>	<b>1</b>
2.1	vimrc	1
2.2	run.sh	1
<b>3</b>	<b>Basic</b>	<b>1</b>
3.1	PBDS	1
3.2	Random	1
<b>4</b>	<b>Python</b>	<b>2</b>
4.1	I/O	2
4.2	Decimal	2
<b>5</b>	<b>Data Structure</b>	<b>2</b>
5.1	CDQ	2
5.2	Mo's Algorithm	2
5.3	Heavy Light Decomposition	2
5.4	Leftist Heap	3
5.5	Persistent Treap	3
5.6	Li Chao Tree	3
5.7	Time Segment Tree	3
<b>6</b>	<b>DP</b>	<b>4</b>
6.1	Aliens	4
6.2	SOS DP	5
6.3	期望 DP (Expected Value DP)	5
6.4	數位 DP (Digit DP)	5
<b>7</b>	<b>Graph</b>	<b>6</b>
7.1	Blossom	6
7.2	Weighted Blossom	6
7.3	Max Clique	7
7.4	Bellman-Ford + SPFA	7
7.5	BCC - AP	8
7.6	BCC - Bridge	9
7.7	SCC - Tarjan with 2-SAT	9
7.8	Eulerian Path - Undir	9
7.9	Eulerian Path - Dir	10
7.10	Kth Shortest Path	10
7.11	System of Difference Constraints	11
<b>8</b>	<b>String</b>	<b>11</b>
8.1	Rolling Hash	11
8.2	Trie	11
8.3	KMP	11
8.4	Z Value	11
8.5	Manacher	11
8.6	Suffix Array	12
8.7	Suffix Automaton	12
8.8	SA-IS	12
8.9	Minimum Rotation	13
8.10	Aho Corasick	13
<b>9</b>	<b>Geometry</b>	<b>13</b>
9.1	Template	13
9.2	Basic Operations	13
9.3	Lower Concave Hull	14
9.4	Pick's Theorem	14
9.5	Minimum Enclosing Circle	14
9.6	PolyUnion	15
9.7	Minkowski Sum	15
<b>10</b>	<b>Number Theory</b>	<b>16</b>
10.1	Prime Sieve and Defactor	16
10.2	Harmonic Series	16
10.3	Count Number of Divisors	16
10.4	數論分塊	16
10.5	Pollard's rho	16
10.6	Miller Rabin	17
10.7	Discrete Log	17
10.8	Discrete Sqrt	17
10.9	Fast Power	17
10.10	Extend GCD	17
10.11	$\mu + \phi$	18
10.12	Other Formulas	18
10.13	Polynomial	18
10.14	Counting Primes	19
10.15	Linear Sieve for Other Number Theoretic Functions	20
<b>11</b>	<b>Linear Algebra</b>	<b>20</b>
11.1	Gaussian-Jordan Elimination	20
11.2	Determinant	20
<b>12</b>	<b>Flow / Matching</b>	<b>21</b>
12.1	Flow Methods	21
12.2	Dinic	21
12.3	ISAP	22
12.4	Bounded Max Flow	22
12.5	MCMF	23
12.6	Hopcroft-Karp	23
12.7	Cover / Independent Set	24
12.8	Kuhn Munkres	24

<b>13</b>	<b>Combinatorics</b>	<b>24</b>
13.1	Catalan Number	24
13.2	Bertrand's Ballot Theorem	24
13.3	Burnside's Lemma	24
<b>14</b>	<b>Special Numbers</b>	<b>24</b>
14.1	Fibonacci Series	24
14.2	Prime Numbers	24

## 1 Reminder

### 1.1 Observations and Tricks

- Contribution Technique
- 二分圖/Spanning Tree/DFS Tree
- 行、列操作互相獨立
- 奇偶性
- 當  $s, t$  遞增並且  $t = f(s)$ ，對  $s$  二分搜不好做，可以改成對  $t$  二分搜，再算  $f(t)$
- 啟發式合併
- Permutation Normalization (做一些平移對齊兩個 permutation)
- 枚舉  $a_1 \sim a_n$  再枚舉  $a_n \sim a_1$  可以包在一個迴圈
- 兩個凸型函數相加還是凸型函數，相減不一定

### 1.2 Bug List

- 沒開 long long
- 陣列戳出界／陣列開不夠大
- 寫好的函式忘記呼叫
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- <= 打成 <+
- dp[i] 從 dp[i-1] 轉移時忘記特判  $i > 0$
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case
- 線段樹改值懶標初始值不能設為 0
- DFS 的時候不小心覆寫到全域變數
- 浮點數誤差
- unsigned int128
- 多筆測資不能沒讀完直接 return
- 記得刪 cerr
- vector 超級肥，小 vector 請用 array，例如矩陣快速冪

## 2 Init (Linux)

開場流程：

```
1 vim ~/.vimrc
2 mkdir contest && cd contest
3
4 vim template.cpp
5 for c in {A..P}; do
6     cp template.cpp $c.cpp
7 done
8
9 vim run.sh && chmod 777 run.sh
```

### 2.1 vimrc

```
1 syn on
2 set nu rnu ru cul mouse=a
3 set cin et ts=4 sw=4 sts=4
4 set autochdir
5 set clipboard=unnamedplus
6
7 colo koehler
8
9 no <C-h> ^
10 no <C-l> $
11 no ; :
12
13 inoremap {<CR> {<CR>}<Esc>ko
```

### 2.2 run.sh

```
1 #!/bin/bash
2
3 g++ -std=c++17 -O2 -g -fsanitize=undefined,address $1
4 && echo DONE COMPILE || exit 1
5 ./a.out
```

## 3 Basic

### 3.1 PBDS

```
1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;
3
4 // map
5 tree<int, int, less<>, rb_tree_tag,
6     tree_order_statistics_node_update> tr;
7 tr.order_of_key(element);
8 tr.find_by_order(rank);
9
10 // set
11 tree<int, null_type, less<>, rb_tree_tag,
12     tree_order_statistics_node_update> tr;
13 tr.order_of_key(element);
14 tr.find_by_order(rank);
15
16 // priority queue
17 __gnu_pbds::priority_queue<int, less<int> > big_q; //
18 // Big First
19 __gnu_pbds::priority_queue<int, greater<int> > small_q;
20 // Small First
21 q1.join(q2); // join
```

### 3.2 Random

```
1 mt19937 gen(chrono::steady_clock::now().
2     time_since_epoch().count());
3 #define RANDINT(a, b) uniform_int_distribution<int> (a,
4     b)(rng) // inclusive
5 #define RANDLL(a, b) uniform_int_distribution<long long>
6     >(a, b)(rng) // inclusive
7 #define RANDFLOAT(a, b) uniform_real_distribution<float>
8     >(a, b)(rng) // exclusive
9 #define RANDDOUBLE(a, b) uniform_real_distribution<
10     double>(a, b)(rng) // exclusive
11 shuffle(v.begin(), v.end(), gen);
```

## 4 Python

### 4.1 I/O

```
1 import sys
2 input = sys.stdin.readline
3
4 # Input
5 def readInt():
6     return int(input())
7 def readIntList():
8     return list(map(int, input().split()))
9 def readStr():
10    s = input()
11    return list(s[:len(s) - 1])
12 def readInts():
13    return map(int, input().split())
```

### 4.2 Decimal

```
1 from decimal import *
2 getcontext().prec = 500 # precision
3 getcontext().Emax = 500 # 科學記號指數最大值
4 # 將東西轉成 Decimal
5 Decimal(x)
6 Decimal(y)
7 Decimal(0.0)
8 # 運算子跟一般浮點數一樣
9 x *= y
10 # 輸出
11 print(x.quantize(Decimal("0.000001"), rounding=
12     ROUND_HALF_EVEN))
13 # ROUND_CEILING (2.9=>3, -2.1=>-2)
14 # ROUND_FLOOR (2.1=>2, -2.9=>-3)
15 # ROUND_HALF_EVEN (2.5=>2, 3.5=>4)
16 # ROUND_HALF_UP (2.5=>3, -2.5=>-3)
17 # ROUND_HALF_DOWN (2.5=>2, -2.5=>-2)
18 # ROUND_UP (2.1=>3, -2.1=>-3)
19 # ROUND_DOWN (2.9=>2, -2.9=>-2)
```

## 5 Data Structure

### 5.1 CDQ

```
1 // preparation
2 // 1. write a 1-base BIT
3 // 2. init(n, mxc): n is number of elements, mxc = 值域
4 // 3. build info array CDQ.v by yourself (x, y, z, id)
5 // 4. call solve()
6 // => cdq.ans[i] is number of j s.t. (xj <= xi, yj <=
7     yi, zj <= zi) (j != i)
8 struct CDQ {
9     struct info {
10         int x, y, z, id;
11         info(int x, int y, int z, int id):
12             x(x), y(y), z(z), id(id) {}
13         info():
14             x(0), y(0), z(0), id(0) {}
15     };
16     int n, mxc;
17     BIT bit;
18     vector<info> v;
19     vector<ll> ans;
20     void init(int _n, int _mxc) {
21         n = _n; mxc = _mxc;
22         v.assign(n, info{});
23         ans.assign(n, 0);
24     }
25     void dfs(int l, int r) {
26         if (l >= r) return;
27         int mid = (l+r) >> 1;
28         dfs(l, mid); dfs(mid+1, r);
29
30         vector<info> tmp;
31         vector<int> bit_op;
32         int pl = l, pr = mid+1;
33         while (pl <= mid && pr <= r) {
```

```

34     if (v[pl].y <= v[pr].y) {
35         tmp.emplace_back(v[pl]);
36         bit.upd(v[pl].z, 1);
37         bit_op.emplace_back(v[pl].z);
38         pl++;
39     }
40     else {
41         tmp.emplace_back(v[pr]);
42         ans[v[pr].id] += bit.qry(v[pr].z);
43         pr++;
44     }
45 }
46 while (pl <= mid) {
47     tmp.emplace_back(v[pl]);
48     pl++;
49 }
50 while (pr <= r) {
51     tmp.emplace_back(v[pr]);
52     ans[v[pr].id] += bit.qry(v[pr].z);
53     pr++;
54 }
55 for (int i = l, j = 0; i <= r; i++, j++) v[i] = tmp[j];
56 for (auto& op : bit_op) bit.upd(op, -1);
57 }
58 void solve() {
59     bit.init(mxc);
60     sort(v.begin(), v.end(), [&](const info& a,
61         const info& b){
62         return (a.x == b.x ? (a.y == b.y ? (a.z ==
63             b.z ? a.id < b.id : a.z < b.z) : a.y <
64             b.y) : a.x < b.x);
65     });
66     dfs(0, n-1);
67     map<pair<int, pii>, int> s;
68     sort(v.begin(), v.end(), [&](const info& a,
69         const info& b){
70         return a.id > b.id;
71     });
72     for (int i = 0, id = n-1; i < n; i++, id--) {
73         pair<int, pii> tmp = make_pair(v[i].x,
74             make_pair(v[i].y, v[i].z));
75         auto it = s.find(tmp);
76         if (it != s.end())
77             ans[id] += it->second;
78         s[tmp]++;
79     }
80 }
81 }
82 };

```

## 5.2 Mo's Algorithm

```

1 // segments are 0-based
2 ll cur = 0; // current answer
3 int pl = 0, pr = -1;
4 for (auto& qi : Q) {
5     // get (l, r, qid) from qi
6     while (pl < l) del(pl++);
7     while (pl > l) add(--pl);
8     while (pr < r) add(++pr);
9     while (pr > r) del(pr--);
10    ans[qid] = cur;
11 }

```

## 5.3 Heavy Light Decomposition

```

1 // Author: Ian
2 void build(V<int>&v);
3 void modify(int p, int k);
4 int query(int ql, int qr);
5 // Insert [ql, qr) segment tree here
6 inline void solve(){
7     int n, q; cin >> n >> q;
8     V<int> v(n);
9     for (auto& i: v) cin >> i;
10    V<V<int>> e(n);
11    for(int i = 1; i < n; i++){
12        int a, b; cin >> a >> b, a--, b--;
13        e[a].emplace_back(b);
14        e[b].emplace_back(a);

```

```

15    }
16    V<int> d(n, 0), f(n, 0), sz(n, 1), son(n, -1);
17    F<void(int, int)> dfs1 = [&](int x, int pre) {
18        for (auto i: e[x]) if (i != pre) {
19            d[i] = d[x]+1, f[i] = x;
20            dfs1(i, x), sz[x] += sz[i];
21            if (son[x] == -1 || sz[son[x]] < sz[i])
22                son[x] = i;
23        }
24    }; dfs1(0,0);
25    V<int> top(n, 0), dfn(n, -1);
26    F<void(int, int)> dfs2 = [&](int x, int t) {
27        static int cnt = 0;
28        dfn[x] = cnt++, top[x] = t;
29        if (son[x] == -1) return;
30        dfs2(son[x], t);
31        for (auto i: e[x]) if (!dfn[i])
32            dfs2(i,i);
33    }; dfs2(0,0);
34    V<int> dfnv(n);
35    for (int i = 0; i < n; i++)
36        dfnv[dfn[i]] = v[i];
37    build(dfnv);
38    while(q--){
39        int op, a, b, ans; cin >> op >> a >> b;
40        switch(op){
41            case 1:
42                modify(dfn[a-1], b);
43                break;
44            case 2:
45                a--, b--, ans = 0;
46                while (top[a] != top[b]) {
47                    if (d[top[a]] > d[top[b]]) swap(a,b);
48                    ans = max(ans, query(dfn[top[b]], dfn[b]+1));
49                    b = f[top[b]];
50                }
51                if (dfn[a] > dfn[b]) swap(a,b);
52                ans = max(ans, query(dfn[a], dfn[b]+1));
53                cout << ans << endl;
54                break;
55            }
56    }
57 }

```

## 5.4 Leftist Heap

```

1 // Author: Ian
2 // Function: min-heap, with worst-time O(lg n) merge
3 struct node {
4     node *l, *r; int d, v;
5     node(int x): d(1), v(x) { l = r = nullptr; }
6 };
7 static inline int d(node* x) { return x ? x->d : 0; }
8 node* merge(node* a, node* b) {
9     if (!a || !b) return a ? b;
10    if (a->v > b->v) swap(a,b);
11    a->r = merge(a->r, b);
12    if (d(a->l) < d(a->r))
13        swap(a->l, a->r);
14    a->d = d(a->r) + 1;
15    return a;
16 }

```

## 5.5 Persistent Treap

```

1 // Author: Ian
2 struct node {
3     node *l, *r;
4     char c; int v, sz;
5     node(char x = '$'): c(x), v(mt()), sz(1) {
6         l = r = nullptr;
7     }
8     node(node* p) { *this = *p; }
9     void pull() {
10        sz = 1;
11        for (auto i : {l, r})
12            if (i) sz += i->sz;
13    }
14 } arr[maxn], *ptr = arr;
15 inline int size(node* p) { return p ? p->sz : 0; }

```

```

16 node* merge(node* a, node* b) {
17     if (!a || !b) return a ? : b;
18     if (a->v < b->v) {
19         node* ret = new(ptr++) node(a);
20         ret->r = merge(ret->r, b), ret->pull();
21         return ret;
22     }
23     else {
24         node* ret = new(ptr++) node(b);
25         ret->l = merge(a, ret->l), ret->pull();
26         return ret;
27     }
28 }
29 P<node*> split(node* p, int k) {
30     if (!p) return {nullptr, nullptr};
31     if (k >= size(p->l) + 1) {
32         auto [a, b] = split(p->r, k - size(p->l) - 1);
33         node* ret = new(ptr++) node(p);
34         ret->r = a, ret->pull();
35         return {ret, b};
36     }
37     else {
38         auto [a, b] = split(p->l, k);
39         node* ret = new(ptr++) node(p);
40         ret->l = b, ret->pull();
41         return {a, ret};
42     }
43 }

```

## 5.6 Li Chao Tree

```

1 // Author: Ian
2 // Function: For a set of lines L, find the maximum L_i
3 //            (x) in L in O(lg n).
4 typedef long double ld;
5 constexpr int maxn = 5e4 + 5;
6 struct line {
7     ld a, b;
8     ld operator()(ld x) {return a * x + b;}
9 } arr[(maxn + 1) << 2];
10 bool operator<(line a, line b) {return a.a < b.a;}
11 #define m ((l+r)>>1)
12 void insert(line x, int i = 1, int l = 0, int r = maxn) {
13     if (r - l == 1) {
14         if (x(l) > arr[i](l))
15             arr[i] = x;
16         return;
17     }
18     line a = max(arr[i], x), b = min(arr[i], x);
19     if (a(m) > b(m))
20         arr[i] = a, insert(b, i << 1, l, m);
21     else
22         arr[i] = b, insert(a, i << 1 | 1, m, r);
23 }
24 ld query(int x, int i = 1, int l = 0, int r = maxn) {
25     if (x < l || r <= x) return -numeric_limits<ld>::max();
26     if (r - l == 1) return arr[i](x);
27     return max({arr[i](x), query(x, i << 1, l, m), query(
28         x, i << 1 | 1, m, r)});
29 }
30 #undef m

```

## 5.7 Time Segment Tree

```

1 // Author: Ian
2 constexpr int maxn = 1e5 + 5;
3 V<P<int>> arr[(maxn + 1) << 2];
4 V<int> dsu, sz;
5 V<tuple<int, int, int>> his;
6 int cnt, q;
7 int find(int x) {
8     return x == dsu[x] ? x : find(dsu[x]);
9 }
10 inline bool merge(int x, int y) {
11     int a = find(x), b = find(y);
12     if (a == b) return false;
13     if (sz[a] > sz[b]) swap(a, b);

```

```

14     his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
15         sz[a];
16     return true;
17 };
18 inline void undo() {
19     auto [a, b, s] = his.back(); his.pop_back();
20     dsu[a] = a, sz[b] = s;
21 }
22 #define m ((l + r) >> 1)
23 void insert(int ql, int qr, P<int> x, int i = 1, int l
24     = 0, int r = q) {
25     // debug(ql, qr, x); return;
26     if (qr <= l || r <= ql) return;
27     if (ql <= l && r <= qr) {arr[i].push_back(x);
28         return;}
29     if (qr <= m)
30         insert(ql, qr, x, i << 1, l, m);
31     else if (m <= ql)
32         insert(ql, qr, x, i << 1 | 1, m, r);
33     else {
34         insert(ql, qr, x, i << 1, l, m);
35         insert(ql, qr, x, i << 1 | 1, m, r);
36     }
37 }
38 void traversal(V<int>& ans, int i = 1, int l = 0, int r
39     = q) {
40     int opcnt = 0;
41     // debug(i, l, r);
42     for (auto [a, b] : arr[i])
43         if (merge(a, b))
44             opcnt++, cnt--;
45     if (r - l == 1) ans[l] = cnt;
46     else {
47         traversal(ans, i << 1, l, m);
48         traversal(ans, i << 1 | 1, m, r);
49     }
50     while (opcnt--)
51         undo(), cnt++;
52     arr[i].clear();
53 }
54 #undef m
55 inline void solve() {
56     int n, m; cin >> n >> m >> q, q++;
57     dsu.resize(cnt = n), sz.assign(n, 1);
58     iota(dsu.begin(), dsu.end(), 0);
59     // a, b, time, operation
60     unordered_map<ll, V<int>> s;
61     for (int i = 0; i < m; i++) {
62         int a, b; cin >> a >> b;
63         if (a > b) swap(a, b);
64         s[(((ll)a << 32) | b).emplace_back(0);
65     }
66     for (int i = 1; i < q; i++) {
67         int op, a, b;
68         cin >> op >> a >> b;
69         if (a > b) swap(a, b);
70         switch (op) {
71             case 1:
72                 s[(((ll)a << 32) | b).push_back(i);
73                 break;
74             case 2:
75                 auto tmp = s[(((ll)a << 32) | b).back();
76                 s[(((ll)a << 32) | b).pop_back();
77                 insert(tmp, i, P<int> {a, b});
78         }
79     }
80     for (auto [p, v] : s) {
81         int a = p >> 32, b = p & -1;
82         while (v.size()) {
83             insert(v.back(), q, P<int> {a, b});
84             v.pop_back();
85         }
86     }
87     V<int> ans(q);
88     traversal(ans);
89     for (auto i : ans)
90         cout << i << ' ';
91     cout << endl;
92 }

```

## 6 DP

- 區間 DP
  - 狀態:  $dp[l][r]$  = 區間  $[l, r]$  的最佳值/方案數
  - 轉移: 枚舉劃分點  $k$
  - 思考: 是否滿足四邊形不等式、Knuth 優化可加速
- 背包 DP
  - 狀態:  $dp[i][w]$  = 前  $i$  個物品容量  $w$  的最佳值
  - 判斷是 0/1、多重、分組  $\rightarrow$  決定轉移方式
  - 若容量大  $\rightarrow$  bitset / 數學變形 / meet-in-the-middle
- 樹形 DP
  - 狀態:  $dp[u][flag]$  = 子樹  $u$  的最佳值
  - 合併子樹資訊  $\rightarrow$  小到大合併 / 捲積式轉移
  - 注意 reroot 技巧 (dp on tree + dp2 上傳)
- 數位 DP
  - 狀態: ( $pos, tight, property$ )
  - tight 控制是否貼上界
  - property 常為「餘數、數字和、相鄰限制」
- 狀態 DP
  - 狀態:  $dp[mask][last]$
  - 常見於 TSP / Hamiltonian path / 覆蓋問題
  - $n \leq 20$  可做, 否則要容斥 / FFT
- 期望 / 機率 DP
  - 狀態  $E[s]$  = 從狀態  $s$  到終點的期望
  - 式子:  $E[s] = c + \sum P(s \rightarrow s')E[s']$
  - 線性期望: 能拆就拆, 少算分布
  - 輸出 mod  $\rightarrow$  分數化  $\rightarrow$  模逆元
- 計數 DP / 組合數
  - 狀態表示方案數, 常搭配「模數取餘」
  - 若轉移是捲積型  $\rightarrow$  FFT/NTT 加速
  - 若能公式化 (Catalan / Ballot / Stirling)  $\rightarrow$  直接套公式
- 優化 DP
  - 判斷轉移方程  $dp[i] = \min_j(dp[j] + C(j, i))$  的性質
  - 單調性  $\rightarrow$  分治優化
  - 凸性  $\rightarrow$  Convex Hull Trick / 斜率優化
  - 四邊形不等式  $\rightarrow$  Knuth 優化

### 6.1 Aliens

```

1 // Author: Gino
2 // Function: TODO
3 int n; ll k;
4 vector<ll> a;
5 vector<pll> dp[2];
6 void init() {
7     cin >> n >> k;
8     for (auto& d : dp) d.clear(), d.resize(n);
9     a.clear(); a.resize(n);
10    for (auto& i : a) cin >> i;
11 }
12 pll calc(ll p) {
13     dp[0][0] = make_pair(0, 0);
14     dp[1][0] = make_pair(-a[0], 0);
15     for (int i = 1; i < n; i++) {
16         if (dp[0][i-1].first > dp[1][i-1].first + a[i] - p) {
17             dp[0][i] = dp[0][i-1];
18         } else if (dp[0][i-1].first < dp[1][i-1].first + a[i] - p) {
19             dp[0][i] = make_pair(dp[1][i-1].first + a[i] - p,
20                               dp[1][i-1].second+1);
21         } else {
22             dp[0][i] = make_pair(dp[0][i-1].first, min(dp[0][i-1].second, dp[1][i-1].second+1));
23         }
24     }
25     if (dp[0][i-1].first - a[i] > dp[1][i-1].first) {
26         dp[1][i] = make_pair(dp[0][i-1].first - a[i], dp[0][i-1].second);
27     } else if (dp[0][i-1].first - a[i] < dp[1][i-1].first) {
28         dp[1][i] = make_pair(dp[1][i-1].first - a[i], dp[1][i-1].second);
29     }
30 }
31 return dp[0][n-1];
32 }
```

```

26     dp[1][i] = dp[1][i-1];
27 } else {
28     dp[1][i] = make_pair(dp[1][i-1].first, min(dp[0][i-1].second, dp[1][i-1].second+1));
29 }
30 }
31 return dp[0][n-1];
32 }
33 void solve() {
34     ll l = 0, r = 1e7;
35     pll res = calc(0);
36     if (res.second <= k) return cout << res.first << endl, void();
37     while (l < r) {
38         ll mid = (l+r)>>1;
39         res = calc(mid);
40         if (res.second <= k) r = mid;
41         else l = mid+1;
42     }
43     res = calc(l);
44     cout << res.first + k*l << endl;
45 }
```

### 6.2 SOS DP

```

1 // Author: Gino
2 // Function: Solve problems that enumerates subsets of
3 // subsets (3^n => n*2^n)
4 for (int msk = 0; msk < (1<<n); msk++) {
5     for (int i = 1; i <= n; i++) {
6         if (msk & (1<<(i-1))) {
7             // dp[msk][i] = dp[msk][i-1] + dp[msk ^ (1<<(i-1))][i-1];
8         } else {
9             // dp[msk][i] = dp[msk][i-1];
10        }
11    }
12 }
```

### 6.3 期望 DP (Expected Value DP)

- 狀態設計:  $E[s]$  = 從狀態  $s$  出發到終點的期望值
- 列式子:

$$E[s] = (\text{當前代價}) + \sum_{s'} P(s \rightarrow s') \cdot E[s']$$

- 若存在自環, 把  $E[s]$  移到左邊, 整理成

$$(1 - P(s \rightarrow s))E[s] = c + \sum_{s' \neq s} P(s \rightarrow s') \cdot E[s']$$

- 線性期望技巧: 能拆就拆, 避免處理整個分布
- 輸出 mod 時, 分母要用模逆元:  $q^{-1} \equiv q^{M-2} \pmod{M}$  (質數模數)

#### 常見題型

- 擲骰子遊戲 (到達終點的期望步數)
- 隨機遊走 hitting time
- 重複試驗直到成功
- 博弈遊戲的期望值
- 機率 DP: 計算到某步時在某狀態的機率

#### 範例: 擲骰子到 $n$ 格

$$E[i] = 1 + \frac{1}{6} \sum_{d=1}^6 E[i+d], \quad (i < n), \quad E[n] = 0$$

```

1 int main() {
2     int n;
3     cin >> n; // 終點位置
4     // E[i] = 從位置 i 走到終點的期望步數
5 }
```

```

6 // 因為每次最多走 6，所以要開 n+6 以避免越界
7 vector<double> E(n+7, 0.0);
8
9 // 從終點往前推 (backward DP)
10 for(int i=n-1; i>=0; i--){
11     double sum=0;
12     // 期望公式:  $E[i] = 1 + (E[i+1] + \dots + E[i+6]) / 6$ 
13     for(int d=1; d<=6; d++) sum += E[i+d];
14     E[i] = 1 + sum/6.0;
15 }
16
17 // 輸出  $E[0]$ ，即從起點到終點的期望擲骰次數
18 cout << fixed << setprecision(10) << E[0] << "\n";
19 }

```

## 6.4 數位 DP (Digit DP)

- 狀態：( $pos, tight, property$ )
  - $pos$  = 當前處理到第幾位
  - $tight$  = 是否受限於上界  $N$
  - $property$  = 額外屬性 (如數位和、餘數、相鄰限制...)
- 遞迴：枚舉當前位數字，遞迴下一位
- 終止條件： $pos == \text{長度}$  → 回傳屬性是否滿足
- 記憶化： $dp[pos][tight][property]$

常見題型

- 計算  $[0, N]$  中數位和可被  $k$  整除的數字個數
- 不含連續相同數字的數字個數
- 含特定數字次數的數字個數
- 位數和 / 餘數 / mod pattern

範例：計算  $[0, N]$  中數位和  $\bmod k = 0$  的數字個數

$dp[pos][tight][\text{sum mod } k]$

```

1 string s; // N 轉成字串，方便逐位處理
2 int k;    // 除數
3
4 // dp[pos][tight][sum_mod]
5 // pos = 當前處理到哪一位 (0 = 最高位)
6 // tight = 是否仍受限於 N 的數字 (1 = 是, 0 = 否)
7 // sum_mod = 當前數位和 mod k 的值
8 long long dp[20][2][105];
9
10 // 計算：從 pos 開始，tight 狀態下，數位和 mod k =
11 // sum_mod 的方案數
12 long long dfs(int pos, int tight, int sum_mod){
13     // 終止條件：所有位數都處理完
14     if(pos == (int)s.size()){
15         // 若數位和 mod k == 0，算作一個合法數字
16         return (sum_mod % k == 0);
17     }
18     // 記憶化查詢
19     if(dp[pos][tight][sum_mod] != -1)
20         return dp[pos][tight][sum_mod];
21
22     long long res = 0;
23     // 如果 tight = 1，本位數字上限 = N 的該位數字
24     // 如果 tight = 0，本位數字上限 = 9
25     int limit = tight ? (s[pos] - '0') : 9;
26
27     // 枚舉當前位可以填的數字
28     for(int d=0; d<=limit; d++){
29         // 下一位是否仍然 tight?
30         int next_tight = (tight && d==limit);
31         // 更新數位和 mod k
32         int next_mod = (sum_mod + d) % k;
33         res += dfs(pos+1, next_tight, next_mod);
34     }
35
36     // 存結果
37     return dp[pos][tight][sum_mod] = res;
38 }

```

```

38 int main(){
39     long long N;
40     cin >> N >> k;
41     s = to_string(N); // 把 N 轉成字串，方便取每一位
42     memset(dp, -1, sizeof(dp));
43     cout << dfs(0, 1, 0) << "\n"; // 從最高位開始，初始
44         tight=1, sum=0
45 }

```

## 7 Graph

### 7.1 Blossom

```

1 // Author: ckiseki
2 // usage: first build 0-based graph vector<vector<int>>
3 // g on your own
4 // then construct the object: Matching M(g)
5 // => M.ans
6 // to check matching status: M.match[x] (== n if not
7 // matched)
8 struct Matching {
9     queue<int> q; int ans, n;
10    vector<int> fa, s, v, pre, match;
11    int Find(int u) {
12        return u == fa[u] ? u : fa[u] = Find(fa[u]); }
13    int LCA(int x, int y) {
14        static int tk = 0; tk++; x = Find(x); y = Find(y);
15        for (; swap(x, y)) if (x != n) {
16            if (v[x] == tk) return x;
17            v[x] = tk;
18            x = Find(pre[match[x]]);
19        }
20    }
21    void Blossom(int x, int y, int l) {
22        for (; Find(x) != l; x = pre[y]) {
23            pre[x] = y, y = match[x];
24            if (s[y] == 1) q.push(y), s[y] = 0;
25            for (int z: {x, y}) if (fa[z] == z) fa[z] = l;
26        }
27    }
28    bool Bfs(auto &&g, int r) {
29        iota(all(fa), 0); ranges::fill(s, -1);
30        q = queue<int>(); q.push(r); s[r] = 0;
31        for (; !q.empty(); q.pop()) {
32            for (int x = q.front(); int u : g[x])
33                if (s[u] == -1) {
34                    if (pre[u] = x, s[u] = 1, match[u] == n) {
35                        for (int a = u, b = x, last;
36                            b != n; a = last, b = pre[a])
37                            last = match[b], match[b] = a, match[a] = b;
38                    }
39                    return true;
40                }
41            q.push(match[u]); s[match[u]] = 0;
42        }
43        else if (!s[u] && Find(u) != Find(x)) {
44            int l = LCA(u, x);
45            Blossom(x, u, l); Blossom(u, x, l);
46        }
47    }
48    return false;
49 }
50 Matching(auto &&g) : ans(0), n((int)g.size()),
51     fa(n+1), s(n+1), v(n+1), pre(n+1), match(n+1, n) {
52     for (int x = 0; x < n; ++x)
53         if (match[x] == n) ans += Bfs(g, x);
54 } // match[x] == n means not matched
55 }; // test @ yosupo judge

```

### 7.2 Weighted Blossom

```

1 struct WeightGraph {
2     static const int INF=INT_MAX, N=514;
3     struct edge{
4         int u, v, w; edge(){}
5         edge(int ui, int vi, int wi):u(ui),v(vi),w(wi){}
6     };
7     int n, n_x, lab[N*2], match[N*2], slack[N*2], st[N*2];
8     int pa[N*2], flo_from[N*2][N+1], S[N*2], vis[N*2];

```



```

9  edge g[N*2][N*2]; vector<int> flo[N*2]; queue<int> q;
10  int e_delta(const edge &e){
11      return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
12  }
13  void update_slack(int u,int x){
14      if(!slack[x]||e_delta(g[u][x])<
15          e_delta(g[slack[x]][x])) slack[x]=u;
16  }
17  void set_slack(int x){
18      slack[x]=0;
19      for(int u=1;u<=n;++u)
20          if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
21              update_slack(u,x);
22  }
23  void q_push(int x){
24      if(x<=n) q.push(x);
25      else for(size_t i=0;i<flo[x].size();i++)
26          q_push(flo[x][i]);
27  }
28  void set_st(int x,int b){
29      st[x]=b;
30      if(x>n) for(size_t i=0;i<flo[x].size();++i)
31          set_st(flo[x][i],b);
32  }
33  int get_pr(int b,int xr){
34      int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].begin();
35      if(pr%2==1){
36          reverse(flo[b].begin()+1,flo[b].end());
37          return (int)flo[b].size()-pr;
38      }else return pr;
39  }
40  void set_match(int u,int v){
41      match[u]=g[u][v].v; if(u<=n) return; edge e=g[u][v];
42      int xr=flo_from[u][e.u],pr=get_pr(u,xr);
43      for(int i=0;i<pr;++i)
44          set_match(flo[u][i],flo[u][i^1]);
45      set_match(xr,v); rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end());
46  }
47  void augment(int u,int v){
48      for(;;){
49          int xnv=st[match[u]]; set_match(u,v);
50          if(!xnv) return;
51          set_match(xnv,st[pa[xnv]]); u=st[pa[xnv]],v=xnv;
52      }
53  }
54  int get_lca(int u,int v){
55      static int t=0;
56      for(++t;u||v;swap(u,v)){
57          if(u==0) continue; if(vis[u]==t) return u;
58          vis[u]=t; u=st[match[u]]; if(u) u=st[pa[u]];
59      }
60      return 0;
61  }
62  void add_blossom(int u,int lca,int v){
63      int b=n+1; while(b<=n_x&&st[b])++b; if(b>n_x)++n_x;
64      lab[b]=0,S[b]=0; match[b]=match[lca];
65      flo[b].clear(); flo[b].push_back(lca);
66      for(int x=u,y,x!=lca;x=st[pa[y]])
67          flo[b].push_back(x),
68          flo[b].push_back(y=st[match[x]]),q_push(y);
69      reverse(flo[b].begin()+1,flo[b].end());
70      for(int x=v,y,x!=lca;x=st[pa[y]])
71          flo[b].push_back(x),
72          flo[b].push_back(y=st[match[x]]),q_push(y);
73      set_st(b,b);
74      for(int x=1;x<=n_x;++x) g[b][x].w=g[x][b].w=0;
75      for(int x=1;x<=n;++x) flo_from[b][x]=0;
76      for(size_t i=0;i<flo[b].size();++i){
77          int xs=flo[b][i];
78          for(int x=1;x<=n_x;++x)
79              if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(
80                  g[b][x])) g[b][x]=g[xs][x],g[x][b]=g[x][xs];
81          for(int x=1;x<=n;++x)
82              if(flo_from[xs][x]flo_from[b][x]=xs;
83      }
84      set_slack(b);
85  }
86  void expand_blossom(int b){
87      for(size_t i=0;i<flo[b].size();++i)
88          set_st(flo[b][i],flo[b][i]);
89      int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
90      for(int i=0;i<pr;i+=2){
91          int xs=flo[b][i],xns=flo[b][i+1];
92          pa[xs]=g[xns][xs].u; S[xs]=1,S[xns]=0;
93          slack[xs]=0,set_slack(xns); q_push(xns);
94      }
95      S[xr]=1,pa[xr]=pa[b];
96      for(size_t i=pr+1;i<flo[b].size();++i){
97          int xs=flo[b][i]; S[xs]=-1,set_slack(xs);
98      }
99      st[b]=0;
100  }
101  bool on_found_edge(const edge &e){
102      int u=st[e.u],v=st[e.v];
103      if(S[v]==-1){
104          pa[v]=e.u,S[v]=1; int nu=st[match[v]];
105          slack[v]=slack[nu]=0; S[nu]=0,q_push(nu);
106      }else if(S[v]==0){
107          int lca=get_lca(u,v);
108          if(!lca) return augment(u,v),augment(v,u),true;
109          else add_blossom(u,lca,v);
110      }
111      return false;
112  }
113  bool matching(){
114      memset(S+1,-1,sizeof(int)*n_x);
115      memset(slack+1,0,sizeof(int)*n_x); q=queue<int>();
116      for(int x=1;x<=n_x;++x)
117          if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
118      if(q.empty()) return false;
119      for(;;){ while(q.size()){
120          int u=q.front();q.pop();if(S[st[u]]==1)continue;
121          for(int v=1;v<=n_x;++v)
122              if(g[u][v].w>0&&st[u]!=st[v]){
123                  if(e_delta(g[u][v])==0){
124                      if(on_found_edge(g[u][v])) return true;
125                      else update_slack(u,st[v]);
126                  }
127              }
128          int d=INF;
129          for(int b=n+1;b<=n_x;++b)
130              if(st[b]==b&&S[b]==1) d=min(d,lab[b]/2);
131          for(int x=1;x<=n_x;++x) if(st[x]==x&&slack[x]){
132              if(S[x]==-1) d=min(d,e_delta(g[slack[x]][x]));
133              else if(S[x]==0)
134                  d=min(d,e_delta(g[slack[x]][x])/2);
135          }
136          for(int u=1;u<=n;++u){ if(S[st[u]]==0){
137              if(lab[u]<=d) return 0; lab[u]-=d;
138              }else if(S[st[u]]==1) lab[u]+=d;
139          }
140          for(int b=n+1;b<=n_x;++b) if(st[b]==b){
141              if(S[st[b]]==0) lab[b]+=d*2;
142              else if(S[st[b]]==1) lab[b]-=d*2;
143          }
144          q=queue<int>();
145          for(int x=1;x<=n_x;++x)
146              if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta(
147                  g[slack[x]][x])==0)
148                  if(on_found_edge(g[slack[x]][x])) return true;
149          for(int b=n+1;b<=n_x;++b) if(st[b]==b&&S[b]==1&&
150              lab[b]==0) expand_blossom(b);
151      }
152      return false;
153  }
154  pair<long long,int> solve(){
155      memset(match+1,0,sizeof(int)*n); n_x=n;
156      int n_matches=0,w_max=0; long long tot_weight=0;
157      for(int u=0;u<=n;++u) st[u]=u,flo[u].clear();
158      for(int u=1;u<=n;++u) for(int v=1;v<=n;++v){
159          flo_from[u][v]=(u==v?u:0);
160          w_max=max(w_max,g[u][v].w);
161      }
162      for(int u=1;u<=n;++u) lab[u]=w_max;
163      while(matching()) ++n_matches;
164      for(int u=1;u<=n;++u) if(match[u]&&match[u]<u)
165          tot_weight+=g[u][match[u]].w;

```

```

165     return make_pair(tot_weight,n_matches);
166 }
167 void add_edge(int ui,int vi,int wi)
168 { g[ui][vi].w=g[vi][ui].w=wi; }
169 void init(int _n){
170     n=_n;
171     for(int u=1;u<=n;++u) for(int v=1;v<=n;++v)
172         g[u][v]=edge(u,v,0);
173 }
174 }graph;

```

### 7.3 Max Clique

```

1 // max N = 64
2 typedef unsigned long long ll;
3 struct MaxClique{
4     ll nb[ N ] , n , ans;
5     void init( ll _n ){
6         n = _n;
7         for( int i = 0 ; i < n ; i ++ ) nb[ i ] = 0LLU;
8     }
9     void add_edge( ll _u , ll _v ){
10         nb[ _u ] |= ( 1LLU << _v );
11         nb[ _v ] |= ( 1LLU << _u );
12     }
13     void B( ll r , ll p , ll x , ll cnt , ll res ){
14         if( cnt + res < ans ) return;
15         if( p == 0LLU && x == 0LLU ){
16             if( cnt > ans ) ans = cnt;
17             return;
18         }
19         ll y = p | x; y &= -y;
20         ll q = p & ( ~nb[ int( log2( y ) ) ] );
21         while( q ){
22             ll i = int( log2( q & (-q) ) );
23             B( r | ( 1LLU << i ) , p & nb[ i ] , x & nb[ i ]
24                 , cnt + 1LLU , __builtin_popcountll( p & nb[
25                     i ] ) );
26             q &= ~( 1LLU << i );
27             p &= ~( 1LLU << i );
28             x |= ( 1LLU << i );
29         }
30     }
31     int solve(){
32         ans = 0;
33         ll _set = 0;
34         if( n < 64 ) _set = ( 1LLU << n ) - 1;
35         else{
36             for( ll i = 0 ; i < n ; i ++ ) _set |= ( 1LLU <<
37                 i );
38             B( 0LLU , _set , 0LLU , 0LLU , n );
39             return ans;
40         }
41     }
42 }maxClique;

```

### 7.4 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll> > > g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n+1, LINF);
15     negCycle.assign(n+1, false);
16     rlx.assign(n+1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n+1, false);
19     pa.assign(n+1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;

```

```

23         q.push(s); inq[s] = true;
24     }
25
26     while (!q.empty()) {
27         int u = q.front();
28         q.pop(); inq[u] = false;
29         if (rlx[u] >= n) {
30             negCycle[u] = true;
31         }
32         else for (auto& e : g[u]) {
33             int v = e.first;
34             ll w = e.second;
35             if (dis[v] > dis[u] + w) {
36                 dis[v] = dis[u] + w;
37                 rlx[v] = rlx[u] + 1;
38                 pa[v] = u;
39                 if (!inq[v]) {
40                     q.push(v);
41                     inq[v] = true;
42                 }
43             }
44         }
45
46         // Bellman-Ford
47         queue<int> q;
48         vector<int> pa;
49         void BellmanFord(vector<int>& src) {
50             dis.assign(n+1, LINF);
51             negCycle.assign(n+1, false);
52             pa.assign(n+1, -1);
53
54             for (auto& s : src) dis[s] = 0;
55
56             for (int rlx = 1; rlx <= n; rlx++) {
57                 for (int u = 1; u <= n; u++) {
58                     if (dis[u] == LINF) continue; // Important
59                     !!
60                     for (auto& e : g[u]) {
61                         int v = e.first; ll w = e.second;
62                         if (dis[v] > dis[u] + w) {
63                             dis[v] = dis[u] + w;
64                             pa[v] = u;
65                             if (rlx == n) negCycle[v] = true;
66                         }
67                     }
68                 }
69             }
70
71             // Negative Cycle Detection
72             void NegCycleDetect() {
73                 /* No Neg Cycle: NO
74                 Exist Any Neg Cycle:
75                 YES
76                 v0 v1 v2 ... vk v0 */
77
78                 vector<int> src;
79                 for (int i = 1; i <= n; i++)
80                     src.emplace_back(i);
81
82                 SPFA(src);
83                 // BellmanFord(src);
84
85                 int ptr = -1;
86                 for (int i = 1; i <= n; i++) if (negCycle[i])
87                     { ptr = i; break; }
88
89                 if (ptr == -1) { return cout << "NO" << endl, void
90                     (); }
91
92                 cout << "YES\n";
93                 vector<int> ans;
94                 vector<bool> vis(n+1, false);
95
96                 while (true) {
97                     ans.emplace_back(ptr);
98                     if (vis[ptr]) break;
99                     vis[ptr] = true;
100                     ptr = pa[ptr];
101                 }
102                 reverse(ans.begin(), ans.end());
103
104                 vis.assign(n+1, false);
105                 for (auto& x : ans) {
106                     cout << x << ' ';
107                     if (vis[x]) break;

```



```

103     vis[x] = true;
104 }
105 cout << endl;
106 }
107 // Distance Calculation
108 void calcDis(int s) {
109     vector<int> src;
110     src.emplace_back(s);
111     SPFA(src);
112     // BellmanFord(src);
113
114     while (!q.empty()) q.pop();
115     for (int i = 1; i <= n; i++)
116         if (negCycle[i]) q.push(i);
117
118     while (!q.empty()) {
119         int u = q.front(); q.pop();
120         for (auto& e : g[u]) {
121             int v = e.first;
122             if (!negCycle[v]) {
123                 q.push(v);
124                 negCycle[v] = true;
125             }
126         }
127     }
}

```

## 7.5 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<maxn> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e]^u;
27         if (!dfn[v]) {
28             // tree edge
29             kid++; dfs(v);
30             low[u] = min(low[u], low[v]);
31             if (!rt && low[v] >= dfn[u]) {
32                 // bcc found: u is ap
33                 isap[u] = true;
34                 popout(u);
35             }
36         } else {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         }
40     }
41     // special case: root
42     if (rt) {
43         if (kid > 1) isap[u] = true;
44         popout(u);
45     }
46 }
47 void init() {
48     cin >> n >> m;
49     fill(low, low+maxn, INF);
50     REP(i, m) {
51         int u, v;
52         cin >> u >> v;
53         g[u].emplace_back(i);
54         g[v].emplace_back(i);

```

```

55     E.emplace_back(u^v);
56 }
57 }
58 void solve() {
59     FOR(i, 1, n+1, 1) {
60         if (!dfn[i]) dfs(i, true);
61     }
62     vector<int> ans;
63     int cnt = 0;
64     FOR(i, 1, n+1, 1) {
65         if (isap[i]) cnt++, ans.emplace_back(i);
66     }
67     cout << cnt << endl;
68     Each(i, ans) cout << i << ' ';
69     cout << endl;
70 }

```

## 7.6 BCC - Bridge

```

1 int n, m;
2 vector<int> g[maxn], E;
3 int low[maxn], dfn[maxn], instp;
4 int bccnt, bccid[maxn];
5 stack<int> stk;
6 bitset<maxn> vis, isbrg;
7 void init() {
8     cin >> n >> m;
9     REP(i, m) {
10         int u, v;
11         cin >> u >> v;
12         E.emplace_back(u^v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low+maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e]^u;
36         if (dfn[v]) {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         } else {
40             // tree edge
41             dfs(v);
42             low[u] = min(low[u], low[v]);
43             if (low[v] == dfn[v]) {
44                 isbrg[e] = true;
45                 popout(u);
46             }
47         }
48     }
49 }
50 void solve() {
51     FOR(i, 1, n+1, 1) {
52         if (!dfn[i]) dfs(i);
53     }
54     vector<pii> ans;
55     vis.reset();
56     FOR(u, 1, n+1, 1) {
57         Each(e, g[u]) {
58             if (!isbrg[e] || vis[e]) continue;
59             vis[e] = true;
60             int v = E[e]^u;
61             ans.emplace_back(mp(u, v));
62         }

```

```

63 }
64 cout << (int)ans.size() << endl;
65 Each(e, ans) cout << e.F << ' ' << e.S << endl;
66 }

```

## 7.7 SCC - Tarjan with 2-SAT

```

1 // Author: Ian
2 // 2-sat + tarjan SCC
3 void solve() {
4     int n, r, l; cin >> n >> r >> l;
5     V<P<int>> v(l);
6     for (auto& [a, b] : v)
7         cin >> a >> b;
8     V<V<int>> e(2 * l);
9     for (int i = 0; i < l; i++)
10         for (int j = i + 1; j < l; j++) {
11             if (v[i].first == v[j].first && abs(v[i].second -
12                 v[j].second) <= 2 * r) {
13                 e[i << 1].emplace_back(j << 1 | 1);
14                 e[j << 1].emplace_back(i << 1 | 1);
15             }
16             if (v[i].second == v[j].second && abs(v[i].first -
17                 v[j].first) <= 2 * r) {
18                 e[i << 1 | 1].emplace_back(j << 1);
19                 e[j << 1 | 1].emplace_back(i << 1);
20             }
21         }
22     V<bool> ins(2 * l, false);
23     V<int> scc(2 * l), dfn(2 * l, -1), low(2 * l, inf);
24     stack<int> s;
25     function<void(int)> dfs = [&](int x) {
26         if (~dfn[x]) return;
27         static int t = 0;
28         dfn[x] = low[x] = t++;
29         s.push(x), ins[x] = true;
30         for (auto i : e[x])
31             if (dfs(i), ins[i])
32                 low[x] = min(low[x], low[i]);
33         if (dfn[x] == low[x]) {
34             static int ncnt = 0;
35             int p; do {
36                 ins[p = s.top()] = false;
37                 s.pop(), scc[p] = ncnt;
38             } while (p != x); ncnt++;
39         }
40     };
41     for (int i = 0; i < 2 * l; i++)
42         dfs(i);
43     for (int i = 0; i < l; i++)
44         if (scc[i << 1] == scc[i << 1 | 1]) {
45             cout << "NO" << endl;
46             return;
47         }
48     cout << "YES" << endl;
49 }

```

## 7.8 Eulerian Path - Undir

```

1 // Author: Gino
2 // Usage: build deg, G first, then eulerian()
3 int n, m; // number of vertices and edges
4 vector<int> deg; // degree
5 vector<set<pii>> G; // G[u] := {(v, edge id)}
6
7 vector<int> path_u, path_e;
8 void dfs(int u) {
9     while (!G[u].empty()) {
10         auto it = G[u].begin();
11         auto [v, i] = *it; G[u].erase(it);
12         G[v].erase(make_pair(u, i)); dfs(v);
13         path_u.emplace_back(v);
14         path_e.emplace_back(i);
15     }
16 }
17 void gogo(int s) {
18     path_u.clear(); path_e.clear();
19     dfs(s); path_u.emplace_back(s);
20     reverse(path_u.begin(), path_u.end());
21     reverse(path_e.begin(), path_e.end());

```

```

22 }
23 bool eulerian() {
24     int oddcnt = 0, s = -1;
25     for (int u = 1; u <= n; u++)
26         if (deg[u] & 1)
27             oddcnt++, s = u;
28
29     if (oddcnt != 0 && oddcnt != 2) return false;
30     if (s == -1) {
31         s = 1; for (int u = 1; u <= n; u++)
32             if (deg[u] > 0)
33                 s = u;
34     }
35     gogo(s);
36
37     for (int u = 1; u <= n; u++)
38         if ((int)G[u].size() > 0)
39             return false;
40     return true;
41 }

```

## 7.9 Eulerian Path - Dir

```

1 // Author: Gino
2 // Usage: build ind, oud, G first, then eulerian()
3 int n, m; // number of vertices, edges
4 vector<int> ind, oud; // indegree, outdegree
5 vector<vector<pii>> G; // G[u] := {(v, edge id)}
6
7 vector<int> path_u, path_e;
8 void dfs(int u) {
9     while (!G[u].empty()) {
10         auto [v, i] = G[u].back(); G[u].pop_back();
11         dfs(v);
12         path_u.emplace_back(v);
13         path_e.emplace_back(i);
14     }
15 }
16 void gogo(int s) {
17     path_u.clear(); path_e.clear();
18     dfs(s); path_u.emplace_back(s);
19     reverse(path_u.begin(), path_u.end());
20     reverse(path_e.begin(), path_e.end());
21 }
22 bool eulerian() {
23     int s = -1;
24     for (int u = 1; u <= n; u++) {
25         if (abs(oud[u] - ind[u]) > 1) return false;
26         if (oud[u] - ind[u] == 1) {
27             if (s != -1) return false;
28             s = u;
29         }
30     }
31     if (s == -1) {
32         s = 1; for (int u = 1; u <= n; u++)
33             if (ind[u] > 0)
34                 s = u;
35     }
36     gogo(s);
37     for (int u = 1; u <= n; u++)
38         if ((int)G[u].size() > 0)
39             return false;
40
41     return true;
42 }

```

## 7.10 Kth Shortest Path

```

1 // time: O(|E| \lg |E| + |V| \lg |V| + K)
2 // memory: O(|E| \lg |E| + |V|)
3 struct KSP { // 1-base
4     struct nd {
5         int u, v; ll d;
6         nd(int ui=0, int vi=0, ll di=INF) { u=ui; v=vi; d=di; }
7     };
8     struct heap { nd* edge; int dep; heap* chd[4]; };
9     static int cmp(heap* a, heap* b)
10     { return a->edge->d > b->edge->d; }
11     struct node {

```

```

12     int v; ll d; heap* H; nd* E;
13     node(){}
14     node(ll _d, int _v, nd* _E){ d = _d; v = _v; E = _E; }
15     node(heap* _H, ll _d){ H = _H; d = _d; }
16     friend bool operator<(node a, node b)
17     { return a.d > b.d; }
18 };
19 int n, k, s, t, dst[N]; nd *nxt[N];
20 vector<nd*> g[N], rg[N]; heap *nullNd, *head[N];
21 void init(int _n, int _k, int _s, int _t){
22     n = _n; k = _k; s = _s; t = _t;
23     for(int i = 1; i <= n; i++){
24         g[i].clear(); rg[i].clear();
25         nxt[i] = NULL; head[i] = NULL; dst[i] = -1;
26     }
27 }
28 void addEdge(int ui, int vi, ll di){
29     nd* e = new nd(ui, vi, di);
30     g[ui].push_back(e); rg[vi].push_back(e);
31 }
32 queue<int> dfsQ;
33 void dijkstra(){
34     while(dfsQ.size()) dfsQ.pop();
35     priority_queue<node> Q; Q.push(node(0, t, NULL));
36     while (!Q.empty()){
37         node p = Q.top(); Q.pop(); if(dst[p.v] != -1) continue;
38         dst[p.v] = p.d; nxt[p.v] = p.E; dfsQ.push(p.v);
39         for(auto e: rg[p.v]) Q.push(node(p.d + e->d, e->u, e));
40     }
41 }
42 heap* merge(heap* curNd, heap* newNd){
43     if(curNd == nullNd) return newNd;
44     heap* root = new heap; memcpy(root, curNd, sizeof(heap));
45     if(newNd->edge->d < curNd->edge->d){
46         root->edge = newNd->edge;
47         root->chd[2] = newNd->chd[2];
48         root->chd[3] = newNd->chd[3];
49         newNd->edge = curNd->edge;
50         newNd->chd[2] = curNd->chd[2];
51         newNd->chd[3] = curNd->chd[3];
52     }
53     if(root->chd[0]->dep < root->chd[1]->dep)
54         root->chd[0] = merge(root->chd[0], newNd);
55     else root->chd[1] = merge(root->chd[1], newNd);
56     root->dep = max(root->chd[0]->dep,
57                     root->chd[1]->dep) + 1;
58     return root;
59 }
60 vector<heap*> V;
61 void build(){
62     nullNd = new heap; nullNd->dep = 0; nullNd->edge = new nd;
63     fill(nullNd->chd, nullNd->chd + 4, nullNd);
64     while(not dfsQ.empty()){
65         int u = dfsQ.front(); dfsQ.pop();
66         if(!nxt[u]) head[u] = nullNd;
67         else head[u] = head[nxt[u]->v];
68         V.clear();
69         for(auto&& e: g[u]){
70             int v = e->v;
71             if(dst[v] == -1) continue;
72             e->d += dst[v] - dst[u];
73             if(nxt[u] != e){
74                 heap* p = new heap; fill(p->chd, p->chd + 4, nullNd);
75                 p->dep = 1; p->edge = e; V.push_back(p);
76             }
77         }
78         if(V.empty()) continue;
79         make_heap(V.begin(), V.end(), cmp);
80 #define L(X) ((X << 1) + 1)
81 #define R(X) ((X << 1) + 2)
82         for(size_t i = 0; i < V.size(); i++){
83             if(L(i) < V.size()) V[i]->chd[2] = V[L(i)];
84             else V[i]->chd[2] = nullNd;
85             if(R(i) < V.size()) V[i]->chd[3] = V[R(i)];
86             else V[i]->chd[3] = nullNd;
87         }
88         head[u] = merge(head[u], V.front());

```

```

89     }
90 }
91 vector<ll> ans;
92 void first_K(){
93     ans.clear(); priority_queue<node> Q;
94     if(dst[s] == -1) return;
95     ans.push_back(dst[s]);
96     if(head[s] != nullNd)
97         Q.push(node(head[s], dst[s] + head[s]->edge->d));
98     for(int _ = 1; _ < k and not Q.empty(); _++){
99         node p = Q.top(); Q.pop(); ans.push_back(p.d);
100         if(head[p.H->edge->v] != nullNd){
101             q.H = head[p.H->edge->v]; q.d = p.d + q.H->edge->d;
102             Q.push(q);
103         }
104         for(int i = 0; i < 4; i++){
105             if(p.H->chd[i] != nullNd){
106                 q.H = p.H->chd[i];
107                 q.d = p.d - p.H->edge->d + p.H->chd[i]->edge->d;
108                 Q.push(q);
109             }
110 } } }
111 void solve(){ // ans[i] stores the i-th shortest path
112     dijkstra(); build();
113     first_K(); // ans.size() might less than k
114 } solver;

```

## 7.11 System of Difference Constraints

```

1 vector<vector<pair<int, ll>>> G;
2 void add(int u, int v, ll w) {
3     G[u].emplace_back(make_pair(v, w));
4 }

```

- $x_u - x_v \leq c \Rightarrow \text{add}(v, u, c)$
- $x_u - x_v \geq c \Rightarrow \text{add}(u, v, -c)$
- $x_u - x_v = c \Rightarrow \text{add}(v, u, c), \text{add}(u, v, -c)$
- $x_u \geq c \Rightarrow \text{add super vertex } x_0 = 0, \text{ then } x_u - x_0 \geq c \Rightarrow \text{add}(u, 0, -c)$
- Don't forget non-negative constraints for every variable if specified implicitly.
- Interval sum  $\Rightarrow$  Use prefix sum to transform into differential constraints. Don't forget  $S_{i+1} - S_i \geq 0$  if  $x_i$  needs to be non-negative.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

## 8 String

### 8.1 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) {return c - 'a' + 1;}
3 struct RollingHash {
4     string s; int n; ll mod;
5     vector<ll> Cexp, hs;
6     RollingHash(string& _s, ll _mod):
7         s(_s), n((int)_s.size()), mod(_mod)
8     {
9         Cexp.assign(n, 0);
10        hs.assign(n, 0);
11        Cexp[0] = 1;
12        for (int i = 1; i < n; i++) {
13            Cexp[i] = Cexp[i-1] * C;
14            if (Cexp[i] >= mod) Cexp[i] %= mod;
15        }
16        hs[0] = id(s[0]);
17        for (int i = 1; i < n; i++) {
18            hs[i] = hs[i-1] * C + id(s[i]);
19            if (hs[i] >= mod) hs[i] %= mod;
20        }
21        inline ll query(int l, int r) {

```

```

22     ll res = hs[r] - (l ? hs[l-1] * Cexp[r-l+1] :
23         0);
24     res = (res % mod + mod) % mod;
25     return res; }
};

```

## 8.2 Trie

```

1 struct node {
2     int c[26]; ll cnt;
3     node(): cnt(0) {memset(c, 0, sizeof(c));}
4     node(ll x): cnt(x) {memset(c, 0, sizeof(c));}
5 };
6 struct Trie {
7     vector<node> t;
8     void init() {
9         t.clear();
10        t.emplace_back(node());
11    }
12    void insert(string s) { int ptr = 0;
13        for (auto& i : s) {
14            if (!t[ptr].c[i-'a']) {
15                t.emplace_back(node());
16                t[ptr].c[i-'a'] = (int)t.size()-1; }
17            ptr = t[ptr].c[i-'a']; }
18        t[ptr].cnt++; }
19 } trie;

```

## 8.3 KMP

```

1 int n, m;
2 string s, p;
3 vector<int> f;
4 void build() {
5     f.clear(); f.resize(m, 0);
6     int ptr = 0; for (int i = 1; i < m; i++) {
7         while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
8         if (p[i] == p[ptr]) ptr++;
9         f[i] = ptr;
10    }
11    void init() {
12        cin >> s >> p;
13        n = (int)s.size();
14        m = (int)p.size();
15        build(); }
16    void solve() {
17        int ans = 0, pi = 0;
18        for (int si = 0; si < n; si++) {
19            while (pi && s[si] != p[pi]) pi = f[pi-1];
20            if (s[si] == p[pi]) pi++;
21            if (pi == m) ans++, pi = f[pi-1];
22        }
23        cout << ans << endl; }

```

## 8.4 Z Value

```

1 string is, it, s;
2 int n; vector<int> z;
3 void init() {
4     cin >> is >> it;
5     s = it+'0'+is;
6     n = (int)s.size();
7     z.resize(n, 0); }
8 void solve() {
9     int ans = 0; z[0] = n;
10    for (int i = 1, l = 0, r = 0; i < n; i++) {
11        if (i <= r) z[i] = min(z[i-l], r-i+1);
12        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
13        if (i+z[i]-1 > r) l = i, r = i+z[i]-1;
14        if (z[i] == (int)it.size()) ans++;
15    }
16    cout << ans << endl; }

```

## 8.5 Manacher

```

1 int n; string S, s;
2 vector<int> m;

```

```

3 void manacher() {
4     s.clear(); s.resize(2*n+1, '.');
5     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
6     m.clear(); m.resize(2*n+1, 0);
7     // m[i] := max k such that s[i-k, i+k] is palindrome
8     int mx = 0, mxk = 0;
9     for (int i = 1; i < 2*n+1; i++) {
10        if (mx-(i-mx) >= 0) m[i] = min(m[mx-(i-mx)], mx+mxk-i);
11        while (0 <= i-m[i]-1 && i+m[i]+1 < 2*n+1 &&
12            s[i-m[i]-1] == s[i+m[i]+1]) m[i]++;
13        if (i+m[i] > mx+mxk) mx = i, mxk = m[i];
14    } }
15 void init() { cin >> S; n = (int)S.size(); }
16 void solve() {
17     manacher();
18     int mx = 0, ptr = 0;
19     for (int i = 0; i < 2*n+1; i++) if (mx < m[i])
20         { mx = m[i]; ptr = i; }
21     for (int i = ptr-mx; i <= ptr+mx; i++)
22         if (s[i] != '.') cout << s[i];
23     cout << endl; }

```

## 8.6 Suffix Array

```

1 #define F first
2 #define S second
3 struct SuffixArray { // don't forget s += "$";
4     int n; string s;
5     vector<int> suf, lcp, rk;
6     vector<int> cnt, pos;
7     vector<pair<pii, int>> buc[2];
8     void init(string _s) {
9         s = _s; n = (int)s.size();
10        // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
11    }
12    void radix_sort() {
13        for (int t : {0, 1}) {
14            fill(cnt.begin(), cnt.end(), 0);
15            for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F.S) ]++;
16            for (int i = 0; i < n; i++)
17                pos[i] = (!i ? 0 : pos[i-1] + cnt[i-1]);
18            for (auto& i : buc[t])
19                buc[t^1][pos[ (t ? i.F.F : i.F.S) ]++] = i;
20        }
21        bool fill_suf() {
22            bool end = true;
23            for (int i = 0; i < n; i++) suf[i] = buc[0][i].S;
24            rk[suf[0]] = 0;
25            for (int i = 1; i < n; i++) {
26                int dif = (buc[0][i].F != buc[0][i-1].F);
27                end &= dif;
28                rk[suf[i]] = rk[suf[i-1]] + dif;
29            } return end;
30        }
31        void sa() {
32            for (int i = 0; i < n; i++)
33                buc[0][i] = make_pair(make_pair(s[i], s[i]), i);
34            sort(buc[0].begin(), buc[0].end());
35            if (fill_suf()) return;
36            for (int k = 0; (1<<k) < n; k++) {
37                for (int i = 0; i < n; i++)
38                    buc[0][i] = make_pair(make_pair(rk[i], rk[(i + (1<<k)) % n]), i);
39                radix_sort();
40                if (fill_suf()) return;
41            }
42            void LCP() { int k = 0;
43                for (int i = 0; i < n-1; i++) {
44                    if (rk[i] == 0) continue;
45                    int pi = rk[i];
46                    int j = suf[pi-1];
47                    while (i+k < n && j+k < n && s[i+k] == s[j+k]) k++;
48                    lcp[pi] = k;
49                    k = max(k-1, 0);

```

```

50     }}
51 };
52 SuffixArray suffixarray;

```

## 8.7 Suffix Automaton

```

1 // any path start from root forms a substring of S
2 // occurrence of P: iff SAM can run on input word P
3 // number of different substring: ds[1]-1
4 // total length of all different substring: dsl[1]
5 // max/min length of state i: mx[i]/mx[mom[i]]+1
6 // assume a run on input word P end at state i:
7 // number of occurrences of P: cnt[i]
8 // first occurrence position of P: fp[i]-|P|+1
9 // all position: !clone nodes in dfs from i through
10 // rmom
11 const int MXM=1000010;
12 struct SAM{
13     int tot,root,lst,mom[MXM],mx[MXM]; // ind[MXM]
14     int nxt[MXM][33]; // cnt[MXM],ds[MXM],dsl[MXM],fp[MXM]
15     // bool v[MXM],clone[MXN]
16     int newNode(){
17         int res=++tot; fill(nxt[res],nxt[res]+33,0);
18         mom[res]=mx[res]=0; // cnt=ds=dsl=fp=v=clone=0
19         return res;
20     }
21     void init(){ tot=0;root=newNode();lst=root; }
22     void push(int c){
23         int p=lst,np=newNode(); // cnt[np]=1,clone[np]=0
24         mx[np]=mx[p]+1; // fp[np]=mx[np]-1
25         for(;p&&nxt[p][c]==0;p=mom[p]) nxt[p][c]=np;
26         if(p==0) mom[np]=root;
27         else{
28             int q=nxt[p][c];
29             if(mx[p]+1==mx[q]) mom[np]=q;
30             else{
31                 int nq=newNode(); // fp[nq]=fp[q],clone[nq]=1
32                 mx[nq]=mx[p]+1;
33                 for(int i=0;i<33;i++) nxt[nq][i]=nxt[q][i];
34                 mom[nq]=mom[q]; mom[q]=nq; mom[np]=nq;
35                 for(;p&&nxt[p][c]==q;p=mom[p]) nxt[p][c]=nq;
36             }
37         }
38         lst=np;
39     }
40     void calc(){
41         calc(root); iota(ind,ind+tot,1);
42         sort(ind,ind+tot,[&](int i,int j){return mx[i]<mx[j];});
43         for(int i=tot-1;i>=0;i--)
44             cnt[mom[ind[i]]]+=cnt[ind[i]];
45     }
46     void calc(int x){
47         v[x]=ds[x]=1;dsl[x]=0; // rmom[mom[x]].push_back(x)
48         for(int i=0;i<26;i++){
49             if(nxt[x][i]){
50                 if(!v[nxt[x][i]]) calc(nxt[x][i]);
51                 ds[x]+=ds[nxt[x][i]];
52                 dsl[x]+=ds[nxt[x][i]]+dsl[nxt[x][i]];
53             }
54         }
55     }
56     void push(char *str){
57         for(int i=0;str[i];i++) push(str[i]-'a');
58     }
59 } sam;

```

## 8.8 SA-IS

```

1 const int N=300010;
2 struct SA{
3     #define REP(i,n) for(int i=0;i<int(n);i++)
4     #define REP1(i,a,b) for(int i=(a);i<=int(b);i++)
5     bool _t[N*2]; int _s[N*2],_sa[N*2];
6     int _c[N*2],x[N],_p[N],_q[N*2],hei[N],r[N];
7     int operator [](int i){ return _sa[i]; }
8     void build(int *s,int n,int m){
9         memcpy(_s,s,sizeof(int)*n);
10        sais(_s,_sa,_p,_q,_t,_c,n,m); mkhei(n);
11    }

```

```

12 void mkhei(int n){
13     REP(i,n) r[_sa[i]]=i;
14     hei[0]=0;
15     REP(i,n) if(r[i]) {
16         int ans=i>0?max(hei[r[i-1]]-1,0):0;
17         while(_s[i+ans]==_s[_sa[r[i]-1]+ans]) ans++;
18         hei[r[i]]=ans;
19     }
20 }
21 void sais(int *s,int *sa,int *p,int *q,bool *t,int *c,
22           int n,int z){
23     bool uniq=t[n-1]=true,neq;
24     int nn=0,nmxz=-1,*nsa=sa+n,*ns=s+n,lst=-1;
25     #define MS0(x,n) memset((x),0,n*sizeof(*(x)))
26     #define MAGIC(XD) MS0(sa,n);\
27     memcpy(x,c,sizeof(int)*z); XD;\
28     memcpy(x+1,c,sizeof(int)*(z-1));\
29     REP(i,n) if(sa[i]&&!t[sa[i]-1]) sa[x[s[sa[i]-1]]++] = sa[i]-1;\
30     memcpy(x,c,sizeof(int)*z);\
31     for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[sa[i]-1]]]=sa[i]-1;
32     MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
33     REP(i,z-1) c[i+1]+=c[i];
34     if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
35     for(int i=n-2;i>=0;i--){
36         t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
37         MAGIC(REP1(i,1,n-1) if(t[i]&&t[i-1]) sa[--x[s[i]]]=p[q[i]=nn++]=i);
38         REP(i,n) if(sa[i]&&t[sa[i]]&&t[sa[i]-1]){
39             neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa[i])*sizeof(int));
40             ns[q[lst=sa[i]]]=nmxz+=neq;
41         }
42         sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
43         MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i]]]]]=p[nsa[i]]);
44     }
45     }sa;
46     int H[N],SA[N],RA[N];
47     void suffix_array(int* ip,int len){
48         // should padding a zero in the back
49         // ip is int array, len is array length
50         // ip[0..n-1] != 0, and ip[len]=0
51         ip[len++]=0; sa.build(ip,len,128);
52         memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2);
53         ;
54         for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
55         // resulting height, sa array \in [0,len)
56     }

```

## 8.9 Minimum Rotation

```

1 // Lexicographically minimal rotation
2 // rotate(begin(s), begin(s)+minRotation(s), end(s))
3 int minRotation(string s) {
4     int a = 0, n = s.size(); s += s;
5     for(int b = 0; b < n; b++) for(int k = 0; k < n; k++) {
6         if(a + k == b || s[a + k] < s[b + k]) {
7             b += max(0, k - 1);
8             break; }
9         if(s[a + k] > s[b + k]) {
10            a = b;
11            break;
12        }
13    }
14    return a; }

```

## 8.10 Aho Corasick

```

1 struct ACautomata{
2     struct Node{
3         int cnt;
4         Node *go[26], *fail, *dic;
5         Node(){
6             cnt = 0; fail = 0; dic=0;
7             memset(go,0,sizeof(go));
8         }
9     }pool[1048576],*root;
10    int nMem;
11    Node* new_Node(){

```



```

12 pool[nMem] = Node();
13 return &pool[nMem++];
14 }
15 void init() { nMem = 0; root = new_Node(); }
16 void add(const string &str) { insert(root, str, 0); }
17 void insert(Node *cur, const string &str, int pos){
18     for(int i=pos; i<str.size(); i++){
19         if(!cur->go[str[i]-'a'])
20             cur->go[str[i]-'a'] = new_Node();
21         cur=cur->go[str[i]-'a'];
22     }
23     cur->cnt++;
24 }
25 void make_fail(){
26     queue<Node*> que;
27     que.push(root);
28     while (!que.empty()){
29         Node* fr=que.front(); que.pop();
30         for (int i=0; i<26; i++){
31             if (fr->go[i]){
32                 Node *ptr = fr->fail;
33                 while (ptr && !ptr->go[i]) ptr = ptr->fail;
34                 fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
35                 fr->go[i]->dic=(ptr->cnt?ptr->dic);
36                 que.push(fr->go[i]);
37             } } }
38 }AC;

```

## 9 Geometry

### 9.1 Template

```

1 // Author: Gino
2 typedef long long T;
3 // typedef Long double T;
4 const long double eps = 1e-8;
5
6 short sgn(T x) {
7     if (abs(x) < eps) return 0;
8     return x < 0 ? -1 : 1;
9 }
10
11 struct Pt {
12     T x, y;
13     Pt(T _x=0, T _y=0):x(_x), y(_y) {}
14     Pt operator+(Pt a) { return Pt(x+a.x, y+a.y); }
15     Pt operator-(Pt a) { return Pt(x-a.x, y-a.y); }
16     Pt operator*(T a) { return Pt(x*a, y*a); }
17     Pt operator/(T a) { return Pt(x/a, y/a); }
18     operator*(Pt a) { return x*a.x + y*a.y; }
19     operator^(Pt a) { return x*a.y - y*a.x; } // 不要打反
20     bool operator<(Pt a) {
21         { return x < a.x || (x == a.x && y < a.y); }
22         //return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.y) < 0); }
23     bool operator==(Pt a) {
24         { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
25     };
26
27     Pt mv(Pt a, Pt b) { return b-a; }
28     T len2(Pt a) { return a*a; }
29     T dis2(Pt a, Pt b) { return len2(b-a); }
30
31     short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
32     bool onseg(Pt p, Pt l1, Pt l2) {
33         Pt a = mv(p, l1), b = mv(p, l2);
34         return ((a^b) == 0) && ((a*b) <= 0);
35     }

```

### 9.2 Basic Operations

```

1 // Author: Gino
2 // Function: Check if a point P sits in a polygon (
3 // doesn't have to be convex hull)
4 // 0 = Bound, 1 = In, -1 = Out
5 short inPoly(Pt p) {
6     for (int i = 0; i < n; i++)
7         if (onseg(p, E[i], E[(i+1)%n])) return 0;

```

```

7 int cnt = 0;
8 for (int i = 0; i < n; i++)
9     if (banana(p, Pt(p.x+1, p.y+2e9), E[i], E[(i+1)%n]))
10         cnt ^= 1;
11 return (cnt ? 1 : -1);
12 }

```

```

1 // Author: Gino
2 int ud(Pt a) { // up or down half plane
3     if (a.y > 0) return 0;
4     if (a.y < 0) return 1;
5     return (a.x >= 0 ? 0 : 1);
6 }
7 sort(ALL(E), [&](const Pt& a, const Pt& b){
8     if (ud(a) != ud(b)) return ud(a) < ud(b);
9     return (a^b) > 0;
10 });

```

```

1 // Author: Gino
2 // Function: check if (p1---p2) (q1---q2) banana
3 inline bool banana(Pt p1, Pt p2, Pt q1, Pt q2) {
4     if (onseg(p1, q1, q2) || onseg(p2, q1, q2) ||
5         onseg(q1, p1, p2) || onseg(q2, p1, p2)) {
6         return true;
7     }
8     Pt p = mv(p1, p2), q = mv(q1, q2);
9     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 &&
10         ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
11 }

```

```

1 // Author: Gino
2 // T: Long double
3 Pt bananaPoint(Pt p1, Pt p2, Pt q1, Pt q2) {
4     if (onseg(q1, p1, p2)) return q1;
5     if (onseg(q2, p1, p2)) return q2;
6     if (onseg(p1, q1, q2)) return p1;
7     if (onseg(p2, q1, q2)) return p2;
8     double s = abs(mv(p1, p2) ^ mv(p1, q1));
9     double t = abs(mv(p1, p2) ^ mv(p1, q2));
10    return q2 * (s/(s+t)) + q1 * (t/(s+t));
11 }

```

```

1 // Author: Gino
2 vector<Pt> hull;
3 void convexHull() {
4     hull.clear(); sort(E.begin(), E.end());
5     for (int t : {0, 1}) {
6         int b = (int)hull.size();
7         for (auto& ei : E) {
8             while ((int)hull.size() - b >= 2 &&
9                 ori(mv(hull[(int)hull.size()-2], hull.
10                     back()),
11                     mv(hull[(int)hull.size()-2], ei)) ==
12                     -1) {
13                 hull.pop_back();
14             }
15             hull.emplace_back(ei);
16         }
17         hull.pop_back();
18         reverse(E.begin(), E.end());
19     } }

```

```

1 // Author: Gino
2 // Function: Return doubled area of a polygon
3 T dbarea(vector<Pt>& e) {
4     ll res = 0;
5     for (int i = 0; i < (int)e.size(); i++)
6         res += e[i]^e[(i+1)%SZ(e)];
7     return abs(res);
8 }

```

### 9.3 Lower Concave Hull

```

1 // Author: Unknown
2 struct Line {
3     mutable ll m, b, p;
4     bool operator<(const Line& o) const { return m < o.m; }
5     bool operator<(ll x) const { return p < x; }

```



```

6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10     const ll inf = LLONG_MAX;
11     ll div(ll a, ll b) { // floored division
12         return a / b - ((a ^ b) < 0 && a % b); }
13     bool isect(iterator x, iterator y) {
14         if (y == end()) { x->p = inf; return false; }
15         if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
16         else x->p = div(y->b - x->b, x->m - y->m);
17         return x->p >= y->p;
18     }
19     void add(ll m, ll b) {
20         auto z = insert({m, b, 0}), y = z++, x = y;
21         while (isect(y, z)) z = erase(z);
22         if (x != begin() && isect(--x, y)) isect(x, y =
23             erase(y));
24         while ((y = x) != begin() && (--x)->p >= y->p)
25             isect(x, erase(y));
26     }
27     ll query(ll x) {
28         assert(!empty());
29         auto l = *lower_bound(x);
30         return l.m * x + l.b;
31     }
32 };

```

## 9.4 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let  $i$  = number of points inside the polygon.

Let  $b$  = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

## 9.5 Minimum Enclosing Circle

```

1 // Author: Gino
2 // Function: Find Min Enclosing Circle using Randomized
3 // O(n) Algorithm
4 Pt circumcenter(Pt A, Pt B, Pt C) {
5     // a1(x-A.x) + b1(y-A.y) = c1
6     // a2(x-A.x) + b2(y-A.y) = c2
7     // solve using Cramer's rule
8     T a1 = B.x-A.x, b1 = B.y-A.y, c1 = dis2(A, B)/2.0;
9     T a2 = C.x-A.x, b2 = C.y-A.y, c2 = dis2(A, C)/2.0;
10    T D = Pt(a1, b1) ^ Pt(a2, b2);
11    T Dx = Pt(c1, b1) ^ Pt(c2, b2);
12    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
13    if (D == 0) return Pt(-INF, -INF);
14    return A + Pt(Dx/D, Dy/D);
15 }
16 Pt center; T r2;
17
18 void minEncloseCircle() {
19     mt19937 gen(chrono::steady_clock::now().
20         time_since_epoch().count());
21     shuffle(ALL(E), gen);
22     center = E[0], r2 = 0;
23
24     for (int i = 0; i < n; i++) {
25         if (dis2(center, E[i]) <= r2) continue;
26         center = E[i], r2 = 0;
27         for (int j = 0; j < i; j++) {
28             if (dis2(center, E[j]) <= r2) continue;
29             center = (E[i] + E[j]) / 2.0;
30             r2 = dis2(center, E[i]);
31             for (int k = 0; k < j; k++) {
32                 if (dis2(center, E[k]) <= r2) continue;
33                 center = circumcenter(E[i], E[j], E[k]);
34                 r2 = dis2(center, E[i]);
35             }
36         }
37     }
38 }

```

## 9.6 PolyUnion

```

1 // Author: Unknown
2 struct PY{
3     int n; Pt pt[5]; double area;
4     Pt& operator[](const int x){ return pt[x]; }
5     void init(){ //n,pt[0~n-1] must be filled
6         area=pt[n-1]^pt[0];
7         for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
8         if((area/=2)<0)reverse(pt,pt+n),area=-area;
9     }
10 };
11 PY py[500]; pair<double,int> c[5000];
12 inline double segP(Pt &p,Pt &p1,Pt &p2){
13     if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
14     return (p.x-p1.x)/(p2.x-p1.x);
15 }
16 double polyUnion(int n){ //py[0~n-1] must be filled
17     int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
18     for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
19     for(i=0;i<n;i++){
20         for(ii=0;ii<py[i].n;ii++){
21             r=0;
22             c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
23             for(j=0;j<n;j++){
24                 if(i==j) continue;
25                 for(jj=0;jj<py[j].n;jj++){
26                     ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]))
27                         ;
28                     tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj
29                         +1]));
30                     if(ta==0 && tb==0){
31                         if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
32                             i][ii])>0&&j<i){
33                             c[r++]=make_pair(segP(py[j][jj],py[i][ii
34                                 ],py[i][ii+1]),1);
35                             c[r++]=make_pair(segP(py[j][jj+1],py[i][
36                                 ii],py[i][ii+1]),-1);
37                         }
38                     }else if(ta>0 && tb<0){
39                         tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
40                         td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
41                         c[r++]=make_pair(tc/(tc-td),1);
42                     }else if(ta<0 && tb>=0){
43                         tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
44                         td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
45                         c[r++]=make_pair(tc/(tc-td),-1);
46                     }
47                 }
48             }
49             sort(c,c+r);
50             z=min(max(c[0].first,0.0),1.0); d=c[0].second; s
51                 =0;
52             for(j=1;j<r;j++){
53                 w=min(max(c[j].first,0.0),1.0);
54                 if(!d) s+=w-z;
55                 d+=c[j].second; z=w;
56             }
57             sum+=(py[i][ii]^py[i][ii+1])*s;
58         }
59     }
60     return sum/2;
61 }

```

## 9.7 Minkowski Sum

```

1 // Author: Unknown
2 /* convex hull Minkowski Sum*/
3 #define INF 1000000000000000LL
4 int pos(const Pt &tp){
5     if(tp.Y == 0) return tp.X > 0 ? 0 : 1;
6     return tp.Y > 0 ? 0 : 1;
7 }
8 #define N 300030
9 Pt pt[N], qt[N], rt[N];
10 LL Lx,Rx;
11 int dn,un;
12 inline bool cmp(Pt a, Pt b){
13     int pa=pos(a),pb=pos(b);
14     if(pa==pb) return (a^b)>0;
15     return pa<pb;
16 }
17 int minkowskiSum(int n,int m){

```

```

18 int i,j,r,p,q,fi,fj;
19 for(i=1,p=0;i<n;i++){
20     if( pt[i].Y<pt[p].Y ||
21         (pt[i].Y==pt[p].Y && pt[i].X<pt[p].X) ) p=i; }
22 for(i=1,q=0;i<m;i++){
23     if( qt[i].Y<qt[q].Y ||
24         (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i; }
25 rt[0]=pt[p]+qt[q];
26 r=1; i=p; j=q; fi=fj=0;
27 while(1){
28     if((fj&&j==q) ||
29         ( !fi || i==p) &&
30         cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%m]-qt[q]) > 0 ){
31         rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
32         p=(p+1)%n;
33         fi=1;
34     }else{
35         rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
36         q=(q+1)%m;
37         fj=1;
38     }
39     if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
40         r++;
41     else rt[r-1]=rt[r];
42     if(i==p && j==q) break;
43     return r-1;
44 }
45 void initInConvex(int n){
46     int i,p,q;
47     LL Ly,Ry;
48     Lx=INF; Rx=-INF;
49     for(i=0;i<n;i++){
50         if(pt[i].X<Lx) Lx=pt[i].X;
51         if(pt[i].X>Rx) Rx=pt[i].X;
52     }
53     Ly=Ry=INF;
54     for(i=0;i<n;i++){
55         if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
56         if(pt[i].X==Rx && pt[i].Y<Ry){ Ry=pt[i].Y; q=i; }
57     }
58     for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
59     qt[dn]=pt[q]; Ly=Ry=-INF;
60     for(i=0;i<n;i++){
61         if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
62         if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
63     }
64     for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
65     rt[un]=pt[q];
66 }
67 inline int inConvex(Pt p){
68     int L,R,M;
69     if(p.X<Lx || p.X>Rx) return 0;
70     L=0;R=dn;
71     while(L<R-1){ M=(L+R)/2;
72         if(p.X<qt[M].X) R=M; else L=M; }
73     if(tri(qt[L],qt[R],p)<0) return 0;
74     L=0;R=un;
75     while(L<R-1){ M=(L+R)/2;
76         if(p.X<rt[M].X) R=M; else L=M; }
77     if(tri(rt[L],rt[R],p)>0) return 0;
78     return 1;
79 }
80 int main(){
81     int n,m,i;
82     Pt p;
83     scanf("%d",&n);
84     for(i=0;i<n;i++) scanf("%lld%lld",&pt[i].X,&pt[i].Y);
85     scanf("%d",&m);
86     for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
87     n=minkowskiSum(n,m);
88     for(i=0;i<n;i++) pt[i]=rt[i];
89     scanf("%d",&m);
90     for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
91     n=minkowskiSum(n,m);
92     for(i=0;i<n;i++) pt[i]=rt[i];
93     initInConvex(n);
94     scanf("%d",&m);
95     for(i=0;i<m;i++){
96         scanf("%lld %lld",&p.X,&p.Y);
97         p.X*=3; p.Y*=3;
98         puts(inConvex(p)? "YES": "NO");

```

```

99 }
100 }

```

## 10 Number Theory

### 10.1 Prime Sieve and Defactor

```

1 // Author: Gino
2 const int maxc = 1e6 + 1;
3 vector<int> lpf;
4 vector<int> prime;
5
6 void seive() {
7     prime.clear();
8     lpf.resize(maxc, 1);
9     for (int i = 2; i < maxc; i++) {
10         if (lpf[i] == 1) {
11             lpf[i] = i;
12             prime.emplace_back(i);
13         }
14         for (auto& j : prime) {
15             if (i * j >= maxc) break;
16             lpf[i * j] = j;
17             if (j == lpf[i]) break;
18         }
19     }
20     vector<pii> fac;
21     void defactor(int u) {
22         fac.clear();
23         while (u > 1) {
24             int d = lpf[u];
25             fac.emplace_back(make_pair(d, 0));
26             while (u % d == 0) {
27                 u /= d;
28                 fac.back().second++;
29             }
30         }
31     }

```

### 10.2 Harmonic Series

```

1 // Author: Gino
2 // O(n log n)
3 for (int i = 1; i <= n; i++) {
4     for (int j = i; j <= n; j += i) {
5         // O(1) code
6     }
7 }
8
9 // PIE
10 // given array a[0], a[1], ..., a[n - 1]
11 // calculate dp[x] = number of pairs (a[i], a[j]) such
12 // that gcd(a[i], a[j]) = x // (i < j)
13 //
14 // idea: Let mc(x) = # of y s.t. x|y
15 // f(x) = # of pairs s.t. gcd(a[i], a[j]) >=
16 // x f(x) = C(mc(x), 2)
17 // dp[x] = f(x) - sum(dp[y], x < y and x|y)
18 const int maxc = 1e6;
19 vector<int> cnt(maxc + 1, 0), dp(maxc + 1, 0);
20 for (int i = 0; i < n; i++)
21     cnt[a[i]]++;
22
23 for (int x = maxc; x >= 1; x--) {
24     ll cnt_mul = 0; // number of multiples of x
25     for (int y = x; y <= maxc; y += x)
26         cnt_mul += cnt[y];
27
28     dp[x] = cnt_mul * (cnt_mul - 1) / 2; // number of
29     // pairs that are divisible by x
30     for (int y = x + x; y <= maxc; y += x)
31         dp[x] -= dp[y]; // PIE: subtract all dp[y] for
32         // y > x and x|y
33 }

```

### 10.3 Count Number of Divisors

```

1 // Author: Gino

```

```

2 // Function: Count the number of divisors for all x <= 10^6 using harmonic series
3 const int maxc = 1e6;
4 vector<int> facs;
5
6 void find_all_divisors() {
7     facs.clear(); facs.resize(maxc + 1, 0);
8     for (int x = 1; x <= maxc; x++) {
9         for (int y = x; y <= maxc; y += x) {
10             facs[y]++;
11         }
12     }
13 }

```

## 10.4 數論分塊

```

1 // Author: Gino
2 /*
3 n = 17
4 i: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
5 n/i: 17 8 5 4 3 2 2 2 1 1 1 1 1 1 1 1 1
6
7           ^       ^
8           L(2)   R(2)
9
10 L(x) := Left bound for n/i = x
11 R(x) := right bound for n/i = x
12
13 ===== FORMULA =====
14 >>> R = n / (n/L) <<<
15 =====
16
17 Example: L(2) = 6
18           R(2) = 17 / (17 / 6)
19               = 17 / 2
20               = 8
21 */
22 // ===== CODE =====
23 for (ll l = 1, r = 1, q = n; l <= n; l = r + 1) {
24     q = n/l;
25     r = n/q;
26     // Process your code here
27 }
28 // q, l, r: 17 1 1
29 // q, l, r: 8 2 2
30 // q, l, r: 5 3 3
31 // q, l, r: 4 4 4
32 // q, l, r: 3 5 5
33 // q, l, r: 2 6 8
34 // q, l, r: 1 9 17

```

## 10.5 Pollard's rho

```

1 // Author: Unknown
2 // Function: Find a non-trivial factor of a big number
3 // in O(n^(1/4) log^2(n))
4
5 ll find_factor(ll number) {
6     __int128 x = 2;
7     for (__int128 cycle = 1; ; cycle++) {
8         __int128 y = x;
9         for (int i = 0; i < (1<<cycle); i++) {
10             x = (x * x + 1) % number;
11             __int128 factor = __gcd(x - y, number);
12             if (factor > 1)
13                 return factor;
14         }
15     }
16 }

```

```

1 # Author: Unknown
2 # Function: Find a non-trivial factor of a big number
3 # in O(n^(1/4) log^2(n))
4 from itertools import count
5 from math import gcd
6 from sys import stdin
7
8 for s in stdin:
9     number, x = int(s), 2
10     brk = False

```

```

10 for cycle in count(1):
11     y = x
12     if brk:
13         break
14     for i in range(1 << cycle):
15         x = (x * x + 1) % number
16         factor = gcd(x - y, number)
17         if factor > 1:
18             print(factor)
19             brk = True
20             break

```

## 10.6 Miller Rabin

```

1 // Author: Unknown, Modified by Gino
2 // Function: Check if a number is a prime in O(100 * log^2(n))
3 // miller_rabin(): return 1 if prime, 0 otherwise
4 inline ll mul(ll x, ll y, ll mod) {
5     return (__int128)(x) * y % mod;
6 }
7 ll mypow(ll a, ll b, ll mod) {
8     ll r = 1;
9     while (b > 0) {
10         if (b & 1) r = mul(r, a, mod);
11         a = mul(a, a, mod);
12         b >>= 1;
13     }
14     return r;
15 }
16 bool witness(ll a, ll n, ll u, int t){
17     ll x = mypow(a, u, n);
18     for(int i = 0; i < t; i++) {
19         ll nx = mul(x, x, n);
20         if (nx == 1 && x != 1 && x != n-1) return true;
21         x = nx;
22     }
23     return x != 1;
24 }
25 bool miller_rabin(ll n) {
26     // if n >= 3,474,749,660,383
27     // change {2, 3, 5, 7, 11, 13} to
28     // {2, 325, 9375, 28178, 450775, 9780504, 1795265022}
29     if (n < 2) return false;
30     if(!(n & 1)) return n == 2;
31     ll u = n - 1; int t = 0;
32     while(!(u & 1)) u >>= 1, t++;
33     for (ll a : {2, 3, 5, 7, 11, 13}) {
34         if (a % n == 0) continue;
35         if (witness(a, n, u, t)) return false;
36     }
37     return true;
38 }
39 // bases that make sure no pseudoprimes flee from test
40 // if WA, replace randll(n - 1) with these bases:
41 // n < 4,759,123,141          3 : 2, 7, 61
42 // n < 1,122,004,669,633     4 : 2, 13, 23, 1662803
43 // n < 3,474,749,660,383     6 : pimes <= 13
44 // n < 2^64                  7 :
45 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022

```

## 10.7 Discrete Log

```

1 // Author: ckiseki
2 // find x^? = y (mod M)
3 // O(sqrt(M))
4 template<typename Int>
5 Int BSGS(Int x, Int y, Int M) {
6     // x^? \equiv y (mod M)
7     Int t = 1, c = 0, g = 1;
8     for (Int M_ = M; M_ > 0; M_ >>= 1) g = g * x % M;
9     for (g = gcd(g, M); t % g != 0; ++c) {
10         if (t == y) return c;
11         t = t * x % M;
12     }
13     if (y % g != 0) return -1;
14     t /= g, y /= g, M /= g;
15     Int h = 0, gs = 1;
16     for (; h * h < M; ++h) gs = gs * x % M;

```

```

17 unordered_map<Int, Int> bs;
18 for (Int s = 0; s < h; bs[y] = ++s) y = y * x % M;
19 for (Int s = 0; s < M; s += h) {
20     t = t * gs % M;
21     if (bs.count(t)) return c + s + h - bs[t];
22 }
23 return -1;
24 }

```

## 10.8 Discrete Sqrt

```

1 // Author: ckiseki
2 // find solution for  $x^2 = n \pmod{P}$ 
3 //  $O(\log^2 P)$ 
4 int get_root(int n, int P) { // ensure  $0 \leq n < p$ 
5     if (P == 2 or n == 0) return n;
6     auto check = [&](lld x) {
7         return modpow(int(x), (P - 1) / 2, P);
8     };
9     if (check(n) != 1) return -1;
10    mt19937 rnd(7122); lld z = 1, w;
11    while (check(w = (z * z - n + P) % P) != P - 1)
12        z = rnd() % P;
13    const auto M = [P, w](auto &u, auto &v) {
14        auto [a, b] = u; auto [c, d] = v;
15        return make_pair((a * c + b * d % P * w) % P,
16                          (a * d + b * c) % P);
17    };
18    pair<lld, lld> r(1, 0), e(z, 1);
19    for (int q = (P + 1) / 2; q; q >>= 1, e = M(e, e))
20        if (q & 1) r = M(r, e);
21    return int(r.first); // sqrt(n) mod P where P is prime

```

## 10.9 Fast Power

Note:  $a^n \equiv a^{(n \bmod (p-1))} \pmod{p}$

## 10.10 Extend GCD

```

1 // Author: Gino
2 // [Usage]
3 // bezout(a, b, c):
4 //     find solution to  $ax + by = c$ 
5 //     return {-LINF, -LINF} if no solution
6 // inv(a, p):
7 //     find modulo inverse of a under p
8 //     return -1 if not exist
9 // CRT(vector<LL>& a, vector<LL>& m)
10 //     find a solution pair (x, mod) satisfies all  $x \equiv a[i] \pmod{m[i]}$ 
11 //     return {-LINF, -LINF} if no solution
12
13 const ll LINF = 4e18;
14 typedef pair<ll, ll> pll;
15 template<typename T1, typename T2>
16 T1 chmod(T1 a, T2 m) {
17     return (a % m + m) % m;
18 }
19
20 ll gcd;
21 pll extgcd(ll a, ll b) {
22     if (b == 0) {
23         gcd = a;
24         return pll{1, 0};
25     }
26     pll ans = extgcd(b, a % b);
27     return pll{ans.second, ans.first - a/b * ans.second};
28 }
29 pll bezout(ll a, ll b, ll c) {
30     bool negx = (a < 0), negy = (b < 0);
31     pll ans = extgcd(abs(a), abs(b));
32     if (c % gcd != 0) return pll{-LINF, -LINF};
33     return pll{ans.first * c/gcd * (negx ? -1 : 1),
34               ans.second * c/gcd * (negy ? -1 : 1)};
35 }
36 ll inv(ll a, ll p) {
37     if (p == 1) return -1;
38     pll ans = bezout(a % p, -p, 1);

```

```

39     if (ans == pll{-LINF, -LINF}) return -1;
40     return chmod(ans.first, p);
41 }
42 pll CRT(vector<ll>& a, vector<ll>& m) {
43     for (int i = 0; i < (int)a.size(); i++)
44         a[i] = chmod(a[i], m[i]);
45
46     ll x = a[0], mod = m[0];
47     for (int i = 1; i < (int)a.size(); i++) {
48         pll sol = bezout(mod, m[i], a[i] - x);
49         if (sol.first == -LINF) return pll{-LINF, -LINF};
50
51         // prevent long long overflow
52         ll p = chmod(sol.first, m[i] / gcd);
53         ll lcm = mod / gcd * m[i];
54         x = chmod((__int128)p * mod + x, lcm);
55         mod = lcm;
56     }
57     return pll{x, mod};
58 }

```

## 10.11 Mu + Phi

```

1 // Author: Gino
2 const int maxn = 1e6 + 5;
3 ll f[maxn];
4 vector<int> lpf, prime;
5 void build() {
6     lpf.clear(); lpf.resize(maxn, 1);
7     prime.clear();
8     f[1] = ...; /* mu[1] = 1, phi[1] = 1 */
9     for (int i = 2; i < maxn; i++) {
10         if (lpf[i] == 1) {
11             lpf[i] = i; prime.emplace_back(i);
12             f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
13         }
14         for (auto& j : prime) {
15             if (i*j >= maxn) break;
16             lpf[i*j] = j;
17             if (i % j == 0) f[i*j] = ...; /* 0, phi[i]*j */
18             else f[i*j] = ...; /* -mu[i], phi[i]*phi[j] */
19             if (j >= lpf[i]) break;
20         }
21     }
22 }

```

## 10.12 Other Formulas

- Pisano Period: 任何線性遞迴（比如費氏數列）模任何一個數字  $M$  都會循環，找循環節  $\pi(M)$  先質因數分解  $M = \prod p_i^{e_i}$ ，然後  $\pi(M) = \text{lcm}(\pi(p_i^{e_i}))$ ，
- Inversion:  $aa^{-1} \equiv 1 \pmod{m}$ .  $a^{-1}$  exists iff  $\gcd(a, m) = 1$ .
- Linear inversion:  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:  $a^p \equiv a \pmod{p}$  if  $p$  is prime.
- Euler function:  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:  $a^{\phi(n)} \equiv 1 \pmod{n}$  if  $\gcd(a, n) = 1$ . If  $a, n$  are not coprime: 質因數分解  $n = \prod p_i^{e_i}$ ，對每個  $p_i^{e_i}$  分開看他們跟  $a$  是否互質（互質：Fermat / 不互質：夠大的指數會直接削成 0），最後用 CRT 合併。
- Extended Euclidean algorithm:  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:  $\sigma_x(n) = \sum_{d|n} d^x$ .  $n = \prod_{i=1}^r p_i^{a_i}$ .  
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$  if  $x \neq 0$ .  $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$ .

- Chinese remainder theorem (Coprime Moduli):

$$x \equiv a_i \pmod{m_i}.$$

$$M = \prod m_i. M_i = M/m_i. t_i = M_i^{-1}.$$

$$x = kM + \sum a_i t_i M_i, k \in \mathbb{Z}.$$

- Chinese remainder theorem:

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 \equiv m_2 q + a_2$$

$$m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$$

Solve for  $(p, q)$  using ExtGCD.

$$x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$$

- Avoiding Overflow:  $ca \bmod cb = c(a \bmod b)$

- Dirichlet Convolution:  $(f * g)(n) = \sum_{d|n} f(d)g(n/d)$

- Important Multiplicative Functions + Properties:

$$1. \epsilon(n) = [n = 1]$$

$$2. 1(n) = 1$$

$$3. id(n) = n$$

$$4. \mu(n) = 0 \text{ if } n \text{ has squared prime factor}$$

$$5. \mu(n) = (-1)^k \text{ if } n = p_1 p_2 \cdots p_k$$

$$6. \epsilon = \mu * 1$$

$$7. \phi = \mu * id$$

$$8. [n = 1] = \sum_{d|n} \mu(d)$$

$$9. [gcd = 1] = \sum_{d|gcd} \mu(d)$$

- Möbius inversion:  $f = g * 1 \Leftrightarrow g = f * \mu$

## 10.13 Polynomial

```

1 // Author: Gino
2 // Preparation: first set_mod(mod, g), then init_ntt()
3 // everytime you change the mod, you have to call
  init_ntt() again
4 //
5 // [Usage]
6 // polynomial: vector<ll> a, b
7 // negation: -a
8 // add/subtract: a += b, a -= b
9 // convolution: a *= b
10 // in-place modulo: mod(a, b)
11 // in-place inversion under mod x^N: inv(id, N)
12
13
14 const int maxk = 20;
15 const int maxn = 1<<maxk;
16
17 using u64 = unsigned long long;
18 using u128 = __uint128_t;
19
20 int g;
21 u64 MOD;
22 u64 BARRETT_IM; // 2^64 / MOD
23
24 inline void set_mod(u64 m, int _g) {
25     g = _g;
26     MOD = m;
27     BARRETT_IM = (u128(1) << 64) / m;
28 }
29 inline u64 chmod(u128 x) {
30     u64 q = (u64)((x * BARRETT_IM) >> 64);
31     u64 r = (u64)(x - (u128)q * MOD);
32     if (r >= MOD) r -= MOD;
33     return r;
34 }
35 inline u64 mmul(u64 a, u64 b) {
36     return chmod((u128)a * b);
37 }
38 ll pw(ll a, ll n) {
39     ll ret = 1;
40     while (n > 0) {
41         if (n & 1) ret = mmul(ret, a);
42         a = mmul(a, a);
43         n >>= 1;
44     }
45     return ret;

```

```

46 }
47
48 vector<ll> X, iX;
49 vector<int> rev;
50 void init_ntt() {
51     X.assign(maxn, 1); // x1 = g^((p-1)/n)
52     iX.assign(maxn, 1);
53
54     ll u = pw(g, (MOD-1)/maxn);
55     ll iu = pw(u, MOD-2);
56     for (int i = 1; i < maxn; i++) {
57         X[i] = mmul(X[i-1], u);
58         iX[i] = mmul(iX[i-1], iu);
59     }
60
61     if ((int)rev.size() == maxn) return;
62     rev.assign(maxn, 0);
63     for (int i = 1, hb = -1; i < maxn; i++) {
64         if (!(i & (i-1))) hb++;
65         rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
66     }
67
68 template<typename T>
69 void NTT(vector<T>& a, bool inv=false) {
70     int _n = (int)a.size();
71     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
72     int n = 1<<k;
73     a.resize(n, 0);
74
75     short shift = maxk-k;
76     for (int i = 0; i < n; i++)
77         if (i > (rev[i]>>shift))
78             swap(a[i], a[rev[i]>>shift]);
79     for (int len = 2, half = 1, div = maxn>>1; len <= n;
80         len<<=1, half<<=1, div>>=1) {
81         for (int i = 0; i < n; i += len) {
82             for (int j = 0; j < half; j++) {
83                 T u = a[i+j];
84                 T v = mmul(a[i+j+half], (inv ? iX[j*div] : X[j*div]));
85                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
86                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
87             }
88         }
89     }
90
91     if (inv) {
92         T dn = pw(n, MOD-2);
93         for (auto& x : a) {
94             x = mmul(x, dn);
95         }
96     }
97
98 template<typename T>
99 inline void shrink(vector<T>& a) {
100     int cnt = (int)a.size();
101     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
102     a.resize(max(cnt, 1));
103 }
104
105 template<typename T>
106 vector<T>& operator*=(vector<T>& a, vector<T> b) {
107     int na = (int)a.size();
108     int nb = (int)b.size();
109     a.resize(na + nb - 1, 0);
110     b.resize(na + nb - 1, 0);
111
112     NTT(a); NTT(b);
113     for (int i = 0; i < (int)a.size(); i++)
114         a[i] = mmul(a[i], b[i]);
115     NTT(a, true);
116
117     shrink(a);
118     return a;
119 }
120
121 inline ll crt(ll a0, ll a1, ll m1, ll m2, ll
122     inv_m1_mod_m2){
123     // x ≡ a0 (mod m1), x ≡ a1 (mod m2)
124     // t = (a1 - a0) * inv(m1) mod m2
125     // x = a0 + t * m1 (mod m1*m2)
126     ll t = chmod(a1 - a0);
127     if (t < 0) t += m2;
128     t = (ll)((__int128)t * inv_m1_mod_m2 % m2);
129     return a0 + (ll)((__int128)t * m1);
130 }
131
132 void mul_crt() {
133     // a copy to a1, a2 | b copy to b1, b2
134     ll M1 = 998244353, M2 = 1004535809;

```



```

124 g = 3; set_mod(M1); init_ntt(); a1 *= b1;
125 g = 3, set_mod(M2); init_ntt(); a2 *= b2;
126
127 ll inv_m1_mod_m2 = pw(M1, M2 - 2);
128 for (int i = 2; i <= 2 * k; i++)
129     cout << crt(a1[i], a2[i], M1, M2, inv_m1_mod_m2
130         ) << ' ';
131 cout << endl;
132 }
133 /* P = r*2^k + 1
134 P          r    k    g
135 998244353    119 23    3
136 1004535809   479 21    3
137
138 P          r    k    g
139 3           1    1    2
140 5           1    2    2
141 17          1    4    3
142 97          3    5    5
143 193         3    6    5
144 257         1    8    3
145 7681        15    9   17
146 12289       3   12   11
147 40961       5   13    3
148 65537       1   16    3
149 786433      3   18   10
150 5767169     11   19    3
151 7340033     7   20    3
152 23068673    11   21    3
153 104857601   25   22    3
154 167772161   5   25    3
155 469762049   7   26    3
156 1004535809  479  21    3
157 2013265921  15   27   31
158 2281701377  17   27    3
159 3221225473  3   30    5
160 75161927681 35   31    3
161 77309411329 9    33    7
162 206158430209 3   36   22
163 2061584302081 15  37    7
164 2748779069441 5   39    3
165 6597069766657 3   41    5
166 39582418599937 9   42    5
167 79164837199873 9   43    5
168 263882790666241 15 44    7
169 1231453023109121 35 45    3
170 1337006139375617 19 46    3
171 3799912185593857 27 47    5
172 4222124650659841 15 48   19
173 7881299347898369 7   50    6
174 31525197391593473 7   52    3
175 180143985094819841 5   55    6
176 1945555039024054273 27 56    5
177 4179340454199820289 29 57    3
178 9097271247288401921 505 54    6 */

```

## 10.14 Counting Primes

```

1 // prime_count — #primes in [1..n] (O(n^{2/3}) time, O(
2   (sqrt(n)) memory)
3
4 using u64 = unsigned long long;
5 static inline u64 prime_count(u64 n){
6     if(n<=1) return 0;
7     int v = (int)floor(sqrt((long double)n));
8     int s = (v+1)>>1, pc = 0;
9     vector<int> smalls(s), roughs(s), skip(v+1);
10    vector<long long> larges(s);
11
12    for(int i=0;i<s;++i){
13        smalls[i]=i;
14        roughs[i]=2*i+1;
15        larges[i]=(long long)((n/roughs[i]-1)>>1);
16    }
17
18    for(int p=3;p<=v;p+=2) if(!skip[p]){
19        int q = p*p;
20        if(1LL*q*q > (long long)n) break;
21        skip[p]=1;
22        for(int i=q;i<=v;i+=2*p) skip[i]=1;

```

```

22    int ns=0;
23    for(int k=0;k<s;++k){
24        int i = roughs[k];
25        if(skip[i]) continue;
26        u64 d = (u64)i * (u64)p;
27        long long sub = (d <= (u64)v)
28            ? larges[smalls[(int)(d>>1)] - pc]
29            : smalls[(int)((n/d - 1) >> 1)];
30        larges[ns] = larges[k] - sub + pc;
31        roughs[ns++] = i;
32    }
33    s = ns;
34    for(int i=(v-1)>>1, j=((v/p)-1)|1; j>=p; j-=2){
35        int c = smalls[j>>1] - pc;
36        for(int e=(j*p)>>1; i>=e; --i) smalls[i] -= c;
37    }
38    ++pc;
39    }
40
41    larges[0] += 1LL*(s + 2*(pc-1))*(s-1) >> 1;
42    for(int k=1;k<s;++k) larges[0] -= larges[k];
43
44    for(int l=1;l<s;++l){
45        int q = roughs[l];
46        u64 m = n / (u64)q;
47        long long t = 0;
48        int e = smalls[(int)((m/q - 1) >> 1)] - pc;
49        if(e < l+1) break;
50        for(int k=l+1;k<=e;++k) t += smalls[(int)((m/ (u64)
51            roughs[k] - 1) >> 1)];
52        larges[0] += t - 1LL*(e - l)*(pc + l - 1);
53    }
54    return (u64)(larges[0] + 1);
55 }

```

## 10.15 Linear Sieve for Other Number Theoretic Functions

```

1 // linear_sieve(n, primes, lp, phi, mu, d, sigma)
2 // Outputs over the index range 0..n (n >= 1):
3 // primes : all primes in [2..n], increasing.
4 // lp      : lowest prime factor; lp[1]=0, lp[x] is
5 //            the smallest prime dividing x.
6 // phi     : Euler totient, phi[x] = |{1<=k<=x : gcd(k
7 //            ,x)=1}|. Multiplicative.
8 // mu      : Möbius; mu[1]=1, mu[x]=0 if x has a
9 //            squared prime factor, else (-1)^{#distinct primes}.
10 // d       : number of divisors; if x=∏ p_i^{e_i},
11 //            then d[x]=∏(e_i+1). Multiplicative.
12 // sigma   : sum of divisors; if x=∏ p_i^{e_i}, then
13 //            sigma[x]=∏(1+p_i+...+p_i^{e_i}). (use ll)
14 //
15 // Complexity: O(n) time, O(n) memory.
16 // Notes: Arrays are resized inside; primes is cleared
17 //         and reserved. sigma uses ll to avoid 32-bit
18 //         overflow.
19
20 static inline void linear_sieve(
21     int n,
22     std::vector<int> &primes,
23     std::vector<int> &lp,
24     std::vector<int> &phi,
25     std::vector<int> &mu,
26     std::vector<int> &d,
27     std::vector<ll> &sigma
28 ) {
29     lp.assign(n + 1, 0); phi.assign(n + 1, 0); mu.assign(
30         n + 1, 0); d.assign(n + 1, 0); sigma.assign(n +
31         1, 0);
32     primes.clear(); primes.reserve(n > 1 ? n / 10 : 0);
33     std::vector<int> cnt(n + 1, 0), core(n + 1, 1);
34     std::vector<ll> p_pow(n + 1, 1), sum_p(n + 1, 1);
35     phi[1] = mu[1] = d[1] = sigma[1] = 1;
36
37     for (int i = 2; i <= n; ++i) {
38         if (!lp[i]) {
39             lp[i] = i; primes.push_back(i);
40             phi[i] = i - 1; mu[i] = -1; d[i] = 2;
41             cnt[i] = 1; p_pow[i] = i; core[i] = 1;
42             sum_p[i] = 1 + (ll)i; sigma[i] = sum_p[i];

```



```

34 }
35 for (int p : primes) {
36     long long ip = 1LL * i * p;
37     if (ip > n) break;
38     lp[ip] = p;
39     if (p == lp[i]) {
40         cnt[ip] = cnt[i] + 1; p_pow[ip] = p_pow[i] * p;
41         core[ip] = core[i];
42         sum_p[ip] = sum_p[i] + p_pow[ip];
43         phi[ip] = phi[i] * p; mu[ip] = 0;
44         d[ip] = d[core[ip]] * (cnt[ip] + 1);
45         sigma[ip] = sigma[core[ip]] * sum_p[ip];
46         break; // critical for linear complexity
47     } else {
48         cnt[ip] = 1; p_pow[ip] = p; core[ip] = i;
49         sum_p[ip] = 1 + (ll)p;
50         phi[ip] = phi[i] * (p - 1); mu[ip] = -mu[i];
51         d[ip] = d[i] * 2;
52         sigma[ip] = sigma[i] * sum_p[ip];
53     }
54 }
55 }
56 }
57 // Optional helper: factorize x in O(log x) using lp (
58 // requires x in [2..n])
59 static inline std::vector<std::pair<int,int>> factorize
60 (int x, const std::vector<int>& lp) {
61     std::vector<std::pair<int,int>> res;
62     while (x > 1) {
63         int p = lp[x], e = 0;
64         do { x /= p; ++e; } while (x % p == 0);
65         res.push_back({p, e});
66     }
67     return res;
68 }

```

## 11 Linear Algebra

### 11.1 Gaussian-Jordan Elimination

```

1 int n; vector<vector<ll>> > v;
2 void gauss(vector<vector<ll>>& v) {
3     int r = 0;
4     for (int i = 0; i < n; i++) {
5         bool ok = false;
6         for (int j = r; j < n; j++) {
7             if (v[j][i] == 0) continue;
8             swap(v[j], v[r]);
9             ok = true; break;
10        }
11        if (!ok) continue;
12        ll div = inv(v[r][i]);
13        for (int j = 0; j < n+1; j++) {
14            v[r][j] *= div;
15            if (v[r][j] >= MOD) v[r][j] %= MOD;
16        }
17        for (int j = 0; j < n; j++) {
18            if (j == r) continue;
19            ll t = v[j][i];
20            for (int k = 0; k < n+1; k++) {
21                v[j][k] -= v[r][k] * t % MOD;
22                if (v[j][k] < 0) v[j][k] += MOD;
23            }
24        }
25    }
26 }

```

### 11.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then  $\det = 0$ , otherwise  $\det = \text{product of diagonal elements}$ .
2. Properties of  $\det$ :
  - Transpose: Unchanged
  - Row Operation 1 - Swap 2 rows:  $-\det$

- Row Operation 2 -  $k\vec{r}_i$ :  $k \times \det$
- Row Operation 3 -  $k\vec{r}_i$  add to  $\vec{r}_j$ : Unchanged

## 12 Flow / Matching

### 12.1 Flow Methods

```

1 // Author: CRYPTOGRAPHER (some modified by Gino)
2 Maximize  $c^T x$  subject to  $Ax \leq b, x \geq 0$ ;
3 with the corresponding symmetric dual problem,
4 Minimize  $b^T y$  subject to  $A^T y \geq c, y \geq 0$ .
5
6 Maximize  $c^T x$  subject to  $Ax \leq b$ ;
7 with the corresponding asymmetric dual problem,
8 Minimize  $b^T y$  subject to  $A^T y = c, y \geq 0$ .
9
10 Maximize  $\sum x$  subject to  $x_i + x_j \leq A_{ij}, x \geq 0$ ;
11 => Maximize  $\sum x$  subject to  $x_i + x_j \leq A_{ij}$ ;
12 => Minimize  $A^T y = \sum A_{ij} y_{ij}$  subject to for all  $v$ 
13     ,  $\sum_{i=v \text{ or } j=v} y_{ij} = 1, y_{ij} \geq 0$ 
14 => possible optimal solution:  $y_{ij} = \{0, 0.5, 1\}$ 
15 =>  $y' = 2y$ :  $\sum_{i=v \text{ or } j=v} y'_{ij} = 2, y'_{ij} = \{0, 1, 2\}$ 
16 => Minimum Bipartite perfect matching/2 ( $V_1=X, V_2=X, E=A$ )
17
18 General Graph:
19  $|Max \text{ Ind. Set}| + |Min \text{ Vertex Cover}| = |V|$ 
20  $|Max \text{ Ind. Edge Set}| + |Min \text{ Edge Cover}| = |V|$ 
21 Bipartite Graph:
22  $|Max \text{ Ind. Set}| = |Min \text{ Edge Cover}|$ 
23  $|Max \text{ Ind. Edge Set}| = |Min \text{ Vertex Cover}|$ 
24
25 To reconstruct the minimum vertex cover, dfs from each
26 unmatched vertex on the left side and with unused edges
27 only. Equivalently, dfs from source with unused edges
28 only and without visiting sink. Then, a vertex is
29 chosen iff. it is on the left side and without visited
30 or on the right side and visited through dfs.
31
32 Minimum Weighted Bipartite Edge Cover:
33 Construct new bipartite graph with  $n+m$  vertices on each
34 side:
35 for each vertex  $u$ , duplicate a vertex  $u'$  on the other
36 side
37 for each edge  $(u,v,w)$ , add edges  $(u,v,w)$  and  $(v',u',w)$ 
38 for each vertex  $u$ , add edge  $(u,u',2w)$  where  $w$  is min
39 edge connects to  $u$ 
40 then the answer is the minimum perfect matching of the
41 new graph (KM)
42
43 Maximum density subgraph (  $\sum W_e + \sum W_v$  ) /  $|V|$ 
44 Binary search on answer:
45 For a fixed  $D$ , construct a Max flow model as follow:
46 Let  $S$  be Sum of all weight ( or inf )
47 1. from source to each node with  $cap = S$ 
48 2. For each  $(u,v,w)$  in  $E$ ,  $(u \rightarrow v, cap=w)$ ,  $(v \rightarrow u, cap=w)$ 
49 3. For each node  $v$ , from  $v$  to sink with  $cap = S + 2 * D$ 
50    -  $deg[v] - 2 * (W \text{ of } v)$ 
51 where  $deg[v] = \sum \text{weight of edge associated with } v$ 
52 If  $\text{maxflow} < S * |V|$ ,  $D$  is an answer.
53
54 Requiring subgraph: all vertex can be reached from
55 source with
56 edge whose  $cap > 0$ .
57
58 Maximum closed subgraph
59 1. connect source with positive weighted vertex(
60    capacity=weight)
61 2. connect sink with negative weighted vertex(capacity
62    =-weight)
63 3. make capacity of the original edges = inf
64 4. ans = sum(positive weighted vertex weight) - (max
65    flow)
66
67 (Node-disjoint) Min DAG Path Cover (用最少路徑覆蓋所有
68 點)
69 Node disjoint: 拆出來的路徑不能共用同一個點
70 將一個點  $u$  裂成  $u_+$  和  $u_-$ ，代表進入和出去
71 對於一條邊  $(u, v)$  建邊  $u_- \rightarrow v_+$ 

```

```

61 1. +點們和 -點們形成一張二分圖
62 2. 最差的答案是  $n$ ，代表每個點自己一個點就是一條路徑
63 3. 只要一組  $(u-, v+)$  匹配成功那對應到的答案恰好會  $-1$ 
64 >>> ans = n - 這張二分圖的最大匹配

```

### General DAG Path Cover

```

65 跟 Node-disjoint 版本差在點可以被很多條路共用
66 建邊方式 Node-disjoint 差別在條件比較鬆
67 Node-disjoint:  $u- \Rightarrow v+$  只能在  $(u, v)$  有邊時建
68 General:  $u- \Rightarrow v+$  在  $u$  能走到  $v$  的時候就建

```

### Dilworth Theorem

```

71 反鏈：一些節點的集合，滿足這些節點互相無法抵達
72 最大反鏈 = 最小 General DAG Path Cover

```

## 12.2 Dinic

```

1 // Author: Benson (Extensions by Gino)
2 // Function: Max Flow,  $O(V^2 E)$ 
3 // Usage: Call init(n) first, then add(u, v, w) based
4 // on your model
5 // (!) vertices 0-based
6 // >>> flow() := return max flow
7 // >>> find_cut() := return min cut + store cut set in
8 // dinic.cut
9 // >>> find_matching() := return |M| + store matching
10 // plan in dinic.matching
11 // >>> flow_decomposition := return max flow + store
12 // decomposition in dinic.D
13 #define int long long
14 #define eb emplace_back
15 #define ALL(a) a.begin(), a.end()
16 struct Dinic {
17     struct Edge {
18         // t: to | c: original capacity | c: current
19         // capacity | r: residual edge | f: current flow
20         int t, c, c_r, f;
21         bool fw; // is in forward-edge graph
22         Edge() {}
23         Edge(int _t, int _c, int _r, bool _fw, int _f=0):
24             t(_t), c(_c), c_r(_c), r(_r), fw(_fw), f(_f) {}
25     };
26     vector<vector<Edge>> G;
27     vector<int> dis, iter;
28     int n, s, t;
29     void init(int _n) {
30         n = _n;
31         G.resize(n), dis.resize(n), iter.resize(n);
32         for(int i = 0; i < n; ++i)
33             G[i].clear();
34     }
35     void add(int a, int b, int c) {
36         G[a].eb(b, c, G[b].size(), true);
37         G[b].eb(a, 0, G[a].size() - 1, false);
38     }
39     bool bfs() {
40         fill(ALL(dis), -1);
41         dis[s] = 0;
42         queue<int> que;
43         que.push(s);
44         while(!que.empty()) {
45             int u = que.front(); que.pop();
46             for(auto& e : G[u]) {
47                 if(e.c > 0 && dis[e.t] == -1) {
48                     dis[e.t] = dis[u] + 1;
49                     que.push(e.t);
50                 }
51             }
52         }
53         return dis[t] != -1;
54     }
55     int dfs(int u, int cur) {
56         if(u == t) return cur;
57         for(int &i = iter[u]; i < (int)G[u].size(); ++i) {
58             auto& e = G[u][i];
59             if(e.c > 0 && dis[u] + 1 == dis[e.t]) {
60                 int ans = dfs(e.t, min(cur, e.c));
61                 if(ans > 0) {
62                     G[e.t][e.r].c += ans;
63                     e.c -= ans;
64                     return ans;
65                 }
66             }
67         }
68     }
69 }

```

```

59 return 0;
60 }
61 // find max flow
62 int flow(int a, int b) {
63     s = a, t = b;
64     int ans = 0;
65     while(bfs()) {
66         fill(ALL(iter), 0);
67         int tmp;
68         while((tmp = dfs(s, INF)) > 0)
69             ans += tmp;
70     }
71     return ans;
72 }
73 // min cut plan
74 vector<pair<pair<int, int>, int>> cut;
75 int find_cut(int a, int b) {
76     int cut_sz = flow(a, b);
77     vector<int> vis(n, 0);
78     cut.clear();
79     function<void(int)> dfs = [&](int u) {
80         vis[u] = 1;
81         for(auto& e : G[u])
82             if(e.c > 0 && !vis[e.t])
83                 dfs(e.t);
84     };
85     dfs(a);
86     for(int u = 0; u < n; u++)
87         if(vis[u])
88             for(auto& e : G[u])
89                 if(!vis[e.t])
90                     cut.eb(make_pair(make_pair(u, e.t), G[e.t][
91                         e.r].c));
92     return cut_sz;
93 }
94 // bipartite matching plan
95 vector<pair<int, int>> matching;
96 int find_matching(int Xstart, int Xend, int Ystart,
97     int Yend, int a, int b) {
98     int msz = flow(a, b);
99     matching.clear();
100     for(int x = Xstart; x <= Xend; x++)
101         for(auto& e : G[x])
102             if(e.c == 0 && Ystart <= e.t && e.t <= Yend)
103                 matching.emplace_back(make_pair(x, e.t));
104     return msz;
105 }
106 // flow decomposition
107 vector<pair<int, vector<int>>> D; // (flow amount, [
108     path p1 ... pk])
109 int flow_decomposition(int a, int b) {
110     int mxflow = flow(a, b);
111     vector<vector<Edge>> fG(n); // graph consists of
112     // forward edges
113     for(int u = 0; u < n; u++) {
114         for(auto& e : G[u]) {
115             if(e.fw) {
116                 e.f = e.c - e.c_r;
117                 if(e.f > 0) fG[u].eb(e);
118             }
119         }
120     }
121     vector<int> vis;
122     function<int(int, int)> dfs = [&](int u, int cur) {
123         if(u == b) {
124             D.back().second.eb(u);
125             return cur;
126         }
127         vis[u] = 1;
128         for(auto& e : fG[u]) {
129             if(e.f > 0 && !vis[e.t]) {
130                 int ans = dfs(e.t, min(cur, e.f));
131                 if(ans > 0) {
132                     e.f -= ans;
133                     D.back().second.eb(u);
134                     return ans;
135                 }
136             }
137         }
138         return 0LL;
139     };
140     D.clear();
141     int quota = mxflow;
142     while(quota > 0) {

```

```

137     D.emplace_back(make_pair(0, vector<int>()));
138     vis.assign(n, 0);
139     int f = dfs(a, INF);
140     if (f == 0) break;
141     reverse(D.back().second.begin(), D.back().second.
        end());
142     D.back().first = f, quota -= f;
143 }
144 return mxflow;
145 }
146 };

```

## 12.3 ISAP

```

1 // Author: CRYPTOGRAPHER
2 #define SZ(c) ((int)(c).size())
3 static const int MAXV=50010;
4 static const int INF =1000000;
5 struct Maxflow{
6     struct Edge{
7         int v,c,r;
8         Edge(int _v,int _c,int _r):v(_v),c(_c),r(_r){}
9     };
10    int s,t; vector<Edge> G[MAXV];
11    int iter[MAXV],d[MAXV],gap[MAXV],tot;
12    void init(int n,int _s,int _t){
13        tot=n,s=_s,t=_t;
14        for(int i=0;i<=tot;i++){
15            G[i].clear(); iter[i]=d[i]=gap[i]=0;
16        }
17    }
18    void addEdge(int u,int v,int c){
19        G[u].push_back(Edge(v,c,SZ(G[v])));
20        G[v].push_back(Edge(u,0,SZ(G[u])-1));
21    }
22    int DFS(int p,int flow){
23        if(p==t) return flow;
24        for(int &i=iter[p];i<SZ(G[p]);i++){
25            Edge &e=G[p][i];
26            if(e.c>0&&d[p]==d[e.v]+1){
27                int f=DFS(e.v,min(flow,e.c));
28                if(f){ e.c-=f; G[e.v][e.r].c+=f; return f; }
29            }
30        }
31        if(--gap[d[p]]==0) d[s]=tot;
32        else{ d[p]++; iter[p]=0; ++gap[d[p]]; }
33        return 0;
34    }
35    int flow(){
36        int res=0;
37        for(res=0,gap[0]=tot;d[s]<tot;res+=DFS(s,INF));
38        return res;
39    } // reset: set iter,d,gap to 0
40 } flow;

```

## 12.4 Bounded Max Flow

```

1 // Author: CRYPTOGRAPHER
2 // Max flow with lower/upper bound on edges
3 // use with ISAP, L,r,a,b must be filled
4 int in[N],out[N],l[M],r[M],a[M],b[M];
5 int solve(int n, int m, int s, int t){
6     flow.init(n+2,n,n+1);
7     for(int i=0;i<m;i++){
8         in[r[i]]+=a[i]; out[l[i]]+=a[i];
9         flow.addEdge(l[i],r[i],b[i]-a[i]);
10        // flow from l[i] to r[i] must in [a[i], b[i]]
11    }
12    int nd=0;
13    for(int i=0;i<=n;i++){
14        if(in[i]<out[i]){
15            flow.addEdge(i,flow.t,out[i]-in[i]);
16            nd+=out[i]-in[i];
17        }
18        if(out[i]<in[i])
19            flow.addEdge(flow.s,i,in[i]-out[i]);
20    }
21    // original sink to source
22    flow.addEdge(t,s,INF);
23    if(flow.flow()!=nd) return -1; // no solution

```

```

24    int ans=flow.G[s].back().c; // source to sink
25    flow.G[s].back().c=flow.G[t].back().c=0;
26    // take out super source and super sink
27    for(size_t i=0;i<flow.G[flow.s].size();i++){
28        Maxflow::Edge &e=flow.G[flow.s][i];
29        flow.G[flow.s][i].c=0; flow.G[e.v][e.r].c=0;
30    }
31    for(size_t i=0;i<flow.G[flow.t].size();i++){
32        Maxflow::Edge &e=flow.G[flow.t][i];
33        flow.G[flow.t][i].c=0; flow.G[e.v][e.r].c=0;
34    }
35    flow.addEdge(flow.s,s,INF);flow.addEdge(t,flow.t,INF);
36    ;
37    flow.reset(); return ans+flow.flow();

```

## 12.5 MCMF

```

1 // Author: CRYPTOGRAPHER
2 // Usage:
3 // 1. MCMF.init(n, s, t)
4 // 2. MCMF.add(u, v, cap, cost)
5 // 3. auto [max_flow, min_cost] = MCMF.flow()
6 typedef int Tcost;
7 const int MAXV = 20010;
8 const int INFf = 1000000;
9 const Tcost INFc = 1e9;
10 struct MCMF {
11     struct Edge{
12         int v, cap;
13         Tcost w;
14         int rev;
15         bool fw;
16         Edge(){}
17         Edge(int t2, int t3, Tcost t4, int t5, bool t6)
18             : v(t2), cap(t3), w(t4), rev(t5), fw(t6) {}
19     };
20    int V, s, t;
21    vector<Edge> G[MAXV];
22    void init(int n){
23        V = n;
24        for(int i = 0; i <= V; i++) G[i].clear();
25    }
26    void add(int a, int b, int cap, Tcost w){
27        G[a].push_back(Edge(b, cap, w, (int)G[b].size(),
28            true));
29        G[b].push_back(Edge(a, 0, -w, (int)G[a].size()-1,
30            false));
31    }
32    Tcost d[MAXV];
33    int id[MAXV], mom[MAXV];
34    bool inqu[MAXV];
35    queue<int> q;
36    pair<int, Tcost> flow(int _s, int _t){
37        s = _s, t = _t;
38        int mxf = 0; Tcost mnc = 0;
39        while(1){
40            fill(d, d+1+V, INFc); // need to use type cast
41            fill(inqu, inqu+1+V, 0);
42            fill(mom, mom+1+V, -1);
43            mom[s] = s;
44            d[s] = 0;
45            q.push(s); inqu[s] = 1;
46            while(q.size()){
47                int u = q.front(); q.pop();
48                inqu[u] = 0;
49                for(int i = 0; i < (int) G[u].size(); i++){
50                    Edge &e = G[u][i];
51                    int v = e.v;
52                    if(e.cap > 0 && d[v] > d[u]+e.w){
53                        d[v] = d[u]+e.w;
54                        mom[v] = u;
55                        id[v] = i;
56                        if(!inqu[v]) q.push(v), inqu[v] = 1;
57                    }
58                }
59            }
60            if(mom[t] == -1) break ;
61            int df = INFf;
62            for(int u = t; u != s; u = mom[u])
63                df = min(df, G[mom[u]][id[u]].cap);

```

```

62     for(int u = t; u != s; u = mom[u]){
63         Edge &e = G[mom[u]][id[u]];
64         e.cap -= df;
65         G[e.v][e.rev].cap += df;
66     }
67     mxf += df;
68     mnc += df*d[t];
69 }
70 return make_pair(mxf, mnc);
71 }
72 };

```

## 12.6 Hopcroft-Karp

```

1 // Author: Gino
2 // Function: Max Bipartite Matching in  $O(V \sqrt{E})$ 
3 // Usage:
4 // >>> init(nx, ny, m) -> add(x, y (+nx))
5 // >>> hk.max_matching() := the matching plan stores in
6 //   mx, my
7 // >>> hk.min_vertex_cover() := the vertex cover plan
8 //   stores in vcover
9 // (!) vertices are 0-based: X = [0, nx), Y = [nx, nx+
10 //   ny)
11 struct HopcroftKarp {
12     int n, nx, ny;
13     vector<vector<int>> > G;
14     vector<int> mx, my;
15     void init(int _nx, int _ny) {
16         nx = _nx, ny = _ny;
17         n = nx + ny;
18         G.clear(); G.resize(n);
19     }
20     void add(int x, int y) {
21         G[x].emplace_back(y);
22         G[y].emplace_back(x);
23     }
24     int max_matching() {
25         vector<int> dis, vis;
26         mx.clear(); mx.resize(n, -1);
27         my.clear(); my.resize(n, -1);
28
29         function<bool(int)> dfs = [&](int x) {
30             vis[x] = true;
31             for (auto& y : G[x]) {
32                 int px = my[y];
33                 if (px == -1 ||
34                     (dis[px] == dis[x]+1 &&
35                      !vis[px] && dfs(px))) {
36                     mx[x] = y;
37                     my[y] = x;
38                     return true;
39                 }
40             }
41             return false;
42         };
43
44         while (true) {
45             queue<int> q;
46             dis.clear(); dis.resize(n, -1);
47             for (int x = 0; x < nx; x++) {
48                 if (mx[x] == -1) {
49                     dis[x] = 0;
50                     q.push(x);
51                 }
52             }
53             while (!q.empty()) {
54                 int x = q.front(); q.pop();
55                 for (auto& y : G[x]) {
56                     if (my[y] != -1 && dis[my[y]] == -1) {
57                         dis[my[y]] = dis[x] + 1;
58                         q.push(my[y]);
59                     }
60                 }
61             }
62             bool brk = true;
63             vis.clear(); vis.resize(n, 0);
64             for (int x = 0; x < nx; x++) {
65                 if (mx[x] == -1 && dfs(x)) {
66                     brk = false;
67                 }
68             }
69             if (brk) break;
70         }
71         int ans = 0;
72         for (int x = 0; x < nx; x++) if (mx[x] != -1) ans++;
73     }
74 };

```

```

64     return ans;
65 }
66 vector<int> vcover;
67 int min_vertex_cover() {
68     int ans = max_matching();
69     vcover.clear();
70
71     vector<int> vis(n, 0);
72     function<void(int)> dfs = [&](int x) {
73         vis[x] = true;
74         for (auto& y : G[x]) {
75             if (y == mx[x] || my[y] == -1 || vis[y])
76                 continue;
77             vis[y] = true;
78             dfs(my[y]);
79         }
80     };
81
82     for (int x = 0; x < nx; x++) if (mx[x] == -1) dfs(x);
83     for (int x = 0; x < nx; x++) if (!vis[x]) vcover.emplace_back(x);
84     for (int y = nx; y < nx + ny; y++) if (vis[y])
85         vcover.emplace_back(y);
86     return ans;
87 }
88 } hk;

```

## 12.7 Cover / Independent Set

- 1 最大邊獨立集 (Ie) 就是最大匹配 (M)
- 2 二分圖上，M 和 Cv 對偶
- 3 對任何圖都有  $|Iv| + |Cv| = |V|$
- 4 對任何圖都有  $|Ie| + |Ce| = |V|$
- 5 二分圖最小帶權點覆蓋 => 建模 (s, u, w[u]) (u, v, INF) (v, t, w[v]) 算最小割

## 12.8 Kuhn Munkres

```

1 // Author: CRYPTOGRAPHER
2 static const int MXN=2001; // 1-based
3 static const ll INF=0x3f3f3f3f;
4 struct KM { // max weight, for min negate the weights
5     int n, mx[MXN], my[MXN], pa[MXN]; bool vx[MXN], vy[MXN];
6     ll g[MXN][MXN], lx[MXN], ly[MXN], sy[MXN];
7     void init(int _n) {
8         n = _n; for (int i=1; i<=n; i++) fill(g[i], g[i]+n+1, 0);
9     }
10    void addEdge(int x, int y, ll w) { g[x][y] = w; }
11    void augment(int y) {
12        for (int x, z; y; y=z) x=pa[y], z=mx[x], my[y]=x, mx[x]=y;
13    }
14    void bfs(int st) {
15        for (int i=1; i<=n; i++) sy[i]=INF, vx[i]=vy[i]=0;
16        queue<int> q; q.push(st);
17        for (; ; ) {
18            while (q.size()) {
19                int x=q.front(); q.pop(); vx[x]=1;
20                for (int y=1; y<=n; y++) if (!vy[y]) {
21                    ll t=lx[x]+ly[y]-g[x][y];
22                    if (t==0) {
23                        pa[y]=x;
24                        if (!my[y]) { augment(y); return; }
25                        vy[y]=1, q.push(my[y]);
26                    } else if (sy[my[y]]>t) pa[y]=x, sy[my[y]]=t;
27                }
28            }
29            ll cut=INF;
30            for (int y=1; y<=n; y++)
31                if (!vy[y] && cut>sy[y]) cut=sy[y];
32            for (int j=1; j<=n; j++) {
33                if (vx[j]) lx[j]-=cut;
34                if (vy[j]) ly[j]+=cut;
35                else sy[j]-=cut;
36            }
37            for (int y=1; y<=n; y++) if (!vy[y] && sy[y]==0) {
38                if (!my[y]) { augment(y); return; }
39                vy[y]=1, q.push(my[y]);
40            }
41        }
42    }
43 };

```

```

41 ll solve(){
42     fill(mx,mx+n+1,0);fill(my,my+n+1,0);
43     fill(ly,ly+n+1,0);fill(lx,lx+n+1,-INF);
44     for(int x=1;x<=n;++x) for(int y=1;y<=n;++y)
45         lx[x]=max(lx[x],g[x][y]);
46     for(int x=1;x<=n;++x) bfs(x);
47     ll ans=0;
48     for(int y=1;y<=n;++y) ans+=g[my[y]][y];
49     return ans;
50 }
51 }graph;

```

- Very large prime numbers:

1000001333	1000500889	2500001909
2000000659	900004151	850001359

- $\pi(n) \equiv$  Number of primes  $\leq n \approx n/((\ln n) - 1)$   
 $\pi(100) = 25, \pi(200) = 46$   
 $\pi(500) = 95, \pi(1000) = 168$   
 $\pi(2000) = 303, \pi(4000) = 550$   
 $\pi(10^4) = 1229, \pi(10^5) = 9592$   
 $\pi(10^6) = 78498, \pi(10^7) = 664579$

## 13 Combinatorics

### 13.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

### 13.2 Bertrand's Ballot Theorem

- $A$  always  $> B$ :  $C(p+q, p) - 2C(p+q-1, p)$
- $A$  always  $\geq B$ :  $C(p+q, p) \times \frac{p+1-q}{p+1}$

### 13.3 Burnside's Lemma

Let  $X$  be the original set.

Let  $G$  be the group of operations acting on  $X$ .

Let  $X^g$  be the set of  $x$  not affected by  $g$ .

Let  $X/G$  be the set of orbits.

Then the following equation holds:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

## 14 Special Numbers

### 14.1 Fibonacci Series

1	1	1	2	3
5	5	8	13	21
9	34	55	89	144
13	233	377	610	987
17	1597	2584	4181	6765
21	10946	17711	28657	46368
25	75025	121393	196418	317811
29	514229	832040	1346269	2178309
33	3524578	5702887	9227465	14930352

$$f(45) \approx 10^9, f(88) \approx 10^{18}$$

### 14.2 Prime Numbers

- First 50 prime numbers:

1	2	3	5	7	11
6	13	17	19	23	29
11	31	37	41	43	47
16	53	59	61	67	71
21	73	79	83	89	97
26	101	103	107	109	113
31	127	131	137	139	149
36	151	157	163	167	173
41	179	181	191	193	197
46	199	211	223	227	229