

Contents

1 BASIC	4
1.1 <code>bigInt.h</code>	4
1.2 <code>bitIntPY.py</code>	12
1.3 <code>bitset.h</code>	13
1.4 <code>pbd.h</code>	14
1.5 <code>时间戳优化.h</code>	15
2 Ds	16
2.1 <code>dsu.h</code>	16
2.2 <code>dsu_classification.h</code>	17
2.3 <code>dsu_weighted.h</code>	18
2.4 <code>segTree_add.h</code>	20
2.5 <code>segTree_add_setto.h</code>	22
2.6 <code>segTree_mul_add.h</code>	24
2.7 <code>segTree_MX_MI.h</code>	26
2.8 <code>segTree_历史最值.h</code>	29
2.9 <code>SparseTable.h</code>	31
2.10 <code>twoDimPrfxSum.h</code>	32
2.11 <code>主席树.h</code>	34
2.12 <code>单调队列.h</code>	35
2.13 <code>左偏树.h</code>	36
2.14 <code>平衡树.h</code>	39
2.15 <code>树状数组.h</code>	41
3 GEO	44
3.1 <code>Rotating_Calipers.h</code>	44
3.2 <code>三维封装.h</code>	46
3.3 <code>距离转化.typ</code>	49
3.4 <code>跨立实验.h</code>	50
4 GRAPH	52
4.1 <code>2-SAT.h</code>	52
4.2 <code>Hopcroft-Carp.h</code>	55
4.3 <code>图上大封装.h</code>	57
4.4 <code>奇偶环.h</code>	62
4.5 FLOW	66
4.5.1 <code>0_introduction.typ</code>	66
4.5.2 <code>DINIC.h</code>	67
4.5.3 <code>DJ.h</code>	71
4.5.4 <code>EK.h</code>	73
4.5.5 <code>HLPP.h</code>	75
4.5.6 <code>ISPA.h</code>	78
4.5.7 <code>min_Cost.h</code>	81

4.6 PATH	84
4.6.1 johonson.h	84
4.6.2 K_path.h	87
4.6.3 SPFA+SLE.h	90
4.6.4 SPFA.h	92
4.6.5 判断欧拉回路或通路.h	94
4.6.6 换边最短路.typ	96
4.6.7 求欧拉回路或通路.h	101
4.7 TREE	103
4.7.1 kruskal 重构树.h	103
4.7.2 lca.h	105
4.7.3 中心.h	107
4.7.4 支配树.h	108
4.7.5 斯坦纳树.h	111
4.7.6 直径.h	113
4.7.7 虚树.h	114
4.7.8 重心.h	115
4.7.9 重链剖分.h	116
5 MATH	120
5.1 LINNER	120
5.1.1 basis.h	120
5.1.2 det.h	122
5.1.3 GaussianELI.h	123
5.1.4 单纯形法.cpp	124
5.2 NUMBER_THEORY	126
5.2.1 basic.h	126
5.2.2 CRT.h	127
5.2.3 Euler_phi.h	128
5.2.4 Euler_sieve.h	129
5.2.5 factor_pr.h	130
5.2.6 factror_pri.h	134
5.2.7 整除分块.typ	136
5.2.8 组合数.h	138
5.2.9 莫反.h	139
5.3 OTHER	141
5.3.1 Frac.h	141
5.4 POLYNOMIAL	143
5.4.1 AxB_prob.h	147
5.4.2 CDQ+NTT_FTT.h	149
5.4.3 FFT_butterfly.h	152
5.4.4 NTT.h	154

5.4.5	NTT_INV.h	156
5.4.6	sqrt.h	158
6	Misc	161
6.1	莫队.h	161
7	STRING	162
7.1	AC_automaton.h	162
7.2	compress_print.h	164
7.3	get_occr.h	165
7.4	hash_print.h	166
7.5	KMP.h	168
7.6	Manacher.h	169
7.7	Palindromic_automaton.h	170
7.8	trie_Tree.h	171

1 BASIC

1.1 bigInt.h

```

1 namespace BigIntMiniNS {
2   const int32_t COMPRESS_MOD = 10000;
3   const uint32_t COMPRESS_DIGITS = 4;
4
5   const uint32_t BIGINT_MUL_THRESHOLD = 60;
6   const uint32_t BIGINT_DIVIDEDIV_THRESHOLD = BIGINT_MUL_THRESHOLD * 3;
7
8   template <typename T> inline T high_digit(T digit) { return digit /
9   (T)COMPRESS_MOD; }
10
11   template <typename T> inline uint32_t low_digit(T digit) { return (uint32_t)
12   (digit % (T)COMPRESS_MOD); }
13
14   class BigIntMini {
15   protected:
16     typedef uint32_t base_t;
17     typedef int32_t carry_t;
18     typedef uint32_t ucarry_t;
19     int sign;
20     std::vector<base_t> v;
21     typedef BigIntMini BigInt_t;
22     template <typename _Tx, typename _Ty> static inline void carry(_Tx &add,
23     _Ty &baseval, _Tx newval) {
24       add += newval;
25       baseval = low_digit(add);
26       add = high_digit(add);
27     }
28     template <typename _Tx, typename _Ty> static inline void borrow(_Tx &add,
29     _Ty &baseval, _Tx newval) {
30       add += newval - COMPRESS_MOD + 1;
31       baseval = (_Tx)low_digit(add) + COMPRESS_MOD - 1;
32       add = high_digit(add);
33     }
34
35     bool raw_less(const BigInt_t &b) const {
36       if (v.size() != b.size()) return v.size() < b.size();
37       for (size_t i = v.size() - 1; i < v.size(); i--)
38         if (v[i] != b.v[i]) return v[i] < b.v[i];
39       return false; // eq
40     }
41
42     bool raw_eq(const BigInt_t &b) const {
43       if (v.size() != b.size()) return false;
44       for (size_t i = 0; i < v.size(); ++i)
45         if (v[i] != b.v[i]) return false;
46       return true;
47     }
48
49     BigInt_t &raw_add(const BigInt_t &b) {
50       if (v.size() < b.size()) v.resize(b.size());
51       ucarry_t add = 0;
52       for (size_t i = 0; i < b.v.size(); i++)
53         carry(add, v[i], (ucarry_t)(v[i] + b.v[i]));
54       for (size_t i = b.v.size(); add && i < v.size(); i++)
55         carry(add, v[i], (ucarry_t)v[i]);
56       add ? v.push_back((base_t)add) : trim();
57     }
58   };
59 }

```

```

51     return *this;
52 }
53 BigInt_t &raw_offset_add(const BigInt_t &b, size_t offset) {
54     ucarry_t add = 0;
55     for (size_t i = 0; i < b.size(); ++i)
56         carry(add, v[i + offset], (ucarry_t)(v[i + offset] + b.v[i]));
57     for (size_t i = b.size() + offset; add; ++i)
58         carry(add, v[i], (ucarry_t)v[i]);
59     return *this;
60 }
61 BigInt_t &raw_sub(const BigInt_t &b) {
62     if (v.size() < b.v.size()) v.resize(b.v.size());
63     carry_t add = 0;
64     for (size_t i = 0; i < b.v.size(); i++)
65         borrow(add, v[i], (carry_t)v[i] - (carry_t)b.v[i]);
66     for (size_t i = b.v.size(); add && i < v.size(); i++)
67         borrow(add, v[i], (carry_t)v[i]);
68     if (add) {
69         sign = -sign;
70         add = 1;
71         for (size_t i = 0; i < v.size(); i++)
72             carry(add, v[i], (carry_t)(COMPRESS_MOD - v[i] - 1));
73     }
74     trim();
75     return *this;
76 }
77 BigInt_t &raw_mul_int(uint32_t m) {
78     if (m == 0) {
79         set(0);
80         return *this;
81     } else if (m == 1)
82         return *this;
83     ucarry_t add = 0;
84     for (size_t i = 0; i < v.size(); i++)
85         carry(add, v[i], v[i] * (ucarry_t)m);
86     if (add) v.push_back((base_t)add);
87     return *this;
88 }
89 BigInt_t &raw_mul(const BigInt_t &a, const BigInt_t &b) {
90     v.clear();
91     v.resize(a.size() + b.size());
92     for (size_t i = 0; i < a.size(); i++) {
93         ucarry_t add = 0, av = a.v[i];
94         for (size_t j = 0; j < b.size(); j++)
95             carry(add, v[i + j], v[i + j] + av * b.v[j]);
96         v[i + b.size()] += (base_t)add;
97     }
98     trim();
99     return *this;
100 }
101 // Karatsuba algorithm
102 BigInt_t &raw_mul_karatsuba(const BigInt_t &a, const BigInt_t &b) {
103     if (std::min(a.size(), b.size()) <= BIGINT_MUL_THRESHOLD) return
raw_mul(a, b);
104     BigInt_t ah, al, bh, bl, h, m;
105     size_t split = std::max(std::min((a.size() + 1) / 2, b.size() - 1),
std::min(a.size() - 1, (b.size() + 1) / 2));

```

```

106     al.v.assign(a.v.begin(), a.v.begin() + split);
107     ah.v.assign(a.v.begin() + split, a.v.end());
108     bl.v.assign(b.v.begin(), b.v.begin() + split);
109     bh.v.assign(b.v.begin() + split, b.v.end());
110
111     raw_mul_karatsuba(al, bl);
112     h.raw_mul_karatsuba(ah, bh);
113     m.raw_mul_karatsuba(al + ah, bl + bh);
114     m.raw_sub(*this);
115     m.raw_sub(h);
116     v.resize(a.size() + b.size());
117
118     raw_offset_add(m, split);
119     raw_offset_add(h, split * 2);
120     trim();
121     return *this;
122 }
123 BigInt_t &raw_div(const BigInt_t &a, const BigInt_t &b, BigInt_t &r) {
124     r = a;
125     if (a.raw_less(b)) {
126         return set(0);
127     }
128     v.clear();
129     v.resize(a.size() - b.size() + 1);
130     r.v.resize(a.size() + 1);
131     size_t offset = b.size();
132     double db = b.v.back();
133     if (b.size() > 2)
134         db += (b.v[b.size() - 2] + (b.v[b.size() - 3] + 1) /
135 (double)COMPRESS_MOD) / COMPRESS_MOD;
136     else if (b.size() > 1)
137         db += b.v[b.size() - 2] / (double)COMPRESS_MOD;
138     db = 1 / db;
139     for (size_t i = a.size() - offset; i <= a.size(); i++) {
140         carry_t rm = (carry_t)r.v[i + offset] * COMPRESS_MOD + r.v[i +
141 offset - 1], m;
142         m = std::max((carry_t)(rm * db), (carry_t)r.v[i + offset]);
143         if (m) {
144             v[i] += (base_t)m;
145             carry_t add = 0;
146             for (size_t j = 0; j < b.size(); j++)
147                 borrow(add, r.v[i + j], (carry_t)r.v[i + j] -
148 (carry_t)b.v[j] * m);
149             for (size_t j = i + b.size(); add && j < r.size(); ++j)
150                 borrow(add, r.v[j], (carry_t)r.v[j]);
151             i -= !r.v[i + offset];
152         }
153         r.trim();
154         carry_t add = 0;
155         while (!r.raw_less(b)) {
156             r.raw_sub(b);
157             ++add;
158         }
159         for (size_t i = 0; i < v.size(); i++)
160             carry(add, v[i], (carry_t)v[i]);
161         trim();

```

```

161         return *this;
162     }
163     BigInt_t &raw_shr(size_t n) {
164         if (n == 0) return *this;
165         if (n >= size()) {
166             set(0);
167             return *this;
168         }
169         v.erase(v.begin(), v.begin() + n);
170         return *this;
171     }
172     BigInt_t raw_shr_to(size_t n) const {
173         BigInt_t r;
174         if (n >= size()) return r;
175         r.v.assign(v.begin() + n, v.end());
176         return BIGINT_STD_MOVE(r);
177     }
178     BigInt_t &raw_shl(size_t n) {
179         if (n == 0 || is_zero()) return *this;
180         v.insert(v.begin(), n, 0);
181         return *this;
182     }
183     BigInt_t &raw_dividediv_recursion(const BigInt_t &a, const BigInt_t &b,
    BigInt_t &r) {
184         if (a < b) {
185             r = a;
186             return set(0);
187         } else if (b.size() <= BIGINT_DIVIDEDIV_THRESHOLD) {
188             return raw_div(a, b, r);
189         }
190         size_t base = (b.size() + 1) / 2;
191         if (a.size() <= base * 3) {
192             base = b.size() / 2;
193             BigInt_t ma = a, mb = b, e;
194             BigInt_t ha = ma.raw_shr_to(base);
195             BigInt_t hb = mb.raw_shr_to(base);
196             raw_dividediv_recursion(ha, hb, r);
197             ha = *this * b;
198             while (a < ha) {
199                 ha.raw_sub(b);
200                 raw_sub(BigInt_t(1));
201             }
202             r = a - ha;
203             return *this;
204         }
205         if (a.size() > base * 4) base = a.size() / 2;
206         BigInt_t ha = a.raw_shr_to(base);
207         BigInt_t c, d, m;
208         raw_dividediv_recursion(ha, b, d);
209         raw_shl(base);
210         m.v.resize(base + d.size());
211         for (size_t i = 0; i < base; ++i)
212             m.v[i] = a.v[i];
213         for (size_t i = 0; i < d.size(); ++i)
214             m.v[base + i] = d.v[i];
215         c.raw_dividediv_recursion(m, b, r);
216         raw_add(c);

```

```

217         return *this;
218     }
219     BigInt_t &raw_dividediv(const BigInt_t &a, const BigInt_t &b, BigInt_t
220     &r) {
221         if (b.size() <= BIGINT_DIVIDEDIV_THRESHOLD) {
222             raw_div(a, b, r);
223             return *this;
224         }
225         if (b.size() * 2 - 2 > a.size()) {
226             BigInt_t ta = a, tb = b;
227             size_t ans_len = a.size() - b.size() + 2;
228             size_t shr = b.size() - ans_len;
229             ta.raw_shr(shr);
230             tb.raw_shr(shr);
231             return raw_dividediv(ta, tb, r);
232         }
233         carry_t mul = (carry_t)((uint64_t)COMPRESS_MOD * COMPRESS_MOD -
234         1) / //
235             (*b.v.begin() + b.v.size() - 1) *
236             (uint64_t)COMPRESS_MOD + //
237             (*b.v.begin() + b.v.size() - 2) + 1));
238         BigInt_t ma = a * BigInt_t(mul), mb = b * BigInt_t(mul);
239         while (mb.v.back() < COMPRESS_MOD >> 1) {
240             int32_t m = 2;
241             ma.raw_mul(ma, BigInt_t(m));
242             mb.raw_mul(mb, BigInt_t(m));
243             mul *= m;
244         }
245         BigInt_t d;
246         ma.sign = mb.sign = 1;
247         raw_dividediv_recursion(ma, mb, d);
248         r.raw_div(d, BigInt_t((int)mul), ma);
249         return *this;
250     }
251     void trim() {
252         while (v.back() == 0 && v.size() > 1)
253             v.pop_back();
254     }
255     size_t size() const { return v.size(); }
256     BigInt_t &from_str_base10(const char *s) {
257         v.clear();
258         int32_t base = 10, sign = 1, digits = COMPRESS_DIGITS;
259         const char *p = s + strlen(s) - 1;
260         while (*s == '-')
261             sign *= -1, ++s;
262         while (*s == '0')
263             ++s;
264
265         int32_t d = digits, hdigit = 0, hdigit_mul = 1;
266         for (; p >= s; p--) {
267             hdigit += (*p - '0') * hdigit_mul;
268             hdigit_mul *= base;
269             if (--d == 0) {
270                 v.push_back(hdigit);
271                 d = digits;
272                 hdigit = 0;
273                 hdigit_mul = 1;

```



```

271     }
272 }
273 if (hdigit || v.empty()) v.push_back(hdigit);
274 this->sign = sign;
275 return *this;
276 }
277
278 public:
279 BigIntMini() { set(0); }
280 explicit BigIntMini(int n) { set(n); }
281 explicit BigIntMini(intmax_t n) { set(n); }
282 explicit BigIntMini(const char *s) { from_str(s); }
283 BigInt_t &set(intmax_t n) {
284     v.resize(1);
285     v[0] = 0;
286     uintmax_t s;
287     if (n < 0) {
288         sign = -1;
289         s = -n;
290     } else {
291         sign = 1;
292         s = n;
293     }
294     for (int i = 0; s; i++) {
295         v.resize(i + 1);
296         v[i] = low_digit(s);
297         s = high_digit(s);
298     }
299     return *this;
300 }
301 BigInt_t &from_str(const char *s) { return from_str_base10(s); }
302 bool is_zero() const { return v.size() == 1 && v[0] == 0; }
303 bool operator<(const BigInt_t &b) const {
304     if (sign * b.sign > 0) {
305         if (sign > 0)
306             return raw_less(b);
307         else
308             return b.raw_less(*this);
309     } else {
310         if (sign > 0)
311             return false;
312         else
313             return true;
314     }
315 }
316 bool operator==(const BigInt_t &b) const {
317     if (is_zero() && b.is_zero()) return true;
318     if (sign != b.sign) return false;
319     return raw_eq(b);
320 }
321 LESS_THAN_AND_EQUAL_COMPARABLE(BigInt_t)
322
323 BigInt_t &operator=(intmax_t n) { return set(n); }
324 BigInt_t &operator=(const char *s) { return from_str(s); }
325 BigInt_t operator+(const BigInt_t &b) const {
326     BigInt_t r = *this;
327     if (sign * b.sign > 0)
328         r.raw_add(b);

```

```

329         else
330             r.raw_sub(b);
331         return BIGINT_STD_MOVE(r);
332     }
333     BigInt_t operator-(const BigInt_t &b) const {
334         BigInt_t r = *this;
335         if (sign * b.sign < 0)
336             r.raw_add(b);
337         else
338             r.raw_sub(b);
339         return BIGINT_STD_MOVE(r);
340     }
341     BigInt_t operator-() const {
342         BigInt_t r = *this;
343         r.sign = -r.sign;
344         return BIGINT_STD_MOVE(r);
345     }
346     BigInt_t operator*(const BigInt_t &b) const {
347         BigInt_t r;
348         r.raw_mul_karatsuba(*this, b);
349         r.sign = sign * b.sign;
350         return BIGINT_STD_MOVE(r);
351     }
352     BigInt_t operator/(const BigInt_t &b) const {
353         BigInt_t r, d;
354         d.raw_dividediv(*this, b, r);
355         d.sign = sign * b.sign;
356         return BIGINT_STD_MOVE(d);
357     }
358     BigInt_t operator%(const BigInt_t &b) const { return
359     BIGINT_STD_MOVE(*this - *this / b * b); }
360     BigInt_t div(const BigInt_t &b, BigInt_t &r) {
361         if (this == &b) {
362             r.set(0);
363             return set(1);
364         }
365         BigInt_t d;
366         d.raw_dividediv(*this, b, r);
367         d.sign = sign * b.sign;
368         return BIGINT_STD_MOVE(d);
369     }
370     std::string out_dec() const {
371         if (is_zero()) return "0";
372         std::string out;
373         int32_t d = 0;
374         for (size_t i = 0, j = 0;;) {
375             if (j < 1) {
376                 if (i < size())
377                     d += v[i];
378                 else if (d == 0)
379                     break;
380                 j += 4;
381                 ++i;
382             }
383             out.push_back((d % 10) + '0');
384             d /= 10;
385             j -= 1;

```

```

386     }
387     while (out.size() > 1 && *out.rbegin() == '0')
388         out.erase(out.begin() + out.size() - 1);
389     if (sign < 0 && !this->is_zero()) out.push_back('-');
390     std::reverse(out.begin(), out.end());
391     return out;
392 }
393
394     std::string to_str() const { return out_dec(); }
395 };
396 } // namespace BigIntMiniNS
397
398 using BigIntMiniNS::BigIntMini;

```

1.2 bitIntPY.py

```

1 from decimal import *
2 import sys
3 setcontext(Context(prec=2000000, Emax=2000000, Emin=0))
4 a = sys.stdin.readline()
5 b = sys.stdin.readline()
6 print(Decimal(a) * Decimal(b))

```

1.3 `bitset.h`

```

1  #include<bitset>
2  int n;
3  auto bin = bitset<32>(n);
4
5  cout << bin;
6  //输出二进制位
7
8  cout << bin.to_ulong();
9  //输出十进制
10
11 bin[1] = 1
12 //随机访问
13
14 bin = !bin ^ (bin & bin | bitset<32>(1))
15 //位运算
16
17 bin != bin
18 //比较运算符
19
20 bin.count()
21 //1 的数量
22
23 bin.test(i)
24 //随机访问, 类似 std::vector::pos()
25
26 bin.any()
27 //有一位 1 就 true
28
29 bin.none()
30 //全 0 就返回 true
31
32 bin.all()
33 //全 1 就返回 true
34
35 bin.flip()
36 //翻转全部
37
38 bin.flip(i)
39 //a[i] = !a[i]
40
41 bin._Find_first()
42 //第一个 1 的下标
43
44 bin._Find_next(i)
45 //从下标 n 往后第一个 1 的下标

```

1.4 pbds.h

```

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

```

1.5 时间戳优化.h

```

1 //时间戳优化：对付多组数据很常见的技巧。
2 int tag[N], t[N], Tag;
3 int lowbit(int x){
4     return x&-x;
5 }
6 void reset(){
7     ++Tag;
8 }
9 void add(int x,int val){
10     while(x<=n){
11         if(tag[x]!=Tag) t[x]=0;
12         t[x]+=val;tag[x]=Tag;
13         x+=lowbit(x);
14     }
15 }
16 int getsum(int x){
17     int ans=0;
18     while(x){
19         if(tag[x]==Tag) ans+=t[x];
20         x-=lowbit(x);
21     }
22     return ans;
23 }

```

2 Ds

2.1 dsu.h

```

1  class DSU {
2      std::vector<int> f, siz;
3  public:
4      DSU() {}
5      DSU(int n) {
6          init(n);
7      }
8
9      void init(int n) {
10         f.resize(n);
11         for(int i = 0; i < n; i++) f[i] = i;
12         siz.assign(n, 1);
13     }
14
15     int find(int x) {
16         while (x != f[x]) {
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y) {
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y) {
27         x = find(x);
28         y = find(y);
29         if (x == y) {
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36
37     int size(int x) {
38         return siz[find(x)];
39     }
40 };

```


2.2 dsu_classification.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e5+100;
4  #define ll long long
5  #define int long long
6
7  struct node{
8      int x,y,z;
9  }s[N];
10 vector<int> a(N),b(N);
11 int find(int x){
12     return (a[x]==x) ? x : a[x]=find(a[x]);
13 }
14 void merge(int x,int y){
15     a[find(x)]=find(y);
16 }
17 void solve(){
18     int n,m;
19     cin>>n>>m;
20     vector<int> v(n+1);
21     for(int i=0;i<=n;i++) a[i]=i;
22     for(int i=0;i<m;i++) cin>>s[i].x>>s[i].y>>s[i].z;
23     sort(s,s+m,[&](node a,node b){
24         return a.z>b.z;
25     });
26     for(int i=0;i<m;i++){
27         if(find(s[i].x)==find(s[i].y)){
28             cout<<s[i].z;
29             return;
30         }
31         if(!b[s[i].x]) b[s[i].x]=s[i].y;
32         else merge(s[i].y,b[s[i].x]);
33         if(!b[s[i].y]) b[s[i].y]=s[i].x;
34         else merge(s[i].x,b[s[i].y]);
35     }
36     cout<<0;
37     return;
38 }
39
40
41 signed main(){
42     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
43     int t=1;
44     // cin>>t;
45     while(t-->0) solve();
46     return 0;
47 }

```

2.3 dsu_weighted.h

```

1  #include <cassert>
2  #include <iostream>
3  #include <numeric>
4  #include <vector>
5
6  using namespace std;
7
8  struct dsu {
9      vector<size_t> pa, size, sum;
10
11      explicit dsu(size_t size_)
12          : pa(size_ * 2), size(size_ * 2, 1), sum(size_ * 2) {
13          // size 与 sum 的前半段其实没有使用, 只是为了让下标计算更简单
14          iota(pa.begin(), pa.begin() + size_, size_);
15          iota(pa.begin() + size_, pa.end(), size_);
16          iota(sum.begin() + size_, sum.end(), 0);
17      }
18
19      void unite(size_t x, size_t y) {
20          x = find(x), y = find(y);
21          if (x == y) return;
22          if (size[x] < size[y]) swap(x, y);
23          pa[y] = x;
24          size[x] += size[y];
25          sum[x] += sum[y];
26      }
27
28      void move(size_t x, size_t y) {
29          auto fx = find(x), fy = find(y);
30          if (fx == fy) return;
31          pa[x] = fy;
32          --size[fx], ++size[fy];
33          sum[fx] -= x, sum[fy] += x;
34      }
35
36      size_t find(size_t x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }
37 };
38
39 int main() {
40     size_t n, m, op, x, y;
41     while (cin >> n >> m) {
42         dsu dsu(n + 1); // 元素范围是 1..n
43         while (m--) {
44             cin >> op;
45             switch (op) {
46                 case 1:
47                     cin >> x >> y;
48                     dsu.unite(x, y);
49                     break;
50                 case 2:
51                     cin >> x >> y;
52                     dsu.move(x, y);
53                     break;
54                 case 3:
55                     cin >> x;
56                     x = dsu.find(x);
57                     cout << dsu.size[x] << ' ' << dsu.sum[x] << '\n';

```

```
58         break;
59     default:
60         assert(false); // not reachable
61     }
62 }
63 }
64 }
```

2.4 segTree_add.h

```

1 // AC 带懒惰标记线段树
2 template <class TYPE_NAME>
3 class lazyTree
4 {
5 private:
6     vector<TYPE_NAME> d;
7     vector<TYPE_NAME> a;
8     vector<TYPE_NAME> b;
9     int n;
10    const TYPE_NAME INI = 0; // 不会影响合并运算的初始值, 如 max 取 INF, min 取 0,
    mti 取 1
11
12    void subbuild(int s, int t, int p)
13    {
14        if (s == t)
15        {
16            d[p] = a[s];
17            return;
18        }
19        int m = s + ((t - s) >> 1); // (s+t)/2
20        subbuild(s, m, p * 2);
21        subbuild(m + 1, t, p * 2 + 1);
22        d[p] = d[p * 2] + d[p * 2 + 1];
23        // 合并运算符 ↑
24    }
25
26    TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
27    {
28        if (l <= s && t <= r)
29            return d[p];
30        int m = s + ((t - s) >> 1);
31        if (b[p] != 0)
32        {
33            d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应
    用懒惰标记
34            d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应
    用懒惰标记
35            b[p * 2] += b[p]; // 下传标记, 无需修改
36            b[p * 2 + 1] += b[p]; // 下传标记, 无需修改
37            b[p] = 0;
38        }
39        TYPE_NAME ans1 = INI;
40        TYPE_NAME ansr = INI;
41        if (l <= m)
42            ans1 = subGetSum(l, r, s, m, p * 2);
43        if (r > m)
44            ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);
45        return ans1 + ansr;
46        // 合并运算符↑
47    }
48
49    void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
50    {
51        if (l <= s && t <= r)
52    
```

```

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

```

2.5 segTree_add_setto.h

```

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

```

```

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

```

2.6 segTree_mul_add.h

```

1  /*
2  三种操作：(模数为 mod)
3  1.区间乘 val
4  2.区间加 val
5  3.区间和
6  */
7  #define int long long
8  #define lc p<<1
9  #define rc p<<1|1
10 const int N=1e5+10;
11 int mod=1e9+7;
12 struct node{
13     int l,r,sum,mul,add;
14 }tr[4*N];
15 vector<int> a(N);
16 void pushup(int p){
17     tr[p].sum=tr[lc].sum+tr[rc].sum;
18     tr[p].sum%=mod;
19 }
20 void build(int p,int l,int r){
21     tr[p].l=l;tr[p].r=r;
22     tr[p].mul=1;tr[p].add=0;
23     if(l==r){
24         tr[p].sum=a[l];
25         return;
26     }
27     int m=l+r>>1;
28     build(lc,l,m);
29     build(rc,m+1,r);
30     pushup(p);
31 }
32 void pushdown(int p){
33     tr[lc].sum=(tr[lc].sum*tr[p].mul+tr[p].add*(tr[lc].r-tr[lc].l+1))%mod;
34     tr[rc].sum=(tr[rc].sum*tr[p].mul+tr[p].add*(tr[rc].r-tr[rc].l+1))%mod;
35     tr[lc].mul=tr[lc].mul*tr[p].mul%mod;
36     tr[rc].mul=tr[rc].mul*tr[p].mul%mod;
37     tr[lc].add=(tr[lc].add*tr[p].mul+tr[p].add)%mod;
38     tr[rc].add=(tr[rc].add*tr[p].mul+tr[p].add)%mod;
39     tr[p].mul=1;tr[p].add=0;
40 }
41 void update_mul(int p,int l,int r,int val){
42     if(l<=tr[p].l&&tr[p].r<=r){
43         tr[p].sum=tr[p].sum*val%mod;
44         tr[p].mul=tr[p].mul*val%mod;
45         tr[p].add=tr[p].add*val%mod;
46         return;
47     }
48     pushdown(p);
49     int m=tr[p].l+tr[p].r>>1;
50     if(l<=m) update_mul(lc,l,r,val);
51     if(m<r) update_mul(rc,l,r,val);
52     pushup(p);
53     return;
54 }
55 void update_add(int p,int l,int r,int val){

```



```

56     if(l<=tr[p].l&&tr[p].r<=r){
57         tr[p].add=(tr[p].add+val)%mod;
58         tr[p].sum=(tr[p].sum+val*(tr[p].r-tr[p].l+1))%mod;
59         return;
60     }
61     pushdown(p);
62     int m=tr[p].l+tr[p].r>>1;
63     if(l<=m) update_add(lc,l,r,val);
64     if(m<r) update_add(rc,l,r,val);
65     pushup(p);
66     return;
67 }
68 int query(int p,int l,int r){
69     if(l<=tr[p].l&&tr[p].r<=r) return tr[p].sum;
70     pushdown(p);
71     int m=tr[p].l+tr[p].r>>1;
72     int sum=0;
73     if(l<=m) sum+=query(lc,l,r);
74     if(m<r) sum+=query(rc,l,r);
75     return sum%mod;
76 }
77 void solve(){
78     int n,m;
79     cin>>n>>m>>mod;
80     for(int i=1;i<=n;i++) cin>>a[i];
81     build(1,1,n);
82     while(m--){
83         int op;
84         cin>>op;
85         if(op==1){
86             int x,y,val;
87             cin>>x>>y>>val;
88             update_mul(1,x,y,val);
89         }
90         else if(op==2){
91             int x,y,val;
92             cin>>x>>y>>val;
93             update_add(1,x,y,val);
94         }
95         else{
96             int x,y;
97             cin>>x>>y;
98             cout<<query(1,x,y)<<"\n";
99         }
100     }
101     return;
102 }

```

2.7 segTree_MX_MI.h

```

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

```

```

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

```

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

2.8 segTree_历史最值.h

```

1  #include <algorithm>
2  #include <climits>
3  #include <iostream>
4  using namespace std;
5  using ll = long long;
6
7  constexpr int N = 1e5 + 7;
8
9  struct Tree {
10     int mx, _mx; // 区间最大值 区间历史最大值
11     int ad, _ad; // 区间加标记 区间阶段历史最大加标记
12     int st, _st; // 区间修改值 区间阶段历史最大修改标记
13 } g[N * 4];
14
15 int a[N];
16 int n, m;
17 #define ls u * 2
18 #define rs u * 2 + 1
19 #define mid (l + r) / 2
20
21 void pushup(int u) {
22     g[u].mx = max(g[ls].mx, g[rs].mx);
23     g[u]._mx = max(g[ls]._mx, g[rs]._mx);
24 }
25
26 void pushadd(int u, int v, int _v) {
27     g[u]._mx = max(g[u]._mx, g[u].mx + _v), g[u].mx += v;
28     if (g[u].st == INT_MIN)
29         g[u]._ad = max(g[u]._ad, g[u].ad + _v), g[u].ad += v;
30     else
31         g[u]._st = max(g[u]._st, g[u].st + _v), g[u].st += v;
32 }
33
34 void pushset(int u, int v, int _v) {
35     g[u]._mx = max(g[u]._mx, _v), g[u].mx = v;
36     g[u]._st = max(g[u]._st, _v), g[u].st = v;
37 }
38
39 void pushdown(int u, int l, int r) {
40     if (g[u].ad || g[u]._ad)
41         pushadd(ls, g[u].ad, g[u]._ad), pushadd(rs, g[u].ad, g[u]._ad),
42         g[u].ad = g[u]._ad = 0;
43     if (g[u].st != INT_MIN || g[u]._st != INT_MIN)
44         pushset(ls, g[u].st, g[u]._st), pushset(rs, g[u].st, g[u]._st),
45         g[u].st = g[u]._st = INT_MIN;
46 }
47
48 void build(int u = 1, int l = 1, int r = n) {
49     g[u]._st = g[u].st = INT_MIN;
50     if (l == r) {
51         g[u].mx = a[l];
52         g[u]._mx = a[l];
53         return;
54     }
55     build(ls, l, mid), build(rs, mid + 1, r);
56     pushup(u);
57 }

```

```

58
59 int L, R;
60
61 void add(int v, int u = 1, int l = 1, int r = n) {
62     if (R < l || r < L) return;
63     if (L <= l && r <= R) return pushadd(u, v, max(v, 0));
64     pushdown(u, l, r);
65     add(v, ls, l, mid), add(v, rs, mid + 1, r);
66     pushup(u);
67 }
68
69 void tset(int v, int u = 1, int l = 1, int r = n) {
70     if (R < l || r < L) return;
71     if (L <= l && r <= R) return pushset(u, v, v);
72     pushdown(u, l, r);
73     tset(v, ls, l, mid), tset(v, rs, mid + 1, r);
74     pushup(u);
75 }
76
77 int qmax(int u = 1, int l = 1, int r = n) {
78     if (R < l || r < L) return INT_MIN;
79     if (L <= l && r <= R) return g[u].mx;
80     pushdown(u, l, r);
81     return max(qmax(ls, l, mid), qmax(rs, mid + 1, r));
82 }
83
84 int qmaxh(int u = 1, int l = 1, int r = n) {
85     if (R < l || r < L) return INT_MIN;
86     if (L <= l && r <= R) return g[u]._mx;
87     pushdown(u, l, r);
88     return max(qmaxh(ls, l, mid), qmaxh(rs, mid + 1, r));
89 }
90
91 int main() {
92     cin.tie(nullptr)->sync_with_stdio(false);
93     cin >> n;
94     for (int i = 1; i <= n; ++i) cin >> a[i];
95     build();
96     int m, z;
97     cin >> m;
98     for (int i = 1; i <= m; ++i) {
99         char op;
100         cin >> op;
101         while (op == ' ' || op == '\r' || op == '\n') cin >> op;
102         cin >> L >> R;
103         int x;
104         if (op == 'Q')
105             cout << qmax() << '\n';
106         else if (op == 'A')
107             cout << qmaxh() << '\n';
108         else if (op == 'P')
109             cin >> x, add(x);
110         else
111             cin >> x, tset(x);
112     }
113     return 0;
114 }

```

2.9 SparseTable.h

```

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

```

2.10 twoDimPrfxSum.h

```

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

```



```

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

```

2.11 主席树.h

```

1  #define int long long
2
3  const int N=2e5+10;
4  #define lc(x) tr[x].l
5  #define rc(x) tr[x].r
6  struct node{
7      int l,r,s; //l,r->son s->cnt
8  }tr[N*20];
9  int root[N],idx;
10 int n,m,a[N];
11 vector<int> b;
12 void build(int &x,int l,int r){
13     x=++idx;
14     if(l==r) return;
15     int m=l+r>>1;
16     build(lc(x),l,m);
17     build(rc(x),m+1,r);
18 }
19 void insert(int x,int &y,int l,int r,int k){
20     y=++idx; tr[y]=tr[x]; tr[y].s++;
21     if(l==r) return;
22     int m=l+r>>1;
23     if(k<=m) insert(lc(x),lc(y),l,m,k);
24     else insert(rc(x),rc(y),m+1,r,k);
25 }
26 int query(int x,int y,int l,int r,int k){
27     if(l==r) return l;
28     int m=l+r>>1;
29     int s=tr[lc(y)].s-tr[lc(x)].s;
30     if(k<=s) return query(lc(x),lc(y),l,m,k);
31     else return query(rc(x),rc(y),m+1,r,k-s);
32 }
33
34 void solve(){
35     int n,m;
36     cin>>n>>m;
37     for(int i=1;i<=n;i++){
38         cin>>a[i];
39         b.push_back(a[i]);
40     }
41     sort(b.begin(),b.end());
42     b.erase(unique(b.begin(),b.end()),b.end());
43     int bn=b.size();
44     build(root[0],1,bn);
45     for(int i=1;i<=n;i++){
46         int id=lower_bound(b.begin(),b.end(),a[i])-b.begin()+1;
47         insert(root[i-1],root[i],1,bn,id);
48     }
49     while(m--){
50         int l,r,k;
51         cin>>l>>r>>k;
52         int id=query(root[l-1],root[r],1,bn,k)-1;
53         cout<<b[id]<<"\n";
54     }
55 }

```

2.12 单调队列.h

```

1  void solve(){
2      int n,k;
3      cin>>n>>k;
4      vector<int> a(n+1),ans(n+1);
5      deque<int> q;
6      for(int i=1;i<=n;i++){
7          cin>>a[i];
8          while(!q.empty()&&(q.front()<=i-k)) q.pop_front();
9          while(!q.empty()&&a[q.back()]<=a[i]) q.pop_back();
10         q.push_back(i);
11         ans[i]=a[q.front()];
12     }
13 }
```

2.13 左偏树.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define N 100005
5  const ll inf=1e17+7;
6  signed main()
7  {
8      ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
9      void solve();
10     int t=1;
11     while(t--)solve();
12     return 0;
13 }
14 vector<int> a;
15 int fa[N];
16 int find(int u)
17 {
18     return (fa[u]==u)?u:fa[u]=find(fa[u]);
19 }
20 struct left_tree
21 {
22     int rs,ls,fa;
23     ll val;
24     int d=-1;
25 }tree[N];
26 void init(int x,ll v)
27 {
28     tree[x].val=v;
29     tree[x].d=tree[0].d+1;
30     tree[x].fa=tree[x].ls=tree[x].rs=0;
31 }
32 int merge(int x,int y)
33 {
34     if(!x||!y)return x|y;//存在空堆
35     if((tree[x].val==tree[y].val)?x>y:tree[x].val>tree[y].val)swap(x,y); //
    维护小根堆，选择权小的作为根 x
36     tree[x].rs=merge(tree[x].rs,y); //合并根的右儿子和 y
37     if(tree[tree[x].rs].d>tree[tree[x].ls].d)swap(tree[x].rs,tree[x].ls);//维
    护左偏性
38     tree[x].d=tree[tree[x].rs].d+1; //距离增加
39     tree[tree[x].rs].fa=x; //更新父子关系
40     return x;
41 }
42 void pushup(int x)
43 {
44     if(!x)return; //无可更新
45     if(tree[tree[x].rs].d>tree[tree[x].ls].d)swap(tree[x].rs,tree[x].ls);//维
    护左偏性
46     if(tree[x].d!=tree[tree[x].rs].d+1) //维护距离
47     {
48         tree[x].d=tree[tree[x].rs].d+1;
49         pushup(tree[x].fa); //向上更新
50     }
51 }
```

```

52 void erase(int x)
53 {
54     int y=merge(tree[x].rs,tree[x].ls); //合并 x 结点的左右儿子
55     tree[y].fa=tree[x].fa; //更新新子树的根的父亲
56
57     if(tree[tree[x].fa].ls==x)tree[tree[x].fa].ls=y; //维护父子关系
58     else if(tree[tree[x].fa].rs==x)tree[tree[x].fa].rs=y;
59
60     pushup(tree[y].fa); //向上维护左偏性
61 }
62 void uni(int x,int y)//合并 x 和 y 所在的堆
63 {
64     if(a[x]==-1||a[y]==-1)return ;
65     int fax=find(x),fay=find(y);
66     if(fax==fay)return;
67     fa[fax]=fa[fay]=merge(fax,fay);
68 }
69 int pop(int x)
70 {
71     if(a[x]==-1)return -1;
72     int fa1=find(x);
73     int ans=a[fa1];
74     a[fa1]=-1;
75
76     fa[tree[fa1].ls]=fa[tree[fa1].rs]=fa[fa1]=merge(tree[fa1].ls,tree[fa1].rs);
77     tree[fa[tree[fa1].ls]].fa=0;
78     return ans;
79 }
80 void solve()
81 {
82     int n,m;
83     cin>>n>>m;
84     a.resize(n+10);
85     tree[0].d=-1;
86     for(int i=1;i<=n;i++)
87     {
88         cin>>a[i];
89         init(i,a[i]);
90     }
91     for(int i=1;i<=n;i++)
92     {
93         fa[i]=i;
94     }
95     while(m--)
96     {
97         int op;
98         cin>>op;
99         if(op==1)
100         {
101             int x,y;
102             cin>>x>>y;
103             uni(x,y);
104         }
105         else if(op==2)
106         {
107             int x;
108             cin>>x;

```

```
108         int ans=pop(x);  
109         cout<<ans<<"\n";  
110     }  
111 }  
112 }
```

2.14 平衡树.h

```

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

```

```

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

```


2.15 树状数组.h

```

1 //区间修改, 区间查询 如果 N=4000, 内存大概在 300 多 MB 注意空间
2 #define int long long
3 const int N=2050;
4 int t1[N][N],t2[N][N],t3[N][N],t4[N][N];
5 int n,m;
6 int lowbit(int x){
7     return x&-x;
8 }
9 void add(int x,int y,int val){
10     for(int i=x;i<=n;i+=lowbit(i)){
11         for(int j=y;j<=m;j+=lowbit(j)){
12             t1[i][j]+=val;
13             t2[i][j]+=val*x;
14             t3[i][j]+=val*y;
15             t4[i][j]+=val*x*y;
16         }
17     }
18 }
19 void range_add(int xa,int ya,int xb,int yb,int val){
20     add(xa,ya,val); add(xa,yb+1,-val);
21     add(xb+1,ya,-val); add(xb+1,yb+1,val);
22 }
23 int ask(int x,int y){
24     int ans=0;
25     for(int i=x;i>=1;i-=lowbit(i)){
26         for(int j=y;j>=1;j-=lowbit(j)){
27             ans+=(x+1)*(y+1)*t1[i][j]-(y+1)*t2[i][j]-(x+1)*t3[i][j]+t4[i][j];
28         }
29     }
30     return ans;
31 }
32 int range_ask(int xa,int ya,int xb,int yb){
33     return ask(xb,yb)-ask(xb,ya-1)-ask(xa-1,yb)+ask(xa-1,ya-1);
34 }
35 void solve(){
36     cin>>n>>m;
37     int op;
38     while(cin>>op){
39         if(op==1){
40             int a,b,c,d,val;
41             cin>>a>>b>>c>>d>>val;
42             range_add(a,b,c,d,val);
43         }
44         else{
45             int a,b,c,d;
46             cin>>a>>b>>c>>d;
47             cout<<range_ask(a,b,c,d)<<"\n";
48         }
49     }
50 }
51
52
53 // 点修 区间查询 可以建一个数组, 可以开更大的 N, 防止 MLE
54 #define int long long
55 const int N=4100;
56 int t[N][N];

```

```

57 int n,m;
58 int lowbit(int x){
59     return x&-x;
60 }
61 void add(int x,int y,int val){
62     for(int i=x;i<=n;i+=lowbit(i)){
63         for(int j=y;j<=m;j+=lowbit(j)){
64             t[i][j]+=val;
65         }
66     }
67 }
68 int sum(int x,int y){
69     int ans=0;
70     for(int i=x;i>=1;i-=lowbit(i)){
71         for(int j=y;j>=1;j-=lowbit(j)){
72             ans+=t[i][j];
73         }
74     }
75     return ans;
76 }
77 int ask(int x1,int y1,int x2,int y2){
78     return sum(x2,y2)-sum(x2,y1-1)-sum(x1-1,y2)+sum(x1-1,y1-1);
79 }
80 void solve(){
81     cin>>n>>m;
82     int op;
83     while(cin>>op){
84         if(op==1){
85             int x,y,val;
86             cin>>x>>y>>val;
87             add(x,y,val);
88         }
89         else{
90             int a,b,c,d;
91             cin>>a>>b>>c>>d;
92             cout<<ask(a,b,c,d)<<"\n";
93         }
94     }
95 }
96
97 // 第 k 大
98 int t[N];
99 int kth(int k){
100     int ans=0,x=0;
101     for(int i=log2(n);~i;i--){
102         x+=(1<<i);
103         if(x>=n||sum+t[x]>=k) x-=(1<<i);
104         else sum+=t[x];
105     }
106     return x+1;
107 }
108
109 // 区间修改 单点查询
110 const int N=4100;
111 int t[N][N];
112 int n,m;
113 int lowbit(int x){
114     return x&-x;

```

```

115 }
116 void add(int x,int y,int val){
117     for(int i=x;i<=n;i+=lowbit(i)){
118         for(int j=y;j<=m;j+=lowbit(j)){
119             t[i][j]+=val;
120         }
121     }
122 }
123 void range_add(int x1,int y1,int x2,int y2,int val){
124     add(x1,y1,val); add(x1,y2+1,-val);
125     add(x2+1,y1,-val); add(x2+1,y2+1,val);
126 }
127 int sum(int x,int y){
128     int ans=0;
129     for(int i=x;i;i-=lowbit(i)){
130         for(int j=y;j;j-=lowbit(j)){
131             ans+=t[i][j];
132         }
133     }
134     return ans;
135 }

```

3 GEO

3.1 Rotating_Calipers.h

```

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

```

```

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

```

3.2 三维封装.h

```

1  using ld = long double;
2
3  struct Point3
4  {
5      ld x, y, z;
6      Point3(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
7      Point3 &operator+=(Point3 p) &
8      {
9          return x += p.x, y += p.y, z += p.z, *this;
10     }
11     Point3 &operator-=(Point3 p) &
12     {
13         return x -= p.x, y -= p.y, z -= p.z, *this;
14     }
15     Point3 &operator*=(Point3 p) &
16     {
17         return x *= p.x, y *= p.y, z *= p.z, *this;
18     }
19     Point3 &operator*=(ld t) &
20     {
21         return x *= t, y *= t, z *= t, *this;
22     }
23     Point3 &operator/=(ld t) &
24     {
25         return x /= t, y /= t, z /= t, *this;
26     }
27     friend Point3 operator+(Point3 a, Point3 b) { return a += b; }
28     friend Point3 operator-(Point3 a, Point3 b) { return a -= b; }
29     friend Point3 operator*(Point3 a, Point3 b) { return a *= b; }
30     friend Point3 operator*(Point3 a, ld b) { return a *= b; }
31     friend Point3 operator*(ld a, Point3 b) { return b *= a; }
32     friend Point3 operator/(Point3 a, ld b) { return a /= b; }
33     friend auto &operator>>(istream &is, Point3 &p)
34     {
35         return is >> p.x >> p.y >> p.z;
36     }
37     friend auto &operator<<(ostream &os, Point3 p)
38     {
39         return os << "(" << p.x << ", " << p.y << ", " << p.z << ")";
40     }
41 };
42 using P3 = Point3;
43 struct Line3
44 {
45     Point3 a, b;
46 };
47 using L3 = Line3;
48 struct Plane
49 {
50     Point3 u, v, w;
51 };
52
53 ld len(P3 p)
54 { // 原点到当前点的距离计算
55     return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
56 }

```

```

57 P3 crossEx(P3 a, P3 b)
58 { // 叉乘
59     P3 ans;
60     ans.x = a.y * b.z - a.z * b.y;
61     ans.y = a.z * b.x - a.x * b.z;
62     ans.z = a.x * b.y - a.y * b.x;
63     return ans;
64 }
65 ld cross(P3 a, P3 b)
66 {
67     return len(crossEx(a, b));
68 }
69 ld dot(P3 a, P3 b)
70 { // 点乘
71     return a.x * b.x + a.y * b.y + a.z * b.z;
72 }
73 P3 getVec(Plane s)
74 { // 获取平面法向量
75     return crossEx(s.u - s.v, s.v - s.w);
76 }
77 ld dis(P3 a, P3 b)
78 { // 三维欧几里得距离公式
79     ld val = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z -
80     b.z) * (a.z - b.z);
81     return sqrt(val);
82 }
83 P3 standardize(P3 vec)
84 { // 将三维向量转换为单位向量
85     return vec / len(vec);
86 }
87 bool onLine(P3 p1, P3 p2, P3 p3)
88 { // 三点是否共线
89     return sign(cross(p1 - p2, p3 - p2)) == 0;
90 }
91 bool onLine(Plane s)
92 {
93     return onLine(s.u, s.v, s.w);
94 }
95 bool onPlane(P3 p1, P3 p2, P3 p3, P3 p4)
96 { // 四点是否共面
97     ld val = dot(getVec({p1, p2, p3}), p4 - p1);
98     return sign(val) == 0;
99 }
100 bool pointOnSegment(P3 p, L3 l)
101 {
102     return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) <= p.x &&
103     p.x <= max(l.a.x, l.b.x) && min(l.a.y, l.b.y) <= p.y && p.y <=
104     max(l.a.y, l.b.y) &&
105     min(l.a.z, l.b.z) <= p.z && p.z <= max(l.a.z, l.b.z);
106 }
107 bool pointOnSegmentEx(P3 p, L3 l)
108 { // pointOnSegment 去除端点版
109     return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) < p.x &&
110     p.x < max(l.a.x, l.b.x) && min(l.a.y, l.b.y) < p.y && p.y <
111     max(l.a.y, l.b.y) &&
112     min(l.a.z, l.b.z) < p.z && p.z < max(l.a.z, l.b.z);

```

```

111 }
112 bool pointOnSegmentSide(P3 p1, P3 p2, L3 l)
113 {
114     if (!onPlane(p1, p2, l.a, l.b))
115     { // 特判不共面
116         return 0;
117     }
118     ld val = dot(crossEx(l.a - l.b, p1 - l.b), crossEx(l.a - l.b, p2 - l.b));
119     return sign(val) == 1;
120 }
121 bool pointOnPlaneSide(P3 p1, P3 p2, Plane s)
122 {
123     ld val = dot(getVec(s), p1 - s.u) * dot(getVec(s), p2 - s.u);
124     return sign(val) == 1;
125 }
126 bool lineParallel(L3 l1, L3 l2)
127 { // 平行
128     return sign(cross(l1.a - l1.b, l2.a - l2.b)) == 0;
129 }
130 bool lineVertical(L3 l1, L3 l2)
131 { // 垂直
132     return sign(dot(l1.a - l1.b, l2.a - l2.b)) == 0;
133 }

```


3.3 距离转化.typ

3.3.0.1 曼哈顿距离

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2| \quad (1)$$

3.3.0.2 欧几里得距离

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2)$$

3.3.0.3 切比雪夫距离

$$d(A, B) = \max(|x_1 - x_2|, |y_1 - y_2|) \quad (3)$$

3.3.0.4 闵可夫斯基距离

$$d(A, B) = (|x_1 - x_2|^p + |y_1 - y_2|^p)^{\{\frac{1}{p}\}} \quad (4)$$

3.3.0.5 曼哈顿转切比雪夫

对于直角坐标中的 $A(x_1, y_1), B(x_2, y_2)$

其曼哈顿距离

$$d(A, B) = \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|) \quad (5)$$

即为点 $A'(x_1 + y_1, x_1 - y_1), B'(x_2 + y_2, x_2 - y_2)$ 的切比雪夫距离。

同理，其切比雪夫距离

$$d(A, B) = \max\left(\left|\frac{x_1 + y_1}{2} - \frac{x_2 + y_2}{2}\right| + \left|\frac{x_1 - y_1}{2} - \frac{x_2 - y_2}{2}\right|\right) \quad (6)$$

即为点 $A'\left(\frac{x_1 + y_1}{2}, \frac{x_1 - y_1}{2}\right), B'\left(\frac{x_2 + y_2}{2}, \frac{x_2 - y_2}{2}\right)$ 的曼哈顿距离。

综上：

曼哈顿距离 \Rightarrow 切比雪夫距离：

$$(x, y) \Rightarrow (x + y, x - y)$$

切比雪夫距离 \Rightarrow 曼哈顿距离：

$$(x, y) \Rightarrow \left(\frac{x + y}{2}, \frac{x - y}{2}\right)$$

(7)

3.4 跨立实验.h

```

1  using pii = pair<int, int>
2
3  #define fi first
4  #define se second
5      const long double EPS = 1e-9;
6
7  template <class T>
8  int sign(T x)
9  {
10     if (-EPS < x && x < EPS)
11         return 0;
12     return x < 0 ? -1 : 1;
13 }
14
15 // 叉乘
16 template <class T>
17 T cross(pair<T, T> a, pair<T, T> b)
18 {
19     return a.fi * b.se - a.se * b.fi;
20 }
21
22 // 二维快速跨立实验
23 template <class T>
24 bool segIntersection(pair<T, T> l1, pair<T, T> l2)
25 {
26     auto [s1, e1] = l1;
27     auto [s2, e2] = l2;
28     auto A = max(s1.fi, e1.fi), AA = min(s1.fi, e1.fi);
29     auto B = max(s1.se, e1.se), BB = min(s1.se, e1.se);
30     auto C = max(s2.fi, e2.fi), CC = min(s2.fi, e2.fi);
31     auto D = max(s2.se, e2.se), DD = min(s2.se, e2.se);
32     return A >= CC && B >= DD && C >= AA && D >= BB &&
33         sign(cross(s1, s2, e1) * cross(s1, e1, e2)) == 1 &&
34         sign(cross(s2, s1, e2) * cross(s2, e2, e1)) == 1;
35 }
36
37 // 三维线段交点, 需要 P3 封装, 不相交返回{0, {}}
38 pair<bool, P3> lineIntersection(L3 l1, L3 l2)
39 {
40     if (!onPlane(l1.a, l1.b, l2.a, l2.b) || lineParallel(l1, l2))
41     {
42         return {0, {}};
43     }
44     auto [s1, e1] = l1;
45     auto [s2, e2] = l2;
46     ld val = 0;
47     if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 0, 1}))
48     {
49         val = ((s1.x - s2.x) * (s2.y - e2.y) - (s1.y - s2.y) * (s2.x -
50 e2.x)) /
51             ((s1.x - e1.x) * (s2.y - e2.y) - (s1.y - e1.y) * (s2.x - e2.x));
52     }
53     else if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 1, 0}))
54     {
55         val = ((s1.x - s2.x) * (s2.z - e2.z) - (s1.z - s2.z) * (s2.x -
56 e2.x)) /

```

```

55         ((s1.x - e1.x) * (s2.z - e2.z) - (s1.z - e1.z) * (s2.x - e2.x));
56     }
57     else
58     {
59         val = ((s1.y - s2.y) * (s2.z - e2.z) - (s1.z - s2.z) * (s2.y -
60 e2.y)) /
61         ((s1.y - e1.y) * (s2.z - e2.z) - (s1.z - e1.z) * (s2.y - e2.y));
62     }
63     return {1, s1 + (e1 - s1) * val};
64 }

```

4 GRAPH

4.1 2-SAT.h

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4
5  using pii = pair<int,int>;
6  class graph {
7  public:
8      vector<vector<pii>> cnj;
9      vector<int> dfn,scc,sz;
10     int sc;
11     int n;
12
13     graph() { }
14
15     graph(int _n) {
16         n = _n;
17         cnj.resize(n+1);
18     };
19
20     graph(vector<vector<int>>> _cnj){
21     }
22
23 private:
24     reset(auto &t){t.clear();t.resize(n+1);};
25 public:
26     void addOrdEdge(int u,int v,int w = 1) {
27
28     }
29
30     void addUnordEdge(int u,int v,int w = 1) {
31
32     }
33
34     //tarjin 缩点
35     void runSCC(){
36         reset(dfn),reset(scc),reset(sz);
37         vector<int> low(n+1),s,vst(n+1),sz(n+1);
38         sc = 0;
39         int tp = 0,dfncnt = 0;
40         using itf = function<void(int)>;
41         itf dfs = [&](int u) -> void {
42             low[u] = dfn[u] = ++dfncnt;
43             s.pb(u),vst[u] = 1;
44             for(auto [v,_]:cnj[u])
45             {
46                 if(!dfn[v])
47                 {
48                     dfs(v);
49                     low[u] = min(low[u],low[v]);
50                 } else if(vst[v]){
51                     low[u] = min(low[u],dfn[v]);
52                 }
53             }
54             if(dfn[u] == low[u]){
55                 sc ++;

```

```

56         while(s.back()!=u) {
57             scc[s.back()] = sc;
58             sz[sc]++;
59             vst[s.back()] = 0;
60             s.pop_back();
61         }
62         scc[s.back()] = sc;
63         sz[sc]++;
64         vst[s.back()] = 0;
65         s.pop_back();
66     }
67 };
68 for(int i = 1;i <= n;i ++)
69     if(!dfn[i]) dfs(i);
70 }
71
72 /**
73  * @param method:string
74  * Kruskal:O(m log m) for 稠密图
75  * Prim:O((n+m) log n) for 稀疏图
76  * @return vector<vector<int>> 生成树邻接表
77  */
78 vector<vector<pii>> runMST(string method = "Kruskal"){
79     if(method == "Kruskal") {
80
81     } else if(method == "Prim"){
82
83     }
84 }
85
86
87 vector<pii> findBridge(string method = "tarjin"){
88
89 }
90
91 vector<int> findCut(string method = "tarjin"){
92
93 }
94
95 void eraseEdge(vector<pii> lst) {
96
97 }
98
99 void eraseVertice(vector<int> lst) {
100
101 }
102 };
103
104 class two_SAT{
105 public:
106     graph g;
107     int n;
108     two_SAT(int _n) {
109         assert(_n%2 == 0);
110         g = graph(_n);
111         n = _n;
112     }
113
114     int tr(int p){
115         if(p%2) return p+1;

```

```

116     else return p-1;
117 }
118
119 void addDistinckPair(int u,int v){
120     g.addOrdEdge(u,tr(v));
121     g.addOrdEdge(v,tr(u));
122 }
123
124 /**
125  * @brief
126  * 图的 2-SAT: 选择一组节点, 对于所有选取的节点满足:
127  * 任意 x, 节点编号 2x 和 2x+1 中必选一个
128  * 所有的节点, 其后缀一定在选取集合中。
129  * @return pair<bool,vector<int>>
130  * 无解返回{0,{}}
131  * 有解返回{1,{无序的节点集合}}
132  */
133 pair<bool,vector<int>> run2SAT(){
134     g.runSCC();
135     for(int i = 1;i <= n;i += 2) if(g.scc[i] == g.scc[tr(i)]){
136         return{0,{}};
137     }
138     vector<int> cel(g.sc+1);
139     vector<int> res;
140     for(int i = 1;i <= n/2;i ++){
141     {
142         READD;
143         if(cel[g.scc[i*2-1]]) res.pb(i*2-1);
144         else if(cel[g.scc[i*2]]) res.pb(i*2);
145         else {
146             cel[min(g.scc[i*2-1],g.scc[i*2])] = 1;
147             goto READD;
148         }
149     }
150     return {1,res};
151 }
152
153 };

```

4.2 Hopcroft-Carp.h

```

1  /*****
2  * 二分图匹配(Hopcroft-Carp 算法)
3  * 复杂度  $O(\sqrt{n} * E)$ 
4  * 邻接表存图
5  * Nx 为左端的端点数,使用前先赋值
6  *****/
7  const int N = 3005, INF = 0x3f3f3f3f;
8  vector<int> G[N]; //邻接表
9  int Nx,Ny,k; //x、y 部的点数、k 边数
10 int Mx[N],My[N]; //记录匹配的点的序号
11 int dx[N],dy[N]; //标记数组
12 int dis,u,v;
13 bool used[N];
14 bool bfs()
15 {
16     queue<int> Q;
17     dis = INF; //初始化增广路的最短长度
18     memset(dx,-1,sizeof(dx)); //初始化标记
19     memset(dy,-1,sizeof(dy));
20     for(int i=1;i<=Nx;++i)
21     {
22         if(Mx[i] == -1)
23         {
24             Q.push(i), dx[i] = 0; //将 x 部未匹配的点加入到队列中
25         }
26     }
27     while(!Q.empty())
28     {
29         int u = Q.front();Q.pop();
30         if(dx[u] > dis) break;
31         for(auto v:G[u])
32         {
33             if(dy[v] == -1) //取未标记过的 y 部的点
34             {
35                 dy[v] = dx[u] + 1; //做上标记
36                 if(My[v] == -1) dis = dy[v]; //若点 v 未匹配, 此即为最短增广路
37                 else dx[My[v]] = dy[v] + 1, Q.push(My[v]); //若已经匹配, 则将 v
38                 的匹配点纳入到增广路, 并入队
39             }
40         }
41     }
42     return dis != INF; //若 dis==inf, 说明无增广路
43 }
44 bool DFS(int u)
45 {
46     for(auto v:G[u])
47     {
48         if(!used[v] && dy[v] == dx[u] + 1) //点 u 只能和增广路上的下一个点匹配
49         { //used[v] 表示这个点已被查询过, 它要么已经匹配了, 要么不可匹配
50             used[v] = true;
51             if(My[v] != -1 && dy[v] == dis) continue; //若 v 匹配过, 但是是增广路
52             的终点, 则忽略它
53             if(My[v] == -1 || DFS(My[v])) //若点 v 未匹配, 或者点 v 的匹配点可以找到
54             新的匹配点, 则匹配 u 和 v

```

```

52         {
53             My[v] = u;
54             Mx[u] = v;
55             return true; //匹配成功就返回
56         }
57     }
58 }
59 return false;
60 }
61 int MaxMatch()
62 {
63     int res = 0;
64     memset(Mx, -1, sizeof(Mx));
65     memset(My, -1, sizeof(My));
66     while(bfs())
67     {
68         memset(used, false, sizeof(used));
69         for(int i = 1; i <= Nx; ++i)
70             if(Mx[i] == -1 && DFS(i)) //取 x 部未匹配的点进行深搜
71                 ++res; //若成功匹配则做出贡献
72     }
73     return res;
74 }
75 void solve()
76 {
77     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
78     cin >> Nx >> Ny >> k;
79     while(k--)
80     {
81         cin >> u >> v;
82         G[u].push_back(v);
83     }
84     cout << MaxMatch() << "\n";
85 }

```


4.3 图上大封装.h

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using pii = pair<int,int>;
4  class graph {
5  public:
6      using tii = array<int,3>;
7      vector<vector<pii>> cnj;
8      vector<pii> links;
9      vector<tii> edges;
10     vector<vector<int>> bcc, h;
11     vector<int> dfn, scc, sz, fa;
12     int sc,dc;
13     int n;
14     graph(int _n) {
15         n = _n;
16         reset(cnj);
17         reset(h);
18     };
19
20     graph(vector<vector<pii>> _cnj) {
21         n = _cnj.size();
22         cnj = _cnj;
23     }
24
25 private:
26     void reset(auto &t) {
27         t.clear();
28         t.resize(n + 1);
29     };
30
31 public:
32     void addOrdEdge(int u, int v, int w = 1) {
33         cnj[u].push_back({v, w});
34         edges.pb({u,v,w});
35     }
36
37     void addUnordEdge(int u, int v, int w = 1) {
38         cnj[u].push_back({v, w});
39         cnj[v].push_back({u, w});
40         //邻接表↑--链式前向星↓
41         links.pb({u,v});
42         h[u].pb(links.size()-1);
43         links.pb({v,u});
44         h[v].pb(links.size()-1);
45         edges.pb({u,v,w});
46         edges.pb({v,u,w});
47     }
48
49     // 强连通分量缩点
50     void runSCC() {
51         reset(dfn), reset(scc), reset(sz);
52         vector<int> low(n + 1), s, vst(n + 1), sz(n + 1);
53         sc = 0;
54         int tp = 0, dfncnt = 0;
55         using itf = function<void(int)>;
56         itf dfs = [&](int u) -> void
57         {

```

```

58         low[u] = dfn[u] = ++dfncnt;
59         s.pb(u), vst[u] = 1;
60         for (auto [v, _] : cnj[u])
61         {
62             if (!dfn[v])
63             {
64                 dfs(v);
65                 low[u] = min(low[u], low[v]);
66             }
67             else if (vst[v])
68             {
69                 low[u] = min(low[u], dfn[v]);
70             }
71         }
72         if (dfn[u] == low[u])
73         {
74             sc++;
75             while (s.back() != u)
76             {
77                 scc[s.back()] = sc;
78                 sz[sc]++;
79                 vst[s.back()] = 0;
80                 s.pop_back();
81             }
82             scc[s.back()] = sc;
83             sz[sc]++;
84             vst[s.back()] = 0;
85             s.pop_back();
86         }
87     };
88     for (int i = 1; i <= n; i++)
89         if (!dfn[i])
90             dfs(i);
91 }
92
93 /**双联通分量缩点
94  * E: 边双: 支持重边
95  * D: 点双
96  */
97 void runBCC(char mod = 'E') {
98     reset(bcc), reset(sz);
99     if(mod == 'E') {
100         auto [ib, _] = this->findBridge();
101         using itf = function<void(int, int)>;
102         dc = 0;
103         itf dfs = [&](int u, int tv) -> void {
104             bcc[u] = {dc};
105             for(auto rv:h[u]){
106                 int v = links[rv].se;
107                 if(ib[rv] || bcc[v].size() || rv == (tv^1)) continue;
108                 else dfs(v,rv);
109             }
110         };
111         for(int i = 1; i <= n; i++) if(!bcc[i].size()) ++dc, dfs(i, 0);
112     } else if(mod == 'D') {
113         auto ic = this->findCut();
114         using itf = function<void(int, int)>;

```

```

115         dc = 0;
116         itf dfs = [&](int u,int f) -> void {
117             for(auto [v,_]:cnj[u]) {
118                 if(v == f || (!ic[v] && bcc[v].size())) continue;
119                 else if(ic[v] || bcc[v].size()) bcc[v].pb(dc);
120                 else bcc[v].pb(dc),dfs(v,u);
121             }
122         };
123         for(int i = 1;i <= n;i ++) if(!bcc[i].size()) +
+dc,bcc[i].pb(dc),dfs(i,0);
124     } else assert(0);
125 }
126
127 /**最小生成树, 默认 Kruskal
128  * Kruskal:O(m log m) for 稀疏图
129  * Prim:O((n+m) log n) for 稠密图
130  * 返回邻接表
131  */
132 vector<vector<pii>> runMST(string method = "Kruskal") {
133     graph ng(n);
134     using tii = array<int,3>;
135     const int INF = 1e17;
136     if (method == "Kruskal") {
137         priority_queue<tii,vector<tii>,greater<tii>> q;
138         DSU dsu(n+1);
139         for(auto [u,v,w]:edges) q.push({w,u,v});
140         while(q.size())
141         {
142             auto [w,u,v] = q.top();
143             q.pop();
144             if(!dsu.same(u,v)) {
145                 ng.addUnordEdge(u,v,w);
146                 dsu.merge(u,v);
147             }
148         }
149         return ng.cnj;
150     }
151     else if (method == "Prim") {
152         vector<int> dis(n+1,INF);
153         using fii = array<int,4>;
154         priority_queue<fii,vector<fii>,greater<fii>> q;
155         q.push({0,0,1});
156         dis[1] = 0;
157         vector<int> LOCK(n+1);
158         while(q.size())
159         {
160             auto [_,w,f,u] = q.top();
161             q.pop();
162             if(LOCK[u]) continue;
163             else LOCK[u] = 1;
164             if(f) ng.addUnordEdge(f,u,w);
165             for(auto [v,wv]:cnj[u])
166             {
167                 if(dis[v] > dis[u] + wv) {
168                     dis[v] = dis[u] + wv;
169                     q.push({dis[v],wv,u,v});
170                 }
171             }
172         }
173     }
174 }

```

```

171     }
172     }
173     return ng.cnj;
174 } else assert(0);
175 }
176
177 /**
178  * tarjin 找桥
179  * 返回{res,ib}中 ib[x]!=0 代表某点的出边 x 是桥, res 为桥的列表
180  */
181 pair<vector<int>,vector<pii>> findBridge() {
182     reset(dfn),reset(fa);
183     vector<int> low(n+1),ib(links.size()+2);
184     vector<pii> res;
185     int dfncnt = 0;
186     using itf = function<void(int,int)>;
187     itf dfs = [&](int u,int tv) -> void {
188         fa[u] = tv;
189         low[u] = dfn[u] = ++dfncnt;
190         // for(auto [v,_]:cnj[u]) {
191         for(auto rv:h[u]){
192             int v = links[rv].se;
193             if(!dfn[v]){
194                 dfs(v,rv);
195                 low[u] = min(low[u],low[v]);
196                 if(low[v] > dfn[u]) ib[rv] = ib[rv^1] = 1,res.pb({u,v});
197             } else if(dfn[v] < dfn[u] && rv != (tv^1))
198                 low[u] = min(low[u],dfn[v]);
199         }
200     };
201     for(int i = 1;i <= n;i++) if(!dfn[i]) dfs(i,0);
202     return {ib,res};
203 }
204
205 // tarjin 找割点
206 vector<int> findCut() {
207     reset(dfn);
208     vector<int> low(n+1),ic(n+1);
209     vector<bool> vis(n+1); //能不能去掉?
210     int dfscnt = 0;
211     using itf = function<void(int,int)>;
212     itf dfs = [&](int u,int f) {
213         // cout << "et:" << u << " ";
214         vis[u] = 1;
215         low[u] = dfn[u] = ++dfscnt;
216         int ch = 0;
217         for(auto [v,_]:cnj[u]){
218             if(!vis[v]){
219                 ch++;
220                 dfs(v,u);
221                 low[u] = min(low[u],low[v]);
222                 if(f != u && low[v] >= dfn[u] && !ic[u]) ic[u] = 1;
223             } else if(v != f) low[u] = min(low[u],dfn[v]);
224         }
225         if(f == u && ch >= 2 && !ic[u]) ic[u] = 1;
226     };
227     for(int i = 1;i <= n;i++) if(!vis[i]) dfscnt = 0, dfs(i,i);
228     // dfs(1,1);

```

```
229         return ic;
230     }
231
232     // 生成 kruskal 重构树，返回邻接表
233     void kruskalRefactor() {
234     }
235 };
```

4.4 奇偶环.h

```

1 //题目：有一张简单无向图，问加至少加多少条新边，才能使得图上既有奇环又有偶环，或不可
   能
2
3 //涉及：树的直径、二分图染色
4 //      貌似只能判定、不能求数量
5
6 struct node //存边
7 {
8     int u;
9     int v;
10    int id;
11 };
12 struct D //用于求树的直径的结构体
13 {
14     int d; //直径 d = maxn + submaxn
15     int maxn=0; //最大子树结点数
16     int submaxn=0; //次大子树结点数
17 }d[N];
18 vector<node> edge; //邻接表
19
20 int odd=0,even=0; //是否存在奇/偶环
21
22 vector<int> e[N]; //存边
23 int col[N]={0}; //二分图染色法
24 int vis[N]={0}; //
25 int dfn[N]={0},cnt=0; //深搜序
26
27 int fa[N]={0}; //并查集
28 int siz[N]={0}; //存连通块的直径
29
30 int pre[N]={0},cov[N]={0}; //前驱点、染色体
31 int pre_edge[N]={0}; //前驱边
32
33 int find(int u) //并查集
34 {
35     if(fa[u]==u)return u;
36     else return fa[u]=find(fa[u]);
37 }
38 void uni(int u,int v)
39 {
40     int fa1=find(u);
41     int fa2=find(v);
42     fa[fa1]=min(fa1,fa2);
43     fa[fa2]=min(fa1,fa2);
44 }
45
46 void dfs(int u,int fa)
47 {
48     if(fa!=0)uni(u,fa); //深搜时顺便建立父子关系
49
50     dfn[u]++; //记录深搜序
51     col[u]=col[fa]^1; //染色
52     vis[u]++; //搜过的点
53
54     d[u].maxn=0; //系列初始化
55     d[u].submaxn=0;

```

```

56     d[u].d=0;
57
58     for(auto re:e[u])
59     {
60         int v=u^edge[re].u^edge[re].v;    //取儿子结点
61
62         if(v==fa)continue;    //跳过父亲
63
64         if(col[v]<0)    //儿子未染色
65         {
66             pre[re]=u;        //记录 边 的前驱点
67             pre_edge[v]=re;    //记录 点 的前驱边
68             dfs(v,u);
69
70             if(d[u].maxn<d[v].maxn+1)    //回溯时更新最大和次大子树，用于求直径
71             {
72                 d[u].submaxn=d[u].maxn;
73                 d[u].maxn=d[v].maxn+1;
74             }
75             else if(d[u].maxn==d[v].maxn+1)
76             {
77                 d[u].submaxn=d[u].maxn;
78             }
79             else if(d[u].submaxn<d[v].maxn+1)
80             {
81                 d[u].submaxn=d[v].maxn+1;
82             }
83         }
84     }
85     else if(col[v]==col[u]^1) //找到偶环
86     {
87         even++; //记录
88     }
89     else if(col[v]==col[u]) //找到奇环
90     {
91         odd++; //记录
92         if(dfn[v]>dfn[u])continue;    //儿子的深搜序更大，说明是个孙子，跳过，返祖边
只走一次
93         //否则遇到了祖宗
94         cov[re]++;    //标记这条边
95
96         int pnt=u; //记录当前点
97
98         int pe=pre_edge[u]; //取出当前点的前驱边，准备走返祖边
99
100         while(!even)    //若没有发现偶环，则尝试用两个奇环制造偶环
101         {
102             if(cov[pe])even++; //找到了一条被 cov 标记的前驱边，说明本奇环与其他奇环相
交，可以制造偶环
103
104             cov[pe]++;    // 标记沿途的边
105
106             pnt=pre[pe]; // 取出 边 的前驱点
107             pe=pre_edge[pnt]; //再取 前驱点 的 前驱边
108
109             if(pnt==v || pnt==-1 || pe==0)break;    //若回到了祖宗 v、找到不存在的前驱点或
边，直接退出

```

```

110     }
111   }
112 }
113 d[u].d=d[u].maxn+d[u].submaxn; //求直径
114 }
115 void solve()
116 {
117   //这里要补初始化//
118   edge.clear();
119   int n,m;
120   cin>>n>>m;
121
122   edge.push_back((node){-1,-1,0}); //这个是凑数用的
123   pre[0]=-1;
124
125   for(int i=1;i<=m;i++) //存边
126   {
127     int u,v;
128     cin>>u>>v;
129     edge.push_back((node){u,v,i});
130
131     e[u].push_back(i); //注意:邻接表存边号
132     e[v].push_back(i);
133   }
134
135   for(int i=1;i<=n;i++) col[i]=-1,fa[i]=i; //初始化
136   col[0]=0; //0点不存在,但初始化为0备用
137
138   for(int i=1;i<=n;i++) //深搜,找环,顺便求连通块的直径
139   {
140     if(!vis[i]) dfs(i,0);
141   }
142
143   for(int i=1;i<=n;i++)
144   {
145     int f=find(i);
146     siz[f]=max(siz[f],d[i].d); //更新连通块的直径
147   }
148
149   if(n<4) cout<<-1<<"\n"; //n<4 is -1
150
151   else if(odd&&even) //若有奇又有偶,直接输出
152   {
153     cout<<0<<"\n";
154   }
155   else if(even) //仅有偶环,只需再加一条边
156   {
157     cout<<1<<"\n";
158   }
159   else //若无偶环,则需要尝试用若干直径制造偶环
160   {
161     map<int,int> is;
162     priority_queue<int> q;
163     for(int i=1;i<=n;i++) //找连通块
164     {
165       int f=find(i);
166       if(!is[f])
167       {

```



```

168         q.push(siz[f]); //记录连通块的直径(最长路)
169     }
170     is[f]++; //标记连通块
171 }
172 int temp=0;
173 int ans=0;
174 while(!q.empty())
175 {
176     temp+=q.top(); //取出一个直径，尝试连出一条长度不小于4的路
177     q.pop();
178     ans++; //新边数量增加
179     temp++; //路长增加
180     if(temp>=4)break; //路长不小于4则退出，已经有偶环了
181 }
182 if(!odd)ans++; //如果没有奇环，还需多加一条边
183 cout<<ans<<"\n";
184 }
185 }
186 /*
187 7 9
188 1 2
189 2 3
190 3 1
191 3 5
192 5 4
193 4 7
194 6 7
195 6 4
196 4 3
197 */

```

4.5 FLOW

4.5.1 0_introduction.typ

网络流的核心难点在于建图

模板方面

4.5.1.1 最大流/最小割：按复杂度排序

1. HLPP（预流推进）
2. ISAP
3. Dinic

4.5.1.2 最大流 -> 最小割 -> 最短路：

此为技巧，无固定模板，只适用于平面图，可以将复杂的网络流问题转化为简单的最短路问题，只需取原图的对偶图即可

4.5.1.3 费用流：

1. 最小费用最大流：最基础的模板
2. 最大费用最大流：原费用取负即可 费用流不建议用 DJ，建议使用 SPFA

4.5.1.4 上下界网络流：

还在学

4.5.2 DINIC.h

```

1 //V1
2 //Dinic 算法, 求解最大流
3 //O (M*N^2)
4 //是对 EK 算法的优化
5 //考虑到 BFS 每次都只能找一条路, 思考可不可以一口气找多条路, 于是就有了 Dinic 算法
6 //先用 BFS 求分层图, 操作和 EK 算法一样, 但不计算, 只分层
7 //然后用 DFS 在分层图上找增广路, 并计算流量, 更新残量图
8 //其中, 运用了先前弧剪枝, 体现在 now 数组上, 即改变遍历链式前向星的起点, 找过的节点不
  重复寻找
9 const ll M=1000000000000;
10 int n,m,s,t,si;
11 ll sum=0;
12 struct edge
13 {
14     int to;
15     int nex;
16     ll w;
17 }a[500005];
18 int head[500005]={0},cnt=1; //链式前向星相关
19 void add(int u,int v,ll w) //链式前向星
20 {
21     //a[++cnt].from=u;
22     a[++cnt].to=v;
23     a[cnt].w=w;
24     a[cnt].nex=head[u];
25     head[u]=cnt;
26
27     //a[++cnt].from=v; //regard edge
28     a[++cnt].to=u;
29     a[cnt].w=0;
30     a[cnt].nex=head[v];
31     head[v]=cnt;
32 }
33 ll dis[500005]={0};
34 int now[500005]={0};
35
36 int bfs() //在残量网络中构造分层图
37 {
38     //事实上, 深度标记 dis[i]就是分层图, 无穷深代表不在图内
39     for(int i=1;i<=n;i++)dis[i]=M;//重置分层图
40     queue<int> q;
41     dis[s]=0; //标记源点为 0 深度
42     now[s]=head[s];
43     q.push(s);
44     while(!q.empty())
45     {
46         int temp=q.front();
47         q.pop();
48         for(int i=head[temp];i;i=a[i].nex)
49         {
50             int v=a[i].to;
51             if(dis[v]!=M||a[i].w<=0)continue;
52             q.push(v);
53             dis[v]=dis[temp]+1; //分层, 标记深度
54         }
55     }
56 }

```

```

55         now[v]=head[v];           //若点在分层图内，就给它的当前弧数组赋值，表示它可
    以找儿子
56
57         if(v==t)return 1;         //找到汇点就可退出，因为已经标好汇点的深度了
58     }
59 }
60 return 0;
61 }
62 ll dfs(int x,ll delta)           //深搜求解 父节点->子节点 的流量
63 {                                 //源点的父节点->源点 就是最大流
64
65     if(x==t)return delta;         //delta 最终会是某条路的最小权
66     ll k,res=0;
67
68     for(int i=now[x];i&&delta;i=a[i].nex) //从当前弧出发找边，当 delta 为 0 时，
    表示流量到达上限
69     {
70         now[x]=i;//更新
71
72         int v=a[i].to;//取儿子
73         if(a[i].w<=0||dis[v]!=dis[x]+1)continue;//跳过 0 权边和非法边，即深度不+1
    的边
74         k=dfs(v,min(a[i].w,delta)); //求出 x->v 的流量
75
76         if(k==0)dis[v]=M;         //发现流量 x->v 等于 0，删点
77
78         a[i].w-=k;                //更新容量
79         a[i^1].w+=k;
80
81         res+=k;                   //更新总和
82         delta-=k;                 //更新剩余容量
83     }
84     return res;                  //返回流入点 x 的流量
85 }
86 void Dinic()
87 {
88     while(bfs()) //bfs 构建分层图
89     {
90         sum+=dfs(s,M);           //dfs 找增广路并计算
91     }
92 }
93 void solve()
94 {
95     cin>>n>>m>>s>>t;
96     for(int i=1;i<=m;i++)
97     {
98         int u,v;
99         ll w;
100         cin>>u>>v>>w;
101         add(u,v,w);
102     }
103     while(bfs()) //bfs 构建分层图
104     {
105         sum+=dfs(s,M);           //dfs 找增广路并计算
106     }
107     cout<<sum<<"\n";
108 }

```

```

109
110 //V2
111 class maxFlow//AC
112 {
113 private:
114     class edge
115     {
116     public:
117         ll int nxt, cap, flow;
118         pair<int, int> revNodeIdx;
119     public:
120         edge(int _nxt, int _cap)
121         {
122             nxt = _nxt, cap = _cap, flow = 0;
123         }
124         void setRevIdx(int _i, int _j) { revNodeIdx = {_i, _j}; }
125     };
126     vector<vector<edge>> edgeList;
127     vector<int> dep, fir;
128     ll int maxFlowAns;
129     int T, S;
130 public:
131     maxFlow(int _n)
132     {
133         maxFlowAns = 0;
134         S = 1;
135         T = _n;
136         edgeList.resize(_n + 1);
137         dep.resize(_n + 1);
138         fir.resize(_n+1);
139     }
140     void resetTS(int _T, int _S) { T = _T, S = _S; }
141
142     void addedge(int _u, int _v, int _w)
143     {
144         edgeList[_u].push_back(edge(_v, _w));
145         edgeList[_v].push_back(edge(_u, 0));
146         edgeList[_u][edgeList[_u].size() - 1].setRevIdx(_v,
147         edgeList[_v].size() - 1);
148         edgeList[_v][edgeList[_v].size() - 1].setRevIdx(_u,
149         edgeList[_u].size() - 1);
150     }
151
152     bool bfs()
153     {
154         queue<int> que;
155         for (auto x = dep.begin(); x != dep.end(); x++) (*x) = 0;
156         dep[S] = 1;
157         que.push(S);
158         while (que.size())
159         {
160             ll int at = que.front();
161             que.pop();
162             for (int i = 0; i < edgeList[at].size(); i++)
163             {
164                 auto tar = edgeList[at][i];
165                 if ((!dep[tar.nxt]) && (tar.flow < tar.cap))
166                 {

```

```

165         dep[tar.nxt] = dep[at] + 1;
166         que.push(tar.nxt);
167     }
168 }
169 }
170 return dep[T];
171 }
172
173 ll int dfs(int at, ll int flow)
174 {
175     if ((at == T) || (!flow))
176         return flow; // 到了或者没了
177     ll int ret = 0; // 本节点最大流
178     for (int &i = fir[at]; i < edgeList[at].size(); i++)
179     {
180         auto tar = edgeList[at][i]; // 目前遍历的边
181         int tlFlow = 0; // 目前边的最大流
182         if (dep[at] == dep[tar.nxt] - 1) // 遍历到的边为合法边
183         {
184             tlFlow = dfs(tar.nxt, min((ll)tar.cap - tar.flow, flow -
185 ret));
186             if (!tlFlow)
187                 continue; // 若最大流为 0, 什么都不做
188             ret += tlFlow; // 本节点最大流累加
189             edgeList[at][i].flow += tlFlow; // 本节点实时流量
190             edgeList[tar.revNodeIdx.first][tar.revNodeIdx.second].flow -=
191 tlFlow; // 反向边流量
192             if (ret == flow)
193                 return ret; // 充满了就不继续扫了
194         }
195     }
196     return ret;
197 }
198
199 ll int dinic()
200 {
201     if (maxFlowAns)
202         return maxFlowAns;
203     while (bfs())
204     {
205         for(auto x = fir.begin(); x != fir.end(); x++) (*x) = 0;
206         maxFlowAns += dfs(S, INT_MAX);
207     }
208     return maxFlowAns;
209 }
210 };

```

4.5.3 DJ.h

```

1  int n,m,s,t;
2  ll max_flow=0,min_cost=0;
3  struct p3381
4  {
5      int to;
6      int nex;
7      ll w;
8      ll c;
9  }a[20005];
10 int head[5005]={0},cnt=1,x[5005][2]={0};
11 void add(int u,int v,ll w,ll c)
12 {
13     a[++cnt].to=v;
14     a[cnt].nex=head[u];
15     a[cnt].w=w;
16     a[cnt].c=c;
17     head[u]=cnt;
18
19     a[++cnt].to=u;
20     a[cnt].nex=head[v];
21     a[cnt].w=0;
22     a[cnt].c=-c;
23     head[v]=cnt;
24 }
25 ll dist[5005]={0};    //最小费用数组
26 ll h[5005]={0};      //势数组, dj 不能跑负边, 所以要操作
27 struct node
28 {
29     int id;
30     bool operator < (const node &x) const {return dist[x.id]<dist[id];}
31 };
32 //////////////////////////////////////
33 bool DJ()
34 {
35     int vis[5005]={0};
36     priority_queue<node> q;
37     q.push({s});
38     for(int i=1;i<=n;i++)dist[i]=inf;
39     dist[s]=0;
40     while(!q.empty())
41     {
42         node temp=q.top();
43         q.pop();
44         vis[temp.id]=0;
45         for(int i=head[temp.id];i;i=a[i].nex)
46         {
47             int v=a[i].to;
48             if(dist[v]>dist[temp.id]+a[i].c+h[temp.id]-h[v]&&a[i].w>0)
49             {
50                 dist[v]=dist[temp.id]+a[i].c+h[temp.id]-h[v];
51                 if(!vis[v])
52                 {
53                     vis[v]=1;
54                     q.push({v});
55                 }
56                 x[v][0]=temp.id;

```

```

57         x[v][1]=i;
58     }
59 }
60 }
61 if(dist[t]==inf)return false;
62 return true;
63 }
64 void EK();//EK 算法求解
65 {
66     ll cost=0,minn=inf;
67     for(int p=t;p!=s;p=x[p][0])
68     {
69         minn=min(minn,a[x[p][1]].w);
70     }
71     for(int p=t;p!=s;p=x[p][0])
72     {
73         a[x[p][1]].w-=minn;
74         a[x[p][1]^1].w+=minn;
75     }
76     cost=(dist[t]-(h[s]-h[t]))*minn;
77     min_cost+=cost;
78     max_flow+=minn;
79     for(int i=1;i<=n;i++)h[i]+=dist[i];
80 }
81 void solve()
82 {
83     cin>>n>>m>>s>>t;
84     for(int i=1;i<=m;i++)
85     {
86         int u,v;
87         ll w,c;
88         cin>>u>>v>>w>>c;
89         add(u,v,w,c);
90     }
91     while(DJ())
92     {
93         EK();
94     }
95     cout<<max_flow<<" "<<min_cost<<"\n";
96 }

```


4.5.4 EK.h

```

1  int n,m,s,t;
2  ll max_flow=0,min_cost=0;
3  struct p3381
4  {
5      int to;
6      int nex;
7      ll w;
8      ll c;
9  }a[200005];
10 int head[5005]={0},cnt=1,x[5005][2]={0};
11 void add(int u,int v,ll w,ll c)
12 {
13     a[++cnt].to=v;
14     a[cnt].nex=head[u];
15     a[cnt].w=w;
16     a[cnt].c=c;
17     head[u]=cnt;
18
19     a[++cnt].to=u;
20     a[cnt].nex=head[v];
21     a[cnt].w=0;
22     a[cnt].c=-c;
23     head[v]=cnt;
24 }
25 ll dist[5005]={0};    //最小费用数组
26 ///////////////////////////////////////////////////
27 //基于 EK 算法的最小费用最大流
28 //用 SPFA 对费用求增广路
29 //用 EK 算法中的更新操作求解
30 bool SPFA() //SPFA 找增广路
31 {
32     int vis[5005]={0};
33     queue<int> q;
34     q.push(s);
35     for(int i=1;i<=n;i++)dist[i]=inf;
36     dist[s]=0;
37     while(!q.empty())
38     {
39         int temp=q.front();
40         q.pop();
41         vis[temp]=0;
42         for(int i=head[temp];i;i=a[i].nex)
43         {
44             int v=a[i].to;
45             if(dist[v]>dist[temp]+a[i].c&& a[i].w>0)
46             {
47                 dist[v]=dist[temp]+a[i].c;
48                 if(!vis[v])
49                 {
50                     vis[v]=1;
51                     q.push(v);
52                 }
53                 x[v][0]=temp;
54                 x[v][1]=i;
55             }
56         }
57     }
58 }

```

```

57     }
58     if(dist[t]==inf)return false;
59     return true;
60 }
61 void EK();//EK 算法求解
62 {
63     ll cost=0,minn=inf;
64     for(int p=t;p!=s;p=x[p][0])
65     {
66         minn=min(minn,a[x[p][1]].w);
67     }
68     for(int p=t;p!=s;p=x[p][0])
69     {
70         a[x[p][1]].w-=minn;
71         a[x[p][1]^1].w+=minn;
72     }
73     cost=dist[t]*minn;
74     min_cost+=cost;
75     max_flow+=minn;
76 }
77 void solve()
78 {
79     cin>>n>>m>>s>>t;
80     for(int i=1;i<=m;i++)
81     {
82         int u,v;
83         ll w,c;
84         cin>>u>>v>>w>>c;
85         add(u,v,w,c);
86     }
87     while(SPFA())
88     {
89         EK();
90     }
91     cout<<max_flow<<" "<<min_cost<<"\n";
92 }

```

4.5.5 HLPP.h

```

1 //比 ISAP 和 DINIC 快一点, 但不稳定  $O(\sqrt{m} * n^2)$ 
2 //思想是从源点开始, 给每条边都推入满的流量, 并设定每个点都可以存储一定的流量 ei, 称超
   额量
3 //且节点会伺机将超额量推向深度低的节点
4 int n,m,s,t;
5 int vis[1500]={0};
6 int head[1500]={0},cnt=1;
7 int h[1500]={0},gap[1500]={0};
8 int sum=0;
9 int e[1500]={0};
10 struct p4722
11 {
12     int to,nex;
13     int w;
14 }a[540005];
15 struct node
16 {
17     bool operator() (int x,int y) const{return h[x]<h[y];}
18 };
19 priority_queue<int,vector<int>,node> q;
20 queue<int> que;
21 void add(int u,int v,int w) //链式前向星
22 {
23     a[++cnt].to=v;
24     a[cnt].nex=head[u];
25     a[cnt].w=w;
26     head[u]=cnt;
27     a[++cnt].to=u;
28     a[cnt].nex=head[v];
29     a[cnt].w=0;
30     head[v]=cnt;
31 }
32 void bfs_rebel() //深度标签初始化, 从汇点广搜
33 {
34     que.push(t);
35     h[t]=0;
36     while(!que.empty())
37     {
38         int temp=que.front();
39         que.pop();
40         gap[h[temp]]++;
41         for(int i=head[temp];i;i=a[i].nex)
42         {
43             int v=a[i].to;
44             if(h[v]==0&&v!=t&&a[i^1].w)
45             {
46                 h[v]=h[temp]+1;
47                 que.push(v);
48             }
49         }
50     }
51 }
52 void push(int u) //推流
53 {
54     int i;

```

```

55     for(i=head[u];i;i=a[i].nex)
56     {
57         int v=a[i].to;
58         if(a[i].w<=0||h[u]!=h[v]+1)continue;//跳过 0 容边和深度不递减的点
59
60         int f=min(e[u],a[i].w); //取残留容量和超额量中的小者
61
62         e[v]+=f;    //推流, 更新
63         e[u]-=f;
64         a[i].w-=f;
65         a[i^1].w+=f;
66
67         if(!vis[v]&&v!=t&&v!=s) //子节点返回队列
68         {
69             vis[v]=1;
70             q.push(v);
71         }
72
73         if(e[u]==0)break;    //超额量分发完毕退出函数
74     }
75 }
76 void rebel(int u)    //重贴标签
77 {
78     int i;
79     h[u]=inf;    //标签初始化
80     for(i=head[u];i;i=a[i].nex)
81     {
82         int v=a[i].to;
83         if(a[i].w&&h[v]+1<h[u])h[u]=h[v]+1;    //找到儿子中的最小者, 使 u 下一次恰
84         好可以给它推流
85     }
86 }
87 void hlpp()    //核心函数
88 {
89     h[s]=n;e[s]=inf; //源点 s 深度为 n, 超额量为无穷
90     for(int i=head[s];i;i=a[i].nex) //源点不入队, 因此在外处理
91     {
92         int v=a[i].to;
93         if(int f=a[i].w)
94         {
95             a[i].w-=f;    //更新网络
96             a[i^1].w+=f;
97             e[s]-=f;
98             e[v]+=f;
99             if(v!=s&&v!=t&&!vis[v]) //注意汇点和重复点不可入队
100             {
101                 q.push(v);
102                 vis[v]=1;
103             }
104         }
105     }
106     while(!q.empty()) //计算最大流
107     {
108         int u=q.top();
109         q.pop();
110         vis[u]=0;    //出队取消标记
111         push(u);//推流

```

```

111         if(e[u]<=0)continue; //无超额流就跳过，不入队
112         gap[h[u]]--; //准备更新深度
113         if(!gap[h[u]]) //本深度无其他点，则比 u 高的点都不可能再给 u 推流，可以把它
们放在 n+1 处
114         {
115             for(int v=1;v<=n;v++)
116             {
117                 if(v!=s&&v!=t&&h[v]>h[u]&&h[v]<n+1)
118                 {
119                     h[v]=n+1;
120                 }
121             }
122         }
123         rebel(u); //重贴标签
124         gap[h[u]]++;
125         q.push(u); //入队
126         vis[u]=1;
127     }
128     sum=e[t];
129 }

```

4.5.6 ISPA.h

```

1  int n,m;
2  struct p3376
3  {
4      int to;
5      ll w=0;
6      int nex;
7  }a[540005];
8  map<pair<int,int>,int> key;
9  int head[1580]={0},cnt=1,dept[1580]={0};
10 void add(int u,int v,ll w)
11 {
12     if(key[{u,v}]!=0)    //去除重边
13     {
14         a[key[{u,v}]].w+=w;
15         return;
16     }
17     a[++cnt].to=v;
18     a[cnt].w=w;
19     a[cnt].nex=head[u];
20     key[{u,v}]=cnt;
21     head[u]=cnt;
22     a[++cnt].to=u;
23     a[cnt].w=0;
24     a[cnt].nex=head[v];
25     key[{v,u}]=cnt;
26     head[v]=cnt;
27 }
28 //.....//
29 //Dinic 算法会调用太多次 bfs, 考虑优化
30 //于是就有了 ISPA 算法
31 //从汇点开始 bfs, 标记深度
32 //再从源点开始 dfs, 对经过的点进行深度修改, 当出现断层时, 算法结束
33 ll ans=0;
34 int g[1580]={0},maxn=0;
35
36 void bfs(int t)    //从汇点 bfs
37 {
38     queue<int> q;
39     bool vis[1580];
40     memset(vis,0,sizeof(vis));
41     q.push(t);
42     vis[t]=true;
43     //g[0]++;
44     while(!q.empty())
45     {
46         int temp=q.front();
47         q.pop();
48         g[dept[temp]]++;
49         maxn=max(dept[temp],maxn);
50         //vis[temp]=true;
51         for(int i=head[temp];i;i=a[i].nex)
52         {
53             int v=a[i].to;
54             if(!vis[v])
55             {
56                 vis[v]=true;

```

```

57         dept[v]=dept[temp]+1;
58         q.push(v);
59     }
60 }
61 }
62 }
63 ll dfs(int p,int s,int t,ll tot)
64 {
65     int i;
66     if(p==t) //到达汇点
67     {
68         ans+=tot; //更新答案
69         return tot; //返回
70     }
71     ll k=0,res=0;
72     for(i=head[p];i&&tot;i=a[i].nex)
73     {
74         int v=a[i].to;
75         if(dept[v]!=dept[p]-1||a[i].w<=0)continue; //残量为0, 或非递减的深度,
        跳过
76         k=dfs(v,s,t,min(a[i].w,tot)); //取 p->v 的流量
77
78         a[i].w-=k; //更新残量
79         a[i^1].w+=k;
80
81         res+=k; //增加总流量
82
83         tot-=k; //残量减少
84     }
85     if(tot>0) //流过来的流量还有余, 使深度增加
86     {
87         dept[p]++;
88         g[dept[p]-1]--;
89         g[dept[p]]++;
90         if(g[dept[p]-1]<=0)dept[s]=n+1; //出现断层, 对 s 的深度进行标记
91     }
92     return res;
93 }
94 void isap(int s,int t)
95 {
96     while(dept[s]<=n)dfs(s,s,t,10000000000); //未出现断层就反复 dfs
97 }
98 void solve()
99 {
100     int s,t;
101     cin>>n>>m>>s>>t;
102     for(int i=1;i<=m;i++)
103     {
104         int u,v;
105         ll w;
106         cin>>u>>v>>w;
107         add(u,v,w);
108     }
109     //ll temp;
110     bfs(t);
111     isap(s,t);
112     cout<<ans<<"\n";

```

113

}

4.5.7 min_Cost.h

```

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

```

```

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

```

```

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

```

4.6 PATH

4.6.1 johonson.h

```

1 // Johnson 算法
2
3 // 带负边的全源最短路
4
5 // 这个算法最重要的功能是使得 dj 能用来处理负边
6
7 // 只跑一次的话复杂度和 spfa 同阶
8 // 跑多次的话会比 spfa 优秀
9
10 // 先建立一个虚拟源点 o, 从该点向所有边连一条边权为 0, 跑一遍 spfa(), 记录点 o 到任意
    点 i 的最短路 h[i]
11 // 再将原边权 w(u,v), 改造为 w(u,v)+h[u]-h[v]
12 // 再以每个点作为起点, 跑 n 轮 dj 即可
13
14 // 最终 d'(s,t)=d(s,t)+h[s]-h[t]
15 #define ll long long
16 #define inf 0x3f3f3f3f
17 int main()
18 {
19     void solve();
20     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
21     solve();
22     return 0;
23 }
24 //Johnson 算法
25 //带负边的全源最短路
26 //先建立一个虚拟源点 o, 从该点向所有边连一条边权为 0, 跑一遍 spfa(), 记录点 o 到任意点
    i 的最短路 h[i]
27 //再将原边权 w(u,v), 改造为 w(u,v)+h[u]-h[v]
28 //再以每个点作为起点, 跑 n 轮 dj 即可
29 int n,m;
30 struct p5905
31 {
32     int to;
33     ll w;
34 };
35 vector<p5905> edge[3005];
36 ll dis[3005]={0},h[3005]={0};
37 int cnt[3005]={0};
38 int vis[3005]={0};
39 queue<int> q;
40 void spfa()
41 {
42     memset(h,inf,sizeof(h));
43     h[0]=0;
44     q.push(0);
45     vis[0]++;
46     while(!q.empty())
47     {
48         int t=q.front();
49         q.pop();
50         cnt[t]++;
51         if(cnt[t]>n)

```

```

52     {
53         cout<<-1<<"\n";
54         exit(0);
55     }
56     for(auto re:edge[t])
57     {
58         int v=re.to;
59         ll w=re.w;
60         if(h[v]>h[t]+w)
61         {
62             h[v]=h[t]+w;
63             if(!vis[v])
64             {
65                 vis[v]++;
66                 q.push(v);
67             }
68         }
69     }
70     vis[t]--;
71 }
72 }
73 struct node
74 {
75     int id;
76     ll w;
77     bool operator < (const node &x) const {return x.w<w;}
78 };
79 void dj(int f)
80 {
81     memset(dis,inf,sizeof(dis));
82     int vv[3005]={0};
83     priority_queue<node> qq;
84     dis[f]=0;
85     qq.push((node){f,dis[f]});
86     while(!qq.empty())
87     {
88         node temp=qq.top();
89         qq.pop();
90         int t=temp.id;
91         ll w=temp.w;
92         if(vv[t])continue;
93         vv[t]++;
94         dis[t]=w;
95         for(auto re:edge[t])
96         {
97             int v=re.to;
98             ll tw=re.w;
99             if(vv[v])continue;
100             if(dis[v]>w+tw)
101             {
102                 dis[v]=w+tw;
103                 qq.push((node){v,dis[v]});
104             }
105         }
106     }
107 }
108 void solve()
109 {

```

```

110     cin>>n>>m;
111     while(m-->0)
112     {
113         int u,v;
114         ll w;
115         cin>>u>>v>>w;
116         edge[u].push_back({v,w});
117     }
118     for(int i=1;i<=n;i++)
119     {
120         edge[0].push_back({i,0});
121     }
122     spfa();
123     for(int i=1;i<=n;i++)
124     {
125         for(auto &re:edge[i])
126         {
127             int v=re.to;
128             re.w+=h[i]-h[v];    //修改边权
129         }
130     }
131     ll ans=0;
132     for(int i=1;i<=n;i++)
133     {
134         dj(i);
135         ans=0;
136         for(int j=1;j<=n;j++)
137         {
138             if(dis[j]>1e9)ans+=j*(1e9);
139             else
140             {
141                 ans+=j*(dis[j]+h[j]-h[i]);//这里 j*是题目要求
142                 //真正的 i~j 的距离是括号内的表达式
143             }
144         }
145         cout<<ans<<"\n";
146     }
147 }

```

4.6.2 k_path.h

```

1  #define ll long long
2  #define N 5050
3  const ll inf=1e17+7;
4  int n;
5  struct node //边
6  {
7      int v;
8      ll w;
9      int is=0;
10 };
11 struct ed //用于优先队列
12 {
13     int v;
14     ll w;
15     ll d;
16     bool operator < (const ed x) const
17     {
18         return x.w<w;
19     }
20 };
21 queue<ll> ans; //答案队列
22 vector<node> e[N]; //邻接表
23 ll d[N]={0}; // d[i]终点到点 i 的最短路, 也是点 i 到终点的最短路
24 void dj(int s)
25 {
26     priority_queue<ed> q;
27     vector<int> vis(n+10,0);
28     for(int i=1;i<=n;i++)d[i]=inf;//初始化
29     d[s]=0;
30     q.push((ed){s,d[s],0}); //终点入队
31     while(!q.empty())
32     {
33         auto r=q.top();
34         q.pop();
35         int u=r.v;
36
37         if(vis[u])continue;
38         vis[u]=1;
39         d[u]=r.w;
40
41         for(auto re:e[u])
42         {
43             int v=re.v;
44             ll w=re.w;
45             if(re.is)continue;//仅搜反边
46             if(vis[v]||d[v]<=w+d[u])continue;
47             d[v]=w+d[u]; //可松弛
48             q.push((ed){v,d[v],0});
49         }
50     }
51 }
52 void astar(int s,int t,ll &k) //A*算法
53 {
54     priority_queue<ed> q;
55     vector<int> vis(n+10,0);

```

```

56  q.push((ed){s,d[s],0}); //起点入队
57  while(!q.empty())
58  {
59      auto temp=q.top();
60      q.pop();
61
62      int u=temp.v;
63
64      vis[u]++; //统计出队次数
65
66      if(vis[t]<=k&&u==t) //若只求第 k 短, 只需 vis[t]==k, 记录并退出
67      {
68          ans.push(temp.d);
69      }
70      if(vis[t]==k) return;
71
72      for(auto re:e[u])
73      {
74          if(!re.is) continue;
75          if(vis[u]>k) continue; //跳过出队次数大于 k 的
76
77          int v=re.v;
78          ll w=re.w;
79          ll dis=temp.d;
80          dis=dis+w; //此为从起点到 v 的距离
81
82          q.push((ed){v,d[v]+dis,dis});
83          //d[v]+dis 是估计距离, dis 是起点到 v 的距离
84          //按估计距离升序排列
85      }
86  }
87 }
88 void solve()
89 {
90     int s,t; //起点和终点
91     int m; //边数
92     ll k; //第 k 短
93     cin>>n>>m>>k;
94
95     s=n;t=1; //处理终点和起点
96
97     for(int i=1;i<=m;i++)
98     {
99         ll c;
100         int u,v;
101         cin>>u>>v>>c;
102         e[u].push_back((node){v,c,1}); //建立正边
103         e[v].push_back((node){u,c,0}); //建立反边
104     }
105
106     if(s==t) k++; //起点与终点重合
107
108     dj(t); //对 t 求 dj, 走反边
109
110     astar(s,t,k); //求 s 到 t 的最短的 k 条路
111
112     for(int i=1;i<=k;i++)
113     {
114         if(s==t)

```



```
115     {
116         cout<<0<<"\n";
117         continue;
118     }
119     if(!ans.empty())
120     {
121         cout<<ans.front()<<"\n";
122         ans.pop();
123     }
124     else cout<<-1<<"\n";
125 }
126 }
```

4.6.3 SPFA+SLE.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  const int inf =1e9+7;
5  ////////////////
6  struct p3008 //链式前向星
7  {
8      int to;
9      int nex;
10     int v;
11 }a[300000];
12 int head[25006]={0},cnt=0;
13
14 int dis[25006]={0}; //距离
15
16 void add(int u,int v,int w)
17 {
18     a[++cnt].nex=head[u];
19     a[cnt].to=v;
20     a[cnt].v=w;
21     head[u]=cnt;
22 }
23 ////////////////
24 //SPFA, 但是双端队列优化
25 //小的放队头, 大的放队尾
26 void spfa(int t,int s)
27 {
28     for(int i=1;i<=t;i++)dis[i]=inf; //初始化距离
29     deque<int> q;
30     vector<int> vis(t+1),cou(t+1);
31
32     q.push_front(s); //起点入队
33
34     dis[s]=0; //起点初始化
35     vis[s]++;
36
37     while(!q.empty())
38     {
39         int u=q.front(); //取出队头
40         q.pop_front();
41
42         cou[u]++;
43
44         if(cou[u]>t)return; //遍历次数过大, 出现负环
45
46         for(int i=head[u];i;i=a[i].nex) //遍历儿子
47         {
48             int v=a[i].to;
49             int w=a[i].v;
50             if(dis[v]>dis[u]+w) //可松弛
51             {
52                 dis[v]=dis[u]+w; //松弛操作
53
54                 if(!vis[v])
55                 {
56                     if(q.empty())q.push_back(v); //特判! 队空则随便入队

```

```

57         else if(dis[q.front()]>=dis[v])q.push_front(v); //小的放队
    头
58         else q.push_back(v); //大的放队尾
59         vis[v]++;
60     }
61 }
62 }
63     vis[u]--; //取消入队标记
64 }
65 }
66 void solve()
67 {
68     int t,r,p,s;
69     cin>>t>>r>>p>>s;
70     for(int i=1;i<=r;i++)
71     {
72         int u,v;
73         int w;
74         cin>>u>>v>>w;
75         add(u,v,w);
76         add(v,u,w);
77     }
78     for(int i=1;i<=p;i++)
79     {
80         int u,v;
81         int w;
82         cin>>u>>v>>w;
83         add(u,v,w);
84     }
85
86     spfa(t,s); //建好边调用即可，t 是点数，s 是起点
87
88     for(int i=1;i<=t;i++)
89     {
90         if(dis[i]>=inf)cout<<"NO PATH\n";
91         else cout<<dis[i]<<"\n";
92     }
93 }

```

4.6.4 SPFA.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  ///////////////////////////////////
5  struct p4779SPFA //链式前向星
6  {
7      int to;
8      int nex;
9      ll w;
10 }star[550000];
11 int head[100005]={0},cnt=0;
12 void add(int u,int v,ll w)
13 {
14     star[++cnt].to=v;
15     star[cnt].nex=head[u];
16     star[cnt].w=w;
17     head[u]=cnt;
18 }
19 ///////////////////////////////////
20
21 //SPFA,一款暴力的最短路算法
22 //常用于网络流、判负环、带负边的最短路
23 //暴力版的dj, 每次松弛都让未入队的点入队, 直到无法松弛为止
24 int n,m,s;
25 ll dis[100005]={0}; //距离
26 int times[100005]={0}; //遍历次数
27 bool vis[100005]; //入队情况
28 void SPFA()
29 {
30     memset(vis, 0 ,sizeof(vis)); //清空 vis
31     for(int i=1;i<=n;i++)dis[i]=2147483647; //距离初始化为无穷大
32     dis[s]=0; //起点归零
33     queue<int> q; //队列
34     q.push(s);
35     while(!q.empty())
36     {
37         int temp=q.front();
38         q.pop();
39         times[temp]++;
40         vis[temp]=false; //出队判定
41
42         if(times[temp]>2*n) //判负环 , 遍历次数过多说明出现了负环
43         {
44             times[s]=n+10; //做标记
45             break;
46         }
47
48         for(int i=head[temp];i;i=star[i].nex) //遍历儿子
49         {
50             int v=star[i].to;
51             if(dis[v]>dis[temp]+star[i].w) //可松弛
52             {
53                 dis[v]=dis[temp]+star[i].w;//松弛
54                 if(!vis[v]) //未入队则入队
55                 {

```

```

56         q.push(v);
57         vis[v]=true;
58     }
59 }
60 }
61 }
62
63 }
64 void solve()
65 {
66     //spfa 用于求最短路
67     cin>>n>>m>>s;
68     for(int i=1;i<=m;i++)
69     {
70         int u,v;
71         ll w;
72         cin>>u>>v>>w;
73         add(u,v,w);
74     }
75
76     SPFA();//建边后调用即可
77
78     if(times[s]>n) //找负环
79     {
80         cout<<"minus circle!\n";
81         return;
82     }
83     for(int i=1;i<=n;i++)cout<<dis[i]<<" ";
84     cout<<"\n";
85 }

```

4.6.5 判断欧拉回路或通路.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  const ll N=250000+7;
5  int main()
6  {
7      // ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
8      void solve();
9      int t=1;
10     while(t--)solve();
11     return 0;
12 }
13 //无向图
14 //判断欧拉回路和通路
15 map<string,int> point;
16 int cnt=0,tot=0;
17 struct node{int u,v,i;};
18 vector<node> edge;
19 vector<int> e[N<<1];
20 int ind[N<<1],vis[N];
21 void yes(){cout<<"Possible\n";}
22 void no(){cout<<"Impossible\n";}
23 int dfs(int u) //深搜找欧拉回路/通路
24 {
25     int ans=0;
26     for(auto re:e[u])
27     {
28         int v=edge[re].u^edge[re].v^u;
29         if(!vis[re])continue;
30         vis[re]--;
31         ans++;
32         ans+=dfs(v);
33     }
34     return ans; //这里在数经过的边数
35     //如果要找路, 可以在这将点加入栈
36 }
37 bool check(int m)
38 {
39     for(int i=1;i<=m;i++) //建图、数度数
40     {
41         ind[edge[i].u]++; //有向图则要分出度和入度
42         ind[edge[i].v]++;
43
44         vis[i]++; //每条边只走一次
45
46         e[edge[i].u].push_back(i); //邻接表存边号
47         e[edge[i].v].push_back(i);
48     }
49
50     int st=1; //起点
51
52     int flag=3; //
53
54     for(int i=1;i<=tot;i++)
55     {
56         if(ind[i]&1) //数奇数度的点, 超过 2 个直接返回 false

```

```

57         { //有向图这里要判断出度和入度相等
58             st=i;
59             flag--;
60         }
61         if(flag==0)break;
62     }
63
64     if(!flag)return flag; //不满足充要条件——恰好两个奇数点，或者没有奇数点
65     if(dfs(st)==cnt)return flag; //满足条件，且能一笔画
66     else return 0;
67 }
68 /// 以上是核心算法////
69 void solve()
70 {
71     int op=1;
72     char c;
73     string s1,s2;
74     edge.push_back({-1,-1,0});
75     for(int i=1;i<=tot;i++)e[i].clear();
76     while(scanf("%c",&c)!=EOF)
77     {
78         if(c=='0')break;
79         if(c==' ')op^=1;
80         else if(c=='\n')
81         {
82             cnt++;
83             if(!point[s1])point[s1]=++tot;
84             if(!point[s2])point[s2]=++tot;
85             edge.push_back({point[s1],point[s2],cnt});
86             s1.clear();
87             s2.clear();
88             op^=1;
89         }
90         else if(op)
91         {
92             s1.push_back(c);
93         }
94         else
95         {
96             s2.push_back(c);
97         }
98     }
99     if(check(cnt))yes();
100    else no();
101 }

```

4.6.6 换边最短路.typ

给你一个 n 个点, m 条边的无向图, 每条边连接点 u, v , 并且有个长度 w 。有 q 次询问, 每次询问给你一对 t, x , 表示仅当前询问下, 将 t 这条边的长度修改为 x , 请你输出当前 1 到 n 的最短路长度。

数据范围

$$2 \leq n \leq 2e5 \quad 1 \leq m, q \leq 2e5 \quad 1 \leq w_i, x_i \leq 1e9$$

我们先不考虑修改, 考虑给出指定的边, 求出经过这条边的最短路 设这条边连接 u, v , 很容易想到这样的话只需要从 1 与 n 分别跑一遍 Dijkstra, 最短路长度就是 $\min(1 \text{ 到 } u \text{ 的距离} + \text{边长} + v \text{ 到 } n \text{ 的距离}, 1 \text{ 到 } v \text{ 的距离} + \text{边长} + u \text{ 到 } n \text{ 的距离})$ 。

我们可以把修改分为以下几类:

- 1. 修改的边在 1 到 n 的最短路上, 边的长度变大了。
- 2. 修改的边在 1 到 n 的最短路上, 边的长度变小了。
- 3. 修改的边不在 1 到 n 的最短路上, 边的长度变大了。
- 4. 修改的边不在 1 到 n 的最短路上, 边的长度变小了。

很容易知道: 对于 2, 原最短路长度 - 原边长 + 新边长 就是答案。对于 3, 原最短路长度 就是答案。对于 4, 由前面的思考得 $\min(\text{原最短路长度}, \min(1 \text{ 到 } u \text{ 的距离} + \text{新边长} + v \text{ 到 } n \text{ 的距离}, 1 \text{ 到 } v \text{ 的距离} + \text{新边长} + u \text{ 到 } n \text{ 的距离}))$ 就是答案。

都是 $O(1)$ 得出答案

于是只剩下 1 了。对于原问题, 我们可以得到一些简单的结论。令原最短路为 E , 其路径为 $E_1, E_2, E_3, \dots, E_k$ 。对于每个不是 E 上的点 u , 1 到 u 的最短路必定会使用 E 的一段前缀 (可以为空)。令这个前缀为 L_u , 1 到 u 的最短路经过 $E_1, E_2, \dots, E(L_u)$ 。

同理可得 u 到 n 的最短路必定会使用 E 的一段后缀 (可以为空)。令这个后缀为 R_u , u 到 n 的最短路经过 $E_1, E_2, \dots, E(R_u)$ 。

特别地, 对于 E 上的点 u , 令其 L 为 E 上连接自己的边的编号, R 为自己连到下一个 E 上点的边的编号

关于如何求出每个点的 L 与 R , 我们可以先从 n 到 1 跑一遍 Dijkstra, 求出 E 的路径。然后分别从 1 到 n , n 到 1 各跑一遍 Dijkstra, 过程中分别更新每个非 E 上点的 L, R 。

有了每个点的 L, R 后, 考虑如何解决 1。1 就是求 $\min(E \text{ 的长度} - \text{原边长} + \text{新边长}, \text{不经过修改的这条边的最短路长度})$

问题从而转化成如何快速求出 不经过 E 上某条边的最短路长度

考虑使用线段树, 树上的每段区间 l, r 的值表示 整个图不经过 E 上 l 到 r 这段的最短路长度

有了每个点的 L, R 我们很容易用图中非 E 上的边更新线段树

例如：一条边连接点 u,v ,经过这条边的最短路长度为 len ,

我们可以把树上 L_{u+1},R_v 的区间用 len 更新,比个 \min

同样地, 可以把 L_{v+1},R_u 的区间用 len 更新

由于代码中点 l 的 l,r 为 0 , 且 $l=r$,所以 l 要加 1

如果图方便, 可以把 $l[1]=1,r[1]=0$,每个点的 l 比 r 大 1

不懂的话可以自己画图比划比划

从而我们可以在 $O(\log n)$ 的时间内回答每个问题

总的复杂度为 $O((m+n+q)\log n)$

4.6.6.1 solution:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  const ll inf=1e18;
5  #define N 200505
6  #define mod 1000000007
7  int main()
8  {
9      ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
10     int t=1;
11     void solve();
12     // cin>>t;
13     while(t--)>solve();
14 }
15 //换边最短路
16 //题目：有一张  $n$  个点,  $m$  条带权无向边的图, 现在有  $qu$  次询问, 每次询问将边  $ti$  的权值修改
17 // 为  $ch$  后的最短路,
18 // 询问对原图无影响
19 // 可用于删边、换边的最短路
20
21 //思想是先跑一遍  $dj$ , 找到原始的最短路, 用线段维护路上每条边删除后的最短路的距离
22 //此模板建议原封不动地抄
23 int n,m,qu; //n 点数, m 边数, qu 询问次数
24 struct Original_edge{int u,v;ll len;}OE[N]; //存原边
25 struct tabl{int to,id;ll len;}; //用于邻接表的结构体、存边
26 struct node{int id; ll len; bool operator < (const node &x) const {return
27 x.len<len;}}; //用于优先队列的结构体, 存点和距离
28 vector<tabl> edge[N<<1]; //邻接表
29 priority_queue<node> q; //用于  $dj$  的优先队列
30 int lstvis[N]={0};
31 int l[N]={0};
32 int r[N]={0}; //
33 int ind[N]={0}; //标记, ind[i]==j, 表示边  $i$  在原始最短路, 且是路上的第  $j$  条边
34 int vis[N]; //用于  $dj$  的  $vis$  数组
35 ll t[N<<4]={0}; //这个是线段树
36

```

```

37 ll disT[N],disN[N];    //disT[i]从起点到点 i 的最短距离, disN[i]从终点到点 i 的最短
    距离
38 bool on_path[N];      //记录原始最短路, on_path[i]==true,表示点 i 在最短路上
39 int path_cnt=0;       //表示原始最短路上的边数
40
41 void dj(int p,ll dis[],int f)//起点, 对应数组, 操作编号
42 {
43     for(int i=1;i<=n;i++)//初始化
44     {
45         vis[i]=0;
46         dis[i]=inf;
47     }
48     dis[p]=0;
49     q.push((node){p,0}); //加入起点
50     while(!q.empty())
51     {
52         node temp=q.top();
53         q.pop();
54         int u=temp.id;
55         ll w=temp.len;
56         if(vis[u])continue;//跳过处理过的点
57         vis[u]++;
58         dis[u]=w;//记录距离
59         for(auto re:edge[u])
60         {
61             int v=re.to;
62             int id=re.id;
63             ll tw=re.len;
64
65             if(dis[v]<=w+tw)continue;
66             dis[v]=w+tw; //松弛
67             lstvis[v]=id; //记前驱边, 表示点 v 从边 id 到达
68             q.push((node){v,w+tw});
69
70             if(f==1&&!on_path[v])l[v]=l[u]; //操作 1, 此时是从起点出发, 需要记住
    前缀
71             if(f==2&&!on_path[v])r[v]=r[u]; //操作 2, 此时是从终点出发, 需要记住
    后缀
72         }
73     }
74 }
75 void trace()
76 {
77     int u=1; //u 初始为起点
78     on_path[u]=true; //做好起点的初始化
79     l[u]=r[u]=0;
80
81     for(int i=1;u!=n;i++) //第一次 dj 是从终点开始, 所以这里要从起点开始, 向终点找
    路
82     {
83         int e_id=lstvis[u]; //取前驱边
84         ind[e_id]=i; //给前驱标记
85
86         u^=OE[e_id].u^OE[e_id].v; //取前驱点
87         on_path[u]=true; //前驱点做标记
88         l[u]=r[u]=i; //做标记

```

```

89     path_cnt++;    //路长增加
90 }
91 }
92 void build(int le,int ri,int p)
93 {
94     t[p]=inf;    //初始化为无穷大
95     if(le==ri)return;
96     int mid=(le+ri)>>1,lef=(p<<1),rig=lef|1;
97     build(le,mid,lef);
98     build(mid+1,ri,rig);
99 }
100 void update(int L,int R,int x,int y,int p,ll k)
101 {
102     if(x>y)return;
103     if(x<=L&&R<=y)    //区间修改
104     {
105         t[p]=min(t[p],k);
106         return;
107     }
108     int mid=(L+R)>>1,lef=(p<<1),rig=lef|1;
109     if(x<=mid)update(L,mid,x,y,lef,k);
110     if(y>mid)update(mid+1,R,x,y,rig,k);
111 }
112 ll query(int L,int R,int x,int y,int p)
113 {
114     ll ans=t[p];
115     if(L==R)    //单点查询
116     {
117         return ans;
118     }
119     int mid=(L+R)>>1,lef=(p<<1),rig=lef|1;
120     if(y<=mid)ans=min(ans,query(L,mid,x,y,lef));    //全程取最小值
121     else ans=min(ans,query(mid+1,R,x,y,rig));
122     return ans;
123 }
124 void solve()
125 {
126     memset(on_path,false,sizeof(on_path));    //初始化
127
128     cin>>n>>m>>qu;
129     for(int i=1;i<=m;i++)
130     {
131         int u,v;
132         ll w;
133         cin>>u>>v>>w;
134
135         OE[i].u=u;    //记录原边, 后续要用下标查询边
136         OE[i].v=v;
137         OE[i].len=w;
138
139         edge[u].push_back({v,i,w});    //邻接表
140         edge[v].push_back({u,i,w});
141     }
142
143     dj(n,disN,0);    //先 dj 一次, 找到原始最短路
144     trace();
145     dj(1,disT,1);    //分别对起点和终点进行 dj, 得到每条边的
146     dj(n,disN,2);

```

```

147
148     build(1,path_cnt,1); //建立线段树，维护不经过某些边的最短路
149
150     for(int i=1;i<=m;i++)
151     {
152         int u=OE[i].u,v=OE[i].v;
153
154         ll w=OE[i].len;
155
156         if(ind[i])continue; //在路上的边不做更新
157
158         //为线段树加入元素
159         update(1,path_cnt,l[u]+1,r[v],1,disT[u]+w+disN[v]); //注意这里左边界要加
160
161         update(1,path_cnt,l[v]+1,r[u],1,disT[v]+w+disN[u]);
162     }
163     while(qu--)
164     {
165         int ti;
166         ll ch;
167         ll ans;
168         cin>>ti>>ch;
169         //询问会有两种大情况，换了最短路上的边，以及，换了最短路外的边
170
171         if(ind[ti]) //若换了路上的边
172         {
173             ans=disT[n]-OE[ti].len+ch; //最理想的情况就是将原最短路的边给换一下
174             //若新的边权更小，则不用比较了
175
176             if(ch>OE[ti].len) //若新的边权更大，则需要比较
177             {
178                 ans=min(ans,query(1,path_cnt,ind[ti],ind[ti],1)); //查询不经
179                 过 ti 边的最短路
180             }
181         }
182         else //若换了最短路外的边
183         {
184             ans=disT[n]; //最理想是原最短路
185             //换的边比原来的边大，则无需考虑
186
187             if(OE[ti].len>ch) //若换的边更小，则需要比较
188             {
189                 int u=OE[ti].u,v=OE[ti].v;
190
191                 ans=min(ans,min(disT[u]+ch+disN[v],disT[v]+ch+disN[u]));
192                 //新的最短路可能是 disT[u]+ch+disN[v] 表示从起点沿最优路径到达
193                 u, 再经过边 ti, 再从 v 沿着最优路径到达终点
194                 //也可能是 disT[v]+ch+disN[u] 表示从起点沿着最优路径到
195                 达 v, 再经过边 ti, 再从 u 沿着最优路径到达终点
196             }
197         }
198         cout<<ans<<"\n";
199     }
200 }

```

4.6.7 求欧拉回路或通路.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  const ll N=1055;
5  int main()
6  {
7      ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
8      void solve();
9      int t=1;
10     while(t--)solve();
11     return 0;
12 }
13 //在已知有欧拉回路或通路的情况下求欧拉回路或欧拉通路
14 int m;
15 struct node{int u,v,i;};
16 vector<node> edge;
17 vector<pair<int,int>> e[N];
18 int vis[N]={0};
19 int ind[N]={0};
20 stack<int> stk;
21 bool cmp(pair<int,int> x,pair<int,int> y)
22 {
23     int ix=x.second,iy=y.second;
24     int u=x.first,v=y.first;
25     return (edge[ix].v^edge[ix].u^u)<(edge[iy].u^edge[iy].v^v);
26 }
27 void dfs(int u)
28 {
29     for(auto re:e[u])
30     {
31         int v=edge[re.second].v^edge[re.second].u^u;
32         if(!vis[re.second])continue;
33         vis[re.second]--;
34         dfs(v);
35     }
36     stk.push(u);
37 }
38 void solve()
39 {
40     memset(vis,0,sizeof(vis));
41     int st=550;
42     cin>>m;
43     edge.clear();
44     edge.push_back({-1,-1,0});
45     for(int i=1;i<=500;i++)
46     {
47         ind[i]=0;
48         e[i].clear();
49     }
50     for(int i=1;i<=m;i++)
51     {
52         int u,v;
53         cin>>u>>v;
54         edge.push_back({u,v,i});
55         e[u].push_back({u,i}); //邻接表, 但是存编号
56         e[v].push_back({v,i});

```

```

57
58     ind[u]++; //记录顶点的度数
59     ind[v]++;
60
61     vis[i]++; //记录边的可遍历次数
62
63     st=min(st,min(u,v));
64 }
65 for(int i=1;i<=500;i++)
66 {
67     if(!e[i].size())continue;
68     sort(e[i].begin(),e[i].end(),cmp); //要求输出字典序最小的
69 }
70 for(int i=1;i<=500;i++)
71 {
72     if(ind[i]&1) //若有奇数度的点，则需要从奇数度的点开始，找欧拉通路
73     {
74         st=i;
75         break;
76     }
77 }
78 dfs(st);
79 while(!stk.empty()) //倒序输出
80 {
81     int u=stk.top();
82     stk.pop();
83     cout<<u<<"\n";
84 }
85 }

```

4.7 TREE

4.7.1 kruskal 重构树.h

```

1  /*
2  n 座城市,m 条双向边,每条路有限重,问从 x 到 y 最多能运输多重的货物
3  */
4  void solve(){
5      int n,m;
6      cin>>n>>m;
7      vector<int> e[n*2+1];
8      priority_queue<array<int,3>> q;
9      for(int i=1;i<=m;i++){
10         int u,v,w;
11         cin>>u>>v>>w;
12         q.push({w,u,v});
13     }
14     vector<int> fa(n*2+10),w(n*2+10);
15     iota(fa.begin(),fa.end(),0);
16     function<int(int)> find=[&](int x){
17         return fa[x]==x ? x : fa[x]=find(fa[x]);
18     };
19     auto merge=[&](int x,int y){
20         int fx=find(x),fy=find(y);
21         fa[fx]=fy;
22     };
23     int cnt;
24     auto Kruskal=[&]()->void {
25         cnt=n;
26         while(!q.empty()){
27             auto a=q.top();q.pop();
28             int fx=find(a[1]),fy=find(a[2]);
29             if(fx==fy) continue;
30             cnt++;
31             merge(fx,cnt);
32             merge(fy,cnt);
33             w[cnt]=a[0];
34             e[cnt].push_back(fy);
35             e[cnt].push_back(fx);
36         }
37     };
38     Kruskal();
39     vector<vector<int>> faa(n*2+10,vector<int>(19));
40     vector<int> dep(n*2+10);
41     vector<bool> vis(n*2+10);
42     function<void(int,int)> dfs=[&](int id,int u){
43         faa[id][0]=u;dep[id]=dep[u]+1;vis[id]=1;
44         for(int i=1;i<=19;i++){
45             faa[id][i]=faa[faa[id][i-1]][i-1];
46         }
47         for(auto x:e[id]) dfs(x,id);
48     };
49     for(int i=cnt;i>=1;i--){
50         if(!vis[i]) dfs(i,i);
51     }
52     auto lca=[&](int x,int y)->int{
53         if(find(x)!=find(y)) return 0;

```

```

54     if(dep[x]<dep[y]) swap(x,y);
55     int tmp=dep[x]-dep[y];
56     for(int i=0;i<19;i++){
57         if((tmp>>i&1)) x=faa[x][i];
58     }
59     if(x==y) return x;
60     for(int i=18;i>=0;i--){
61         if(faa[x][i]!=faa[y][i]){
62             x=faa[x][i];
63             y=faa[y][i];
64         }
65     }
66     return faa[x][0];
67 };
68 w[0]=-1;
69 int qq;cin>>qq;
70 while(qq--){
71     int x,y;
72     cin>>x>>y;
73     int l=lca(x,y);
74     cout<<w[l]<<"\n";
75 }
76 }

```


4.7.2 lca.h

```

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

```

```

53 };
54
55 //-----VERSION 2-----
56
57
58 void solve(){
59     int n,m,root;
60     cin>>n>>m>>root;
61     vector<int> e[n+1],dep(n+1);
62     vector<vector<int>> fa(n+1,vector<int>(21));
63     for(int i=1;i<n;i++){
64         int u,v;
65         cin>>u>>v;
66         e[u].push_back(v);
67         e[v].push_back(u);
68     }
69     function<void(int,int)> dfs=[&](int id,int u){
70         fa[id][0]=u;
71         dep[id]=dep[u]+1;
72         for(int i=1;i<=20;i++) fa[id][i]=fa[fa[id][i-1]][i-1];
73         for(auto x:e[id]){
74             if(x==u) continue;
75             dfs(x,id);
76         }
77     };
78     dfs(root,root);
79     function<int(int,int)> lca=[&](int x,int y){
80         if(dep[x]<dep[y]) swap(x,y);
81         int tmp=dep[x]-dep[y];
82         for(int i=0;i<=20;i++){
83             if(tmp>>i&1) x=fa[x][i];
84         }
85         if(x==y) return x;
86         for(int i=20;i>=0;i--){
87             if(fa[x][i]!=fa[y][i]){
88                 x=fa[x][i];
89                 y=fa[y][i];
90             }
91         }
92         return fa[x][0];
93     };
94     while(m--){
95         int a,b;
96         cin>>a>>b;
97         cout<<lca(a,b)<<"\n";
98     }
99     return;
100 }

```

4.7.3 中心.h

```

1 // 这份代码默认节点编号从 1 开始, 即  $i \in [1, n]$ , 使用 vector 存图
2 int d1[N], d2[N], up[N], x, y, mini = 1e9; // d1,d2 对应上文中的 len1,len2
3
4 struct node {
5     int to, val; // to 为边指向的节点, val 为边权
6 };
7
8 vector<node> nbr[N];
9
10 void dfsd(int cur, int fa) { // 求取 len1 和 len2
11     for (node nxt : nbr[cur]) {
12         int nxt = nxt.to, w = nxt.val; // nxt 为这条边通向的节点, val 为边权
13         if (nxt == fa) {
14             continue;
15         }
16         dfsd(nxt, cur);
17         if (d1[nxt] + w > d1[cur]) { // 可以更新最长链
18             d2[cur] = d1[cur];
19             d1[cur] = d1[nxt] + w;
20         } else if (d1[nxt] + w > d2[cur]) { // 不能更新最长链, 但可更新次长链
21             d2[cur] = d1[nxt] + w;
22         }
23     }
24 }
25
26 void dfsu(int cur, int fa) {
27     for (node nxt : nbr[cur]) {
28         int nxt = nxt.to, w = nxt.val;
29         if (nxt == fa) {
30             continue;
31         }
32         up[nxt] = up[cur] + w;
33         if (d1[nxt] + w != d1[cur]) { // 如果自己子树里的最长链不在 nxt 子树里
34             up[nxt] = max(up[nxt], d1[cur] + w);
35         } else { // 自己子树里的最长链在 nxt 子树里, 只能使用次长链
36             up[nxt] = max(up[nxt], d2[cur] + w);
37         }
38         dfsu(nxt, cur);
39     }
40 }
41
42 void GetTreeCenter() { // 统计树的中心, 记为 x 和 y (若存在)
43     dfsd(1, 0);
44     dfsu(1, 0);
45     for (int i = 1; i <= n; i++) {
46         if (max(d1[i], up[i]) < mini) { // 找到了当前 max(len1[x], up[x]) 最小点
47             mini = max(d1[i], up[i]);
48             x = i;
49             y = 0;
50         } else if (max(d1[i], up[i]) == mini) { // 另一个中心
51             y = i;
52         }
53     }
54 }

```

4.7.4 支配树.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define pb push_back
5  #define emp empty
6  #define fi first
7  #define se second
8  const int N=3e5+7;
9  const ll inf=1e16+7;
10 const ll mod=998244353;
11 signed main()
12 {
13     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
14     void solve();
15     int t=1;
16     // cin>>t;
17     while(t--)solve();
18     return 0;
19 }
20 struct node
21 {
22     int v;
23     int nex;
24 }a[N<<2];
25 int n,m,u,v,cnt=0,dfc=0;
26 int siz[N],dfn[N],pos[N],sdm[N],idm[N],fa[N],fth[N],mn[N];
27 //siz 儿子数、dfn 深搜序、pos 深搜序对应结点, sdm 半支配, idm 最近支配, fa 并查集, fth
    深搜树前驱, mn 半支配点深搜序最小的祖宗
28 int h[3][N<<1];//三幅图的表头, 分别是原图、反图、支配树图
29 void clear()
30 {
31     memset(h,0,sizeof(h));
32     memset(dfn,0,sizeof(dfn));
33     memset(siz,0,sizeof(siz));
34     cnt=0;
35     dfc=0;
36 }
37 void add(int u,int v,int x)
38 {
39     a[++cnt]=(node){v,h[x][u]};
40     h[x][u]=cnt;
41 }
42 void dfs(int u)
43 {
44     dfn[u]=++dfc;
45     pos[dfc]=u;
46     for(int i=h[0][u];i;i=a[i].nex)
47     {
48         int v=a[i].v;
49         if(dfn[v])continue;
50         dfs(v);
51         fth[v]=u;
52     }
53 }
54 int find(int u)//这是一个带权并查集?
55 {

```

```

56     if(fa[u]==u) return u;
57     int temp=fa[u];
58     fa[u]=find(fa[u]);
59     if(dfn[sdm[mn[temp]]]<dfn[sdm[mn[u]]])
60     {
61         mn[u]=mn[temp];
62     }
63     return fa[u];
64 }
65 void tar(int st)
66 {
67     dfs(st);
68     for(int i=1;i<=n;i++)
69     {
70         sdm[i]=fa[i]=mn[i]=i;
71     }
72     for(int i=dfc;i>=2;i--)
73     {
74         int u=pos[i];
75         int res=mod;
76         for(int j=h[1][u];j;j=a[j].nex)
77         {
78             int v=a[j].v;
79             if(!dfn[v]) continue;
80             find(v);
81             if(dfn[v]<dfn[u]) res=min(res,dfn[v]);
82             else res=min(res,dfn[sdm[mn[v]]]);
83         }
84         sdm[u]=pos[res];
85         fa[u]=fth[u];
86         add(sdm[u],u,2);
87         u=fth[u];
88         for(int j=h[2][u];j;j=a[j].nex)
89         {
90             int v=a[j].v;
91             find(v);
92             if(u==sdm[mn[v]])
93             {
94                 idm[v]=u;
95             }
96             else
97             {
98                 idm[v]=mn[v];
99             }
100         }
101         h[2][u]=0;
102     }
103     for(int i=2;i<=dfc;i++)
104     {
105         int u=pos[i];
106         if(idm[u]!=sdm[u]) idm[u]=idm[idm[u]];
107     }
108     for(int i=dfc;i>=2;i--)
109     {
110         int u=pos[i];
111         ++siz[u];
112         siz[idm[u]]+=siz[u];
113     }

```

```

114     ++siz[st];
115 }//这里支配树消失了，但可以通过 idm 重建
116 void solve()
117 {
118     cin>>n>>m;
119     clear();
120     for(int i=1;i<=m;i++)
121     {
122         int u,v;
123         cin>>u>>v;
124         add(u,v,0);
125         add(v,u,1);
126     }
127     tar(1);
128     for(int i=1;i<=n;i++)cout<<siz[i]<<" ";
129     cout<<"\n";
130 }

```

4.7.5 斯坦纳树.h

```

1  /*
2  n(<=100)个点 m(<=500)条带权无向边 G
3  给定 k (<=10) 个节点的点集 S, 选出 G 的子图 G1
4  使得 S 属于 G1, G1 是联通图 边权和最小
5
6  */
7
8  #include<bits/stdc++.h>
9  using namespace std;
10 #define int long long
11 const int N=4e3;
12 const int inf=2e9;
13 int n,m,k,p[N],state;
14 int dp[N][N];
15 int head[N],to[N],ne[N],w[N],tot;
16 void add(int x,int y,int z){
17     ne[++tot]=head[x];
18     to[tot]=y,w[tot]=z;
19     head[x]=tot;
20 }
21 queue<int>q;
22 bool vis[N];
23 void spfa(int s){
24     while(!q.empty()){
25         int u=q.front();q.pop();
26         vis[u]=0;
27         for(int i=head[u];i;i=ne[i]){
28             int v=to[i];
29             if(dp[v][s]>dp[u][s]+w[i]){
30                 dp[v][s]=dp[u][s]+w[i];
31                 if(!vis[v]) q.push(v),vis[v]=1;
32             }
33         }
34     }
35 }
36 signed main(){
37     cin>>n>>m>>k;
38     for(int i=1;i<=m;i++){
39         int x,y,z;cin>>x>>y>>z;
40         add(x,y,z),add(y,x,z);
41     }
42     state=(1<<k)-1;
43     for(int i=1;i<=n;i++) for(int s=0;s<=state;s++) dp[i][s]=inf;
44     for(int i=1;i<=k;i++){
45         cin>>p[i];
46         dp[p[i]][1<<(i-1)]=0;
47     }
48     for(int s1=1;s1<=state;s1++){
49         for(int i=1;i<=n;i++){
50             for(int s2=s1&(s1-1);s2;s2=s1&(s2-1)) dp[i][s1]=min(dp[i][s1],dp[i]
51 [s2]+dp[i][s1^s2]); //枚举子集
52             if(dp[i][s1]<inf) q.push(i),vis[i]=1; //将这个点看成出发点
53         }
54         spfa(s1);
55     }
56     cout<<dp[p[1]][state]; //此时以哪个关键点为根都无所谓, 答案是一样的

```

56 }

4.7.6 直径.h

```

1  vector<int> dep(n+1);
2  int mx=0,tmp=-1;
3  function<void(int,int)> dfs=[&](int id,int fa){
4      dep[id]=dep[fa]+1;
5      for(auto x:e[id]){
6          if(x==fa) continue;
7          dfs(x,id);
8      }
9      if(dep[id]>mx){
10         mx=dep[id];
11         tmp=id;
12     }
13 };
14 dfs(1,0);
15 int d1=tmp;
16 mx=0;
17 dfs(d1,0);
18 int d2=tmp; //d1,d2

```

4.7.7 虚树.h

```

1  int dfn[N];
2  int h[N],m,a[N],len;
3  bool cmp(int x,int y){
4      return dfn[x]<dfn[y];
5  }
6  void build(){
7      sort(h+1,h+1+m,cmp); //dfn 排序
8      for(int i=1;i<m;i++){
9          a[++len]=h[i];
10         a[++len]=lca(h[i],h[i+1]); //插入 LCA
11     }
12     a[++len]=h[m];
13     sort(a+1,a+1+len,cmp); //DFN 排序
14     len=unique(a+1,a+1+len)-a+1; //去重
15     for(int i=1,lc;i<len;i++){
16         lc=lca(a[i],a[i+1]);
17         conn(lc,a[i+1]); //连边
18     }
19 }

```


4.7.9 重链剖分.h

```

1  /*
2  n 个节点树，节点有权值，q 次询问，四种操作
3  op1: x 到 y 最短路径上所有节点的值加上 val
4  op2: 输出 x 到 y 最短路径上所有节点的权值和
5  op3: 将以 x 为根节点的子树内所有节点值都加上 val
6  op4: 输出以 x 为根节点的子树内所有节点的权值和
7  op5: 输入 x,y 输出 lca(x,y)
8  */
9  const int N=5e5+10;
10 vector<int> e[N];
11 //如果开 vector 有爆空间的可能
12 int fa[N],son[N],sz[N],dep[N],dfn[N],rkn[N],top[N];
13 int fi[N],la[N];
14 int idx=0;
15 void dfs1(int id,int u){
16     fa[id]=u;
17     sz[id]=1;
18     dep[id]=dep[u]+1;
19     for(auto x:e[id]){
20         if(x==u) continue;
21         dfs1(x,id);
22         sz[id]+=sz[x];
23         if(sz[x]>sz[son[id]]) son[id]=x;
24     }
25 }
26 void dfs2(int id,int tp){
27     top[id]=tp;
28     dfn[id]=++idx;
29     fi[id]=la[id]=idx;
30     rkn[idx]=id;
31     if(son[id]) dfs2(son[id],tp);
32     for(auto x:e[id]){
33         if(x==fa[id]||x==son[id]) continue;
34         dfs2(x,x);
35     }
36     for(auto x:e[id]){
37         if(x==fa[id]) continue;
38         la[id]=max(la[id],la[x]);
39     }
40 }
41 vector<int> a(N);
42 #define lc p<<1
43 #define rc p<<1|1
44 struct node{
45     int l,r,sum,mx;
46     int lz;
47 }tr[4*N];
48 void pushup(int p){
49     tr[p].sum=tr[lc].sum+tr[rc].sum;
50     tr[p].mx=max(tr[lc].mx,tr[rc].mx);
51 }
52 void build(int p,int l,int r){ // (1,1,n)
53     tr[p].l=l;tr[p].r=r;tr[p].lz=0;
54     if(l==r){
55         tr[p].sum=tr[p].mx=a[rkn[l]];

```

```

56     return;
57 }
58 int m=l+r>>1;
59 build(lc,l,m);
60 build(rc,m+1,r);
61 pushup(p);
62 }
63 void pushdown(int p){
64     if(tr[p].lz){
65         tr[lc].mx+=tr[p].lz;
66         tr[rc].mx+=tr[p].lz;
67         tr[lc].sum+=(tr[lc].r-tr[lc].l+1)*tr[p].lz;
68         tr[rc].sum+=(tr[rc].r-tr[rc].l+1)*tr[p].lz;
69         tr[lc].lz+=tr[p].lz;
70         tr[rc].lz+=tr[p].lz;
71         tr[p].lz=0;
72     }
73 }
74 void update1(int p,int x,int val){ //单点修改 传(1, dfn[id],val)
75     if(tr[p].l==tr[p].r){
76         // tr[p].mx=tr[p].sum=val;
77         tr[p].mx+=val;
78         tr[p].sum+=(tr[p].r-tr[p].l+1)*val;
79         return;
80     }
81     pushdown(p);
82     int m=tr[p].l+tr[p].r>>1;
83     if(x<=m) update1(lc,x,val);
84     else update1(rc,x,val);
85     pushup(p);
86 }
87 void update2(int p,int l,int r,int val){ //区间修改 传(1,dfn[x],dfn[y],val)
88     if(l<=tr[p].l&&tr[p].r<=r){
89         tr[p].sum+=(tr[p].r-tr[p].l+1)*val;
90         tr[p].mx+=val;
91         tr[p].lz+=val;
92         return;
93     }
94     pushdown(p);
95     int m=tr[p].l+tr[p].r>>1;
96     if(l<=m) update2(lc,l,r,val);
97     if(m<r) update2(rc,l,r,val);
98     pushup(p);
99 }
100 int query1(int p,int l,int r){ // 查询 sum 传(1,dfn[x],dfn[y])
101     if(l<=tr[p].l&&tr[p].r<=r) return tr[p].sum;
102     pushdown(p);
103     int sum=0;
104     int m=tr[p].l+tr[p].r>>1;
105     if(l<=m) sum+=query1(lc,l,r);
106     if(m<r) sum+=query1(rc,l,r);
107     return sum;
108 }
109 int query2(int p,int l,int r){ //查询 mx 传(1,dfn[x],dfn[y])
110     if(l<=tr[p].l&&tr[p].r<=r) return tr[p].mx;
111     pushdown(p);
112     int mx=-1e9;

```

```

113     int m=tr[p].l+tr[p].r>>1;
114     if(l<=m) mx=max(mx,query2(lc,l,r));
115     if(m<r) mx=max(mx,query2(rc,l,r));
116     return mx;
117 }
118 int query_sum(int x,int y){ //查询两点路径的 sum 传(x,y)
119     int ans=0,fx=top[x],fy=top[y];
120     while(fx!=fy){
121         if(dep[fx]>=dep[fy]) ans+=query1(1,dfn[fx],dfn[x]),x=fa[fx];
122         else ans+=query1(1,dfn[fy],dfn[y]),y=fa[fy];
123         fx=top[x];fy=top[y];
124     }
125     if(dfn[x]<dfn[y]) ans+=query1(1,dfn[x],dfn[y]);
126     else ans+=query1(1,dfn[y],dfn[x]);
127     return ans;
128 }
129 int query_mx(int x,int y){ //查询两点路径的 mx 传(x,y)
130     int ans=-1e9,fx=top[x],fy=top[y];
131     while(fx!=fy){
132         if(dep[fx]>=dep[fy]) ans=max(ans,query2(1,dfn[fx],dfn[x])),x=fa[fx];
133         else ans=max(ans,query2(1,dfn[fy],dfn[y])),y=fa[fy];
134         fx=top[x];fy=top[y];
135     }
136     if(dfn[x]<dfn[y]) ans=max(ans,query2(1,dfn[x],dfn[y]));
137     else ans=max(ans,query2(1,dfn[y],dfn[x]));
138     return ans;
139 }
140 void update3(int x,int y,int val){ //区间更新两点路径的值 传(x,y)
141     int fx=top[x],fy=top[y];
142     while(fx!=fy){
143         if(dep[fx]>=dep[fy]) update2(1,dfn[fx],dfn[x],val),x=fa[fx];
144         else update2(1,dfn[fy],dfn[y],val),y=fa[fy];
145         fx=top[x];fy=top[y];
146     }
147     if(dfn[x]<dfn[y]) update2(1,dfn[x],dfn[y],val);
148     else update2(1,dfn[y],dfn[x],val);
149 }
150 int lca(int x,int y){ //最近公共祖先
151     while(top[x]!=top[y]){
152         if(dep[top[x]]>dep[top[y]]) x=fa[top[x]];
153         else y=fa[top[y]];
154     }
155     return dep[x]>dep[y] ? y : x;
156 }
157 void solve() {
158     int n,q,root;
159     cin>>n>>q>>root;
160     for(int i=1;i<=n;i++) cin>>a[i];
161     for(int i=1;i<n;i++){
162         int u,v;
163         cin>>u>>v;
164         e[u].push_back(v);
165         e[v].push_back(u);
166     }
167     dfs1(root,0);
168     dfs2(root,root);
169     build(1,1,n);

```

```

170 while(q--){
171     int op;cin>>op;
172     if(op==1){
173         int x,y,val;
174         cin>>x>>y>>val;
175         update3(x,y,val);
176     }
177     else if(op==2){
178         int x,y;
179         cin>>x>>y;
180         cout<<query_sum(x,y)%mod<<"\n";
181     }
182     else if(op==3){
183         int x,val;
184         cin>>x>>val;
185         update2(1,fi[x],la[x],val);
186     }
187     else if(op==4){
188         int x;
189         cin>>x;
190         cout<<query1(1,fi[x],la[x])%mod<<"\n";
191     }
192     else if(op==5){
193         int x,y;
194         cin>>x>>y;
195         cout<<lca(x,y)<<"\n";
196     }else cout<<"FUCK YOU!\n";
197 }
198 }

```

5 MATH

5.1 LINNER

5.1.1 basis.h

```

1  #include <algorithm>
2  #include <iostream>
3  using ull = unsigned long long;
4
5  ull p[64];
6
7  void insert(ull x) {
8      for (int i = 63; ~i; --i) {
9          if (!(x >> i)) // x 的第 i 位是 0
10             continue;
11         if (!p[i]) {
12             p[i] = x;
13             break;
14         }
15         x ^= p[i];
16     }
17 }
18
19 using std::cin;
20 using std::cout;
21
22 int main() {
23     int n;
24     cin >> n;
25     ull a;
26     for (int i = 1; i <= n; ++i) {
27         cin >> a;
28         insert(a);
29     }
30     ull ans = 0;
31     for (int i = 63; ~i; --i) {
32         ans = std::max(ans, ans ^ p[i]);
33     }
34     cout << ans << '\n';
35     return 0;
36 }
37 /// =====VERSION 2=====
38 #include <iostream>
39 using ull = unsigned long long;
40 constexpr int MAXN = 1e5 + 5;
41
42 ull deg(ull num, int deg) { return num & (1ull << deg); }
43
44 ull a[MAXN];
45 using std::cin;
46 using std::cout;
47
48 int main() {
49     cin.tie(nullptr)->sync_with_stdio(false);
50     int n;
51     cin >> n;
52     for (int i = 1; i <= n; ++i) cin >> a[i];

```



```

53 int row = 1;
54 for (int col = 63; ~col && row <= n; --col) {
55     for (int i = row; i <= n; ++i) {
56         if (deg(a[i], col)) {
57             std::swap(a[row], a[i]);
58             break;
59         }
60     }
61     if (!deg(a[row], col)) continue;
62     for (int i = 1; i <= n; ++i) {
63         if (i == row) continue;
64         if (deg(a[i], col)) {
65             a[i] ^= a[row];
66         }
67     }
68     ++row;
69 }
70 ull ans = 0;
71 for (int i = 1; i < row; ++i) {
72     ans ^= a[i];
73 }
74 cout << ans << '\n';
75 return 0;
76 }

```

5.1.2 det.h

```

1  constexpr double EPS = 1E-9;
2  int n;
3  vector<vector<double>> a(n, vector<double>(n));
4
5  double det = 1;
6  for (int i = 0; i < n; ++i) {
7      int k = i;
8      for (int j = i + 1; j < n; ++j)
9          if (abs(a[j][i]) > abs(a[k][i])) k = j;
10     if (abs(a[k][i]) < EPS) {
11         det = 0;
12         break;
13     }
14     swap(a[i], a[k]);
15     if (i != k) det = -det;
16     det *= a[i][i];
17     for (int j = i + 1; j < n; ++j) a[i][j] /= a[i][i];
18     for (int j = 0; j < n; ++j)
19         if (j != i && abs(a[j][i]) > EPS)
20             for (int k = i + 1; k < n; ++k) a[j][k] -= a[i][k] * a[j][i];
21 }
22
23 cout << det;

```

5.1.3 GaussianELI.h

```

1  using ld = long double;
2  // A:[NxN] b[1xN]
3  //返回: {1,[]}代表有唯一解, {-1,[]}代表无解或无穷解
4  pair<int,vector<ld>> GaussianELI(vector<vector<ld>> a,vector<ld> b) {
5      assert(a.size()),assert(a.size() == a[0].size()),assert(a.size() ==
6      b.size());
7      int n = a.size();
8      vector<int> p(n);
9      iota(all(p),0);
10     for(int i = 0;i < n;i ++){
11         sort(p.begin()+i,p.end(),[&](int x,int y) -> bool {return abs(a[x][i])
12         > abs(a[y][i]);});
13         if(a[p[i]][i] == 0) continue;
14         for(int j = i+1;j < n;j ++){
15             ld k = a[p[j]][i]/a[p[i]][i] ;
16             for(int f = i+1;f < n;f ++){
17                 a[p[j]][f] -= a[p[i]][f] * k;
18                 b[p[j]] -= b[p[i]] * k;
19             }
20         }
21         vector<ld> res(n);
22         for(int i = n-1;i >= 0;i --){
23             ld dev = 0;
24             for(int f = i+1;f < n;f ++){
25                 dev += a[p[i]][f] * res[f];
26                 if(fabs(a[p[i]][i]) <= 1e-8){ return {-1,{}};} else res[i] = (b[p[i]]
27                 - dev)/a[p[i]][i];
28             }
29             return {f,res};
30         }
31     }
32 }
    
```

5.1.4 单纯形法.cpp

```

1  #include <cmath>
2  #include <cstring>
3  #include <iostream>
4  using namespace std;
5  constexpr int M = 10005, N = 1005, INF = 1e9;
6
7  int n, m;
8  double a[M][N], b[M], c[N], v;
9
10 void pivot(int l, int e) { // 转轴操作函数
11     b[l] /= a[l][e];
12     for (int j = 1; j <= n; j++)
13         if (j != e) a[l][j] /= a[l][e];
14     a[l][e] = 1 / a[l][e];
15
16     for (int i = 1; i <= m; i++)
17         if (i != l && fabs(a[i][e]) > 0) {
18             b[i] -= a[i][e] * b[l];
19             for (int j = 1; j <= n; j++)
20                 if (j != e) a[i][j] -= a[i][e] * a[l][j];
21             a[i][e] = -a[i][e] * a[l][e];
22         }
23
24     v += c[e] * b[l];
25     for (int j = 1; j <= n; j++)
26         if (j != e) c[j] -= c[e] * a[l][j];
27     c[e] = -c[e] * a[l][e];
28
29     // swap(B[l],N[e])
30 }
31
32 double simplex() {
33     while (true) {
34         int e = 0, l = 0;
35         for (e = 1; e <= n; e++)
36             if (c[e] > (double)0) break;
37         if (e == n + 1) return v; // 此时 v 即为最优解
38         double mn = INF;
39         for (int i = 1; i <= m; i++) {
40             if (a[i][e] > (double)0 && mn > b[i] / a[i][e]) {
41                 mn = b[i] / a[i][e]; // 找对这个 e 限制最紧的 l
42                 l = i;
43             }
44         }
45         if (mn == INF) return INF; // unbounded
46         pivot(l, e); // 转动 l,e
47     }
48 }
49
50 int main() {
51     cin.tie(nullptr)->sync_with_stdio(false);
52     cin >> n >> m;
53     for (int i = 1; i <= n; i++) cin >> c[i];
54     for (int i = 1; i <= m; i++) {
55         int s, t;
56         cin >> s >> t;

```

```

57         for (int j = s; j <= t; j++) a[i][j] = 1; // 表示第 i 种志愿者在 j 时间可以服
    务
58         cin >> b[i];
59     }
60     cout << (int)(simplex() + 0.5);
61 }

```

5.2 NUMBER_THEORY

5.2.1 basic.h

```

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

```

5.2.2 CRT.h

```

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

```

5.2.3 Euler_phi.h

```

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


5.2.4 Euler_sieve.h

```

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

5.2.5 factor_pr.h

```

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

```

```

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

```

```

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

```

```
173     }  
174 };
```

5.2.6 factror_pri.h

```

1  int n, m, p, b[1000005], prime[100005], t, min_prime[1000005];
2  void euler_Prime(int n)
3  { // 用欧拉筛求出 1~n 中每个数的最小质因数的编号是多少, 保存在 min_prime 中
4      for (int i = 2; i <= n; i++)
5      {
6          if (b[i] == 0)
7          {
8              prime[++t] = i;
9              min_prime[i] = t;
10         }
11         for (int j = 1; j <= t && i * prime[j] <= n; j++)
12         {
13             b[prime[j] * i] = 1;
14             min_prime[prime[j] * i] = j;
15             if (i % prime[j] == 0)
16                 break;
17         }
18     }
19 }
20 long long c(int n, int m, int p)
21 { // 计算 C(n,m)%p 的值
22     euler_Prime(n);
23     int a[t + 5]; // t 代表 1~n 中质数的个数, a[i] 代表编号为 i 的质数在答案中出现的
    次数
24     for (int i = 1; i <= t; i++)
25         a[i] = 0; // 注意清 0, 一开始是随机数
26     for (int i = n; i >= n - m + 1; i--)
27     { // 处理分子
28         int x = i;
29         while (x != 1)
30         {
31             a[min_prime[x]]++; // 注意 min_prime 中保存的是这个数的最小质因数的编
    号 (1~t)
32             x /= prime[min_prime[x]];
33         }
34     }
35     for (int i = 1; i <= m; i++)
36     { // 处理分母
37         int x = i;
38         while (x != 1)
39         {
40             a[min_prime[x]]--;
41             x /= prime[min_prime[x]];
42         }
43     }
44     long long ans = 1;
45     for (int i = 1; i <= t; i++)
46     { // 枚举质数的编号, 看它出现了几次
47         while (a[i] > 0)
48         {
49             ans = ans * prime[i] % p;
50             a[i]--;
51         }
52     }
53     return ans;

```

```
54 }  
55 int main()  
56 {  
57     cin >> n >> m;  
58     m = min(m, n - m); // 小优化  
59     cout << c(n, m, MOD);  
60 }
```

5.2.7 整除分块.typ

过程 ¶

数论分块的过程大概如下：考虑和式

$$\sum_{i=1}^n f(i) \left\lfloor \frac{n}{i} \right\rfloor$$

那么由于我们可以知道 $\left\lfloor \frac{n}{i} \right\rfloor$ 的值成一个块状分布（就是同样的值都聚集在连续的块中），那么就可以用数论分块加速计算，降低时间复杂度。

利用上述结论，我们先求出 $f(i)$ 的 **前缀和**（记作 $s(i) = \sum_{j=1}^i f(j)$ ），然后每次以 $[l, r] = [l, \left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor]$ 为一块，分块求出贡献累加到结果中即可。

伪代码如下：

```

1  Calculate  $s(i)$ , the prefix sum of  $f(i)$ .
2   $l \leftarrow 1$ 
3   $r \leftarrow 0$ 
4   $result \leftarrow 0$ 
5  while  $l \leq n$  do :
6       $r \leftarrow \left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor$ 
7       $result \leftarrow result + [s(r) - s(l-1)] \times \left\lfloor \frac{n}{l} \right\rfloor$ 
8       $l \leftarrow r + 1$ 
9  end while
```

最终得到的 $result$ 即为所求的和式。

Figure 1: 整除分块.png

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define inf 1000000000000000
5  const int N=2e6+7;
6  //题目，在  $1 \leq a < b \leq n$  的条件下，求  $\gcd(a,b)$  的和
7
8  //这里用到了欧拉函数
9  //欧拉函数也可用欧拉筛求出
10 int main()
11 {
12     void solve();
13     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
14     solve();
15     return 0;
16 }
17 bool isprime[N];
```



```

18 vector<ll> p;
19 ll phi[N]={0,1}; //边界条件
20 void eular(int n)
21 {
22     for(int i=2;i<=n;i++)
23     {
24         if(!isprime[i])
25         {
26             p.push_back(i);
27             phi[i]=i-1;
28         }
29         for(auto re:p)
30         {
31             if(i*re>n)break;
32             isprime[i*re]=1;
33             if(i%re==0){phi[i*re]=phi[i]*re;break;}
34             phi[i*re]=phi[i]*(re-1);
35         }
36     }
37     //到此，欧拉函数就求出来了
38     for(int i=1;i<=n;i++)phi[i]+=phi[i-1]; //此处是在求函数的前缀和，用于整除分块
39 }
40
41
42 ll cal(ll n,ll m)
43 {
44     ll l=1,r=0,ans=0;
45     while(l<=n)
46     {
47         r=min((n/(n/l)),(m/(m/l)));
48         ans+=(phi[r]-phi[l-1])*(n/l)*(m/l);
49         l=r+1;
50     }
51     return ans;
52 }
53 void solve()
54 {
55     ll n;
56     cin>>n;
57     eular(n);
58     ll ans=(cal(n,n)-n*(n+1)/2)/2;
59     cout<<ans<<"\n";
60 }

```

5.2.8 组合数.h

```

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

```

5.2.9 莫反.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  const int N=5e4+7;
5  //题目：在  $1 \leq a \leq n, 1 \leq b \leq m$  的条件下，求满足  $\gcd(a,b)=d$  的  $(a,b)$  的个数
6
7  //莫比乌斯反演
8  //莫比乌斯函数可以用欧拉筛求出
9  bool isprime[N];
10 vector<ll> prime;
11 int mu[N];
12 void eular()
13 {
14     mu[1]=1; //边界条件
15     for(int i=2; i<N; i++)
16     {
17         if(!isprime[i])
18         {
19             prime.push_back(i);
20             mu[i]=-1;
21         }
22         for(auto re:prime)
23         {
24             if(i*re>=N) break;
25             isprime[i*re]=1;
26             if(i%re==0)
27             {
28                 mu[i*re]=0;
29                 break;
30             }
31             mu[i*re]=-mu[i];
32         }
33     }
34     //到此，莫比乌斯函数已经求出
35
36     for(int i=1; i<N; i++) mu[i]+=mu[i-1]; //这里在求函数的前缀和，用于整除分块
37 }
38 int main()
39 {
40     ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
41     int t=1;
42     cin>>t;
43     eular();
44     void solve();
45     while(t-->0) solve();
46     return 0;
47 }
48 ll cal(int n,int m,int d) //n,m,是两个范围的上下界，d为题目所求的  $\gcd(a,b)=d$ 
49 {
50     ll ans=0;
51     n/=d;
52     m/=d;
53     for(int l=1, r=0; l<=min(n,m);)
54     {
55         r=min(min(n/(n/l), m/(m/l)), min(n,m));
56         ans+=(ll)(n/l)*(m/l)*(mu[r]-mu[l-1]);
57     }
58 }

```

```

57      //用莫比乌斯函数解题常常要求出形如： 个数*个数*mu[i] 的公式
58      l=r+1;
59  }
60  return ans;
61  }
62  void solve()
63  {
64      int n,m,d;
65      cin>>n>>m>>d;
66      cout<<cal(n,m,d)<<"\n";
67  }

```

5.3 OTHER

5.3.1 Frac.h

```

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

```

```

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

```

5.4 POLYNOMIAL

前置知识：多项式求逆 + NTT

$$\text{设 } H^2(x) \equiv F(x)(\text{mod } x^{\lceil \frac{n}{2} \rceil})$$

$$G(x) \equiv H(x)(\text{mod } x^{\lceil \frac{n}{2} \rceil})$$

$$G(x) - H(x) \equiv 0(\text{mod } x^{\lceil \frac{n}{2} \rceil})$$

$$(G(x) - H(x))^2 \equiv 0(\text{mod } x^{\lceil \frac{n}{2} \rceil})$$

$$G^2(x) - 2H(x) * G(x) + H^2(x) \equiv 0(\text{mod } x^n)$$

$$F(x) - 2H(x) * G(x) + H^2(x) \equiv 0(\text{mod } x^n)$$

$$G(x) = \frac{F(x) + H^2(x)}{2H(x)}$$

对上述式子进行多项式求逆 + NTT 即可

Figure 2: 多项式开根.png

求

$$F(x)G(x) \equiv 1 \pmod{x^n}$$

令

$$F(x)G_1(x) \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

作差可得

$$F(x)(G(x) - G_1(x)) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

除去 $F(x)$,得

$$G(x) - G_1(x) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$$

两边平方, 得

$$(G(x) - G_1(x))^2 \equiv 0 \pmod{x^n}$$

展开有

$$G^2(x) - 2G(x)G_1(x) + G_1^2(x) \equiv 0 \pmod{x^n}$$

同时乘上 $F(x)$, 由定义, 有

$$G(x) - 2G_1(x) + F(x)G_1^2(x) \equiv 0 \pmod{x^n}$$

求得递归式

$$G(x) \equiv 2G_1(x) - F(x)G_1^2(x) \pmod{x^n}$$

再提取一下公因式

$$G(x) \equiv G_1(x)(2 - F(x)G_1(x)) \pmod{x^n}$$

Figure 3: 多项式求逆元的公式及推导.png

**题目大意: **给定长度为 $n - 1$ 的数组 $g_{[1,n]}$, 求 $f_{[0,n]}$, 要求:

$$f_i = \sum_{j=1}^i f_{i-j} g_j$$

$$f_0 = 1$$

**题解: **直接求复杂度是 $O(n^2)$, 明显不可以通过此题

分治FFT, 可以用CDQ分治, 先求出 $f_{[l,mid]}$, 可以发现这部分对区间的 $f_{[mid,r]}$ 的贡献是 $f_{[l,mid]} * g_{[0,r-l]}$, 卷出来加到对应位置就行了, 复杂度 $O(n \log^2 n)$

Figure 4: 分治 fft.png

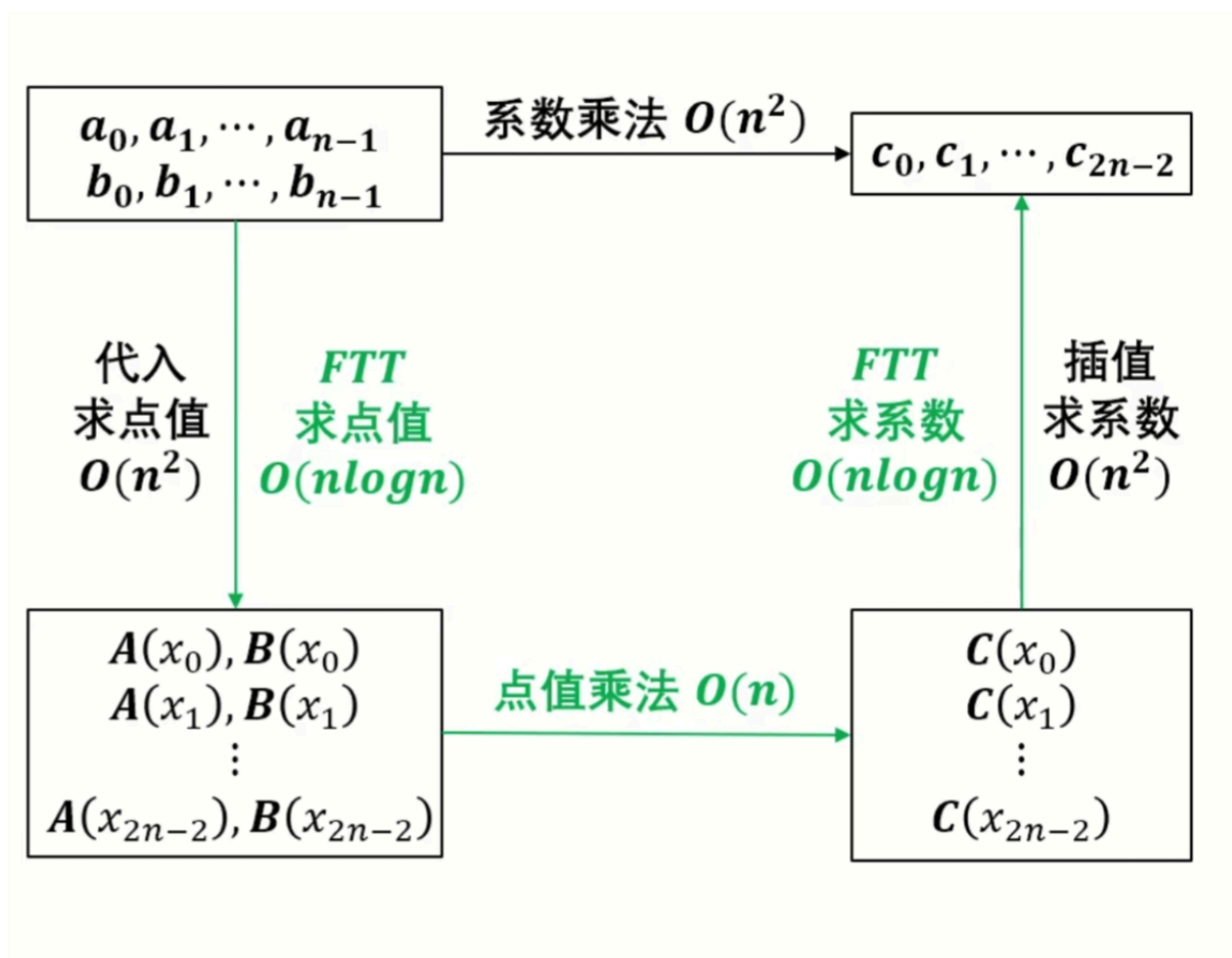


Figure 5: FFT 思想图解.png

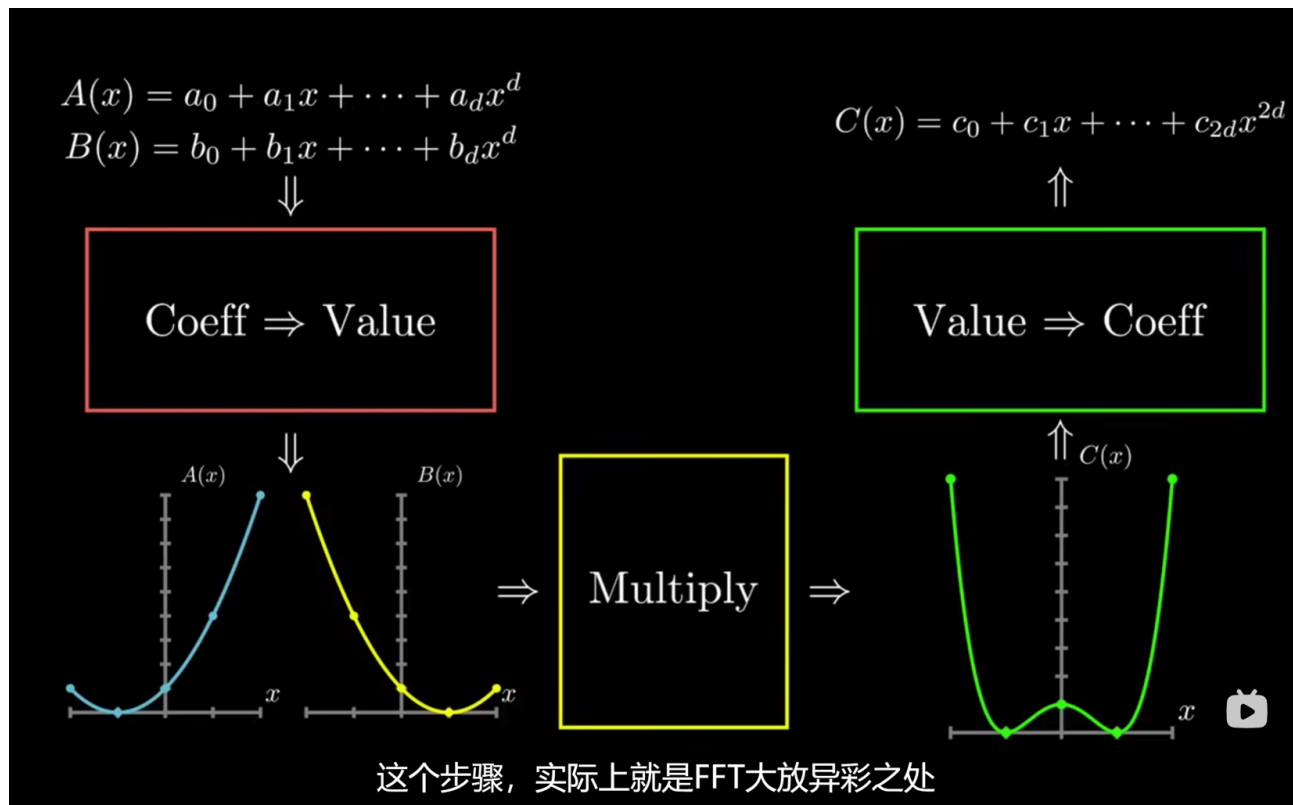


Figure 6: FFT 思想图解 2.png

5.4.1 AxB_prob.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define ld double
5  #define inf 100000000
6  #define mod 998244353
7  const ld pi=acos(-1);
8  signed main()
9  {
10     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
11     void solve();
12     solve();
13     return 0;
14 }
15 //高精乘法
16 //但是 FFT
17 //注意只能 FFT, 不能 NTT,NTT 会乱
18 //只需将每一个位置上的数视作系数, 对应一个 10 的 k 次方即可
19 const int N=(1<<22)|10;
20 string s1,s2;
21 int n,m;
22 int rev[N]={0};
23 struct comp //手搓复数
24 {
25     ld r=0,i=0;
26 }a[N],b[N];
27 ll c[N];
28 comp mul(comp x,comp y)
29 {
30     comp res;
31     res.r=x.r*y.r-x.i*y.i;
32     res.i=x.r*y.i+x.i*y.r;
33     return res;
34 }
35 comp add(comp x,comp y,int op)
36 {
37     comp res;
38     res.r=x.r+op*y.r;
39     res.i=x.i+op*y.i;
40     return res;
41 }
42
43 void chang(comp *A,int n)
44 {
45     for(int i=0;i<n;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
46     for(int i=0;i<n;i++)if(i<rev[i])swap(A[i],A[rev[i]]);
47 }
48 void FFT(comp *A,int n,int op)
49 {
50     chang(A,n);
51     for(int mid=1;mid<n;mid<<=1)
52     {
53         comp g1;
54         g1.r=cos(pi/(ld)mid);
55         g1.i=sin(op*pi/(ld)mid);
56         for(int j=0;j<n;j+=(mid<<1))

```

```

57     {
58         comp gk;
59         gk.r=1;gk.i=0;
60         for(int k=0;k<mid;k++,gk=mul(gk,g1))
61         {
62             comp x=A[j+k],y=mul(A[j+k+mid],gk);
63             A[j+k]=add(x,y,1);
64             A[j+k+mid]=add(x,y,-1);
65         }
66     }
67 }
68 if(op==1)return ;
69 for(int i=0;i<n;i++)c[i]=(ll)(A[i].r/n+0.5);
70 }
71
72 void solve()
73 {
74     stack<int> ans;
75     cin>>s1;
76     cin>>s2;
77     n=s1.length();
78     m=s2.length();
79     for(int i=0;i<n;i++)a[n-1-i].r=s1[i]-'0',a[n-1-i].i=0;
80     for(int i=0;i<m;i++)b[m-1-i].r=s2[i]-'0',b[n-1-i].i=0;
81     int mx=1;
82     while(mx<n+m)mx<<=1;
83     for(int i=n;i<mx;i++)a[i].i=0,a[i].r=0;
84     for(int i=m;i<mx;i++)b[i].i=0,b[i].r=0;
85     FFT(a,mx,1);FFT(b,mx,1);
86     for(int i=0;i<mx;i++)a[i]=mul(a[i],b[i]);
87     FFT(a,mx,-1);
88     for(int i=0;i<mx;i++)
89     {
90         if(c[i]>=10)
91         {
92             c[i+1]+=(ll)c[i]/10;
93             c[i]%=10;
94         }
95         ans.push(c[i]);
96     }
97     bool flag=true;
98     while(!ans.empty())
99     {
100         int temp=ans.top();
101         ans.pop();
102         if(temp==0&&flag)continue;
103         flag=false;
104         cout<<temp;
105     }
106     cout<<"\n";
107 }

```

5.4.2 CDQ+NTT_FTT.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define inf 0x3f3f3f3f
5  #define mod 998244353
6  const int N=1<<22;
7  ll ge=3,gi;
8  int main()
9  {
10     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
11     int t =1;
12     void solve();
13     // cin>>t;
14     while(t--)<math>\text{solve}();</math>
15 }
16
17 //CDQ+NTT/FFT
18 //可用于处理卷积
19
20 //若我们求得了左区间，即可求出左区间对于右区间的贡献
21 //而这个贡献可通过 NTT 求出
22 //这就是所谓的分治 FFT（用 NTT 精度更高）
23 ll f[N],g[N];
24 int rev[N]={0};
25 ll ksm(ll a,ll b)
26 {
27     ll ans=1;
28     while(b)
29     {
30         if(b&1)ans=ans*a%mod;
31         a=a*a%mod;
32         b>>=1;
33     }
34     return ans;
35 }
36 void chang(ll *A,int n)
37 {
38     for(int i=0;i<n;i++)
39     {
40         rev[i]=rev[i>>1]>>1;
41         rev[i]|=(i&1)?(n>>1):0;
42     }
43     for(int i=0;i<n;i++)
44     {
45         if(i<rev[i])swap(A[i],A[rev[i]]);
46     }
47 }
48 void NTT(ll *A,int n,int opt)
49 {
50     chang(A,n);
51     for(int mid=1;mid<n;mid<=<1)
52     {
53         ll g1=ksm((opt==1)?ge:gi,(mod-1)/(mid<1));
54         for(int R=mid<1,j=0;j<n;j+=R)
55         {
56             ll gk=1;

```

```

57         for(int k=0;k<mid;k++,gk=gk*g1%mod)
58         {
59             ll x=A[j+k],y=A[j+k+mid]*gk%mod;
60             A[j+k]=(x+y)%mod;
61             A[j+k+mid]=(x-y+mod)%mod;
62         }
63     }
64 }
65 }
66 void mul(ll *A,ll *B,ll n) //用于处理多项式乘法, A、B 均为系数式, 且 A 为返回的系数式
67 {
68     NTT(A,n,1);NTT(B,n,1);
69     for(int i=0;i<n;i++)A[i]=A[i]*B[i]%mod;
70     NTT(A,n,-1);
71     ll inv=ksm(n,mod-2);
72     for(int i=0;i<n;i++)A[i]=A[i]*inv%mod;
73 }
74 ll ta[N],tb[N];
75 void CDQ(int l,int r)
76 {
77     int mid=(l+r)>>1;
78     if(l==r)return ;
79     CDQ(l,mid);
80     ll mx=1;
81     while(mx<(mid-l+r-1)+1)mx<<=1; //取不小于区间长度的 二的幂
82     for(int i=0;i<mx;i++)ta[i]=tb[i]=0; //初始化
83     for(int i=l;i<=mid;i++)ta[i-l]=f[i];
84     for(int i=1;i<=r-l;i++)tb[i]=g[i];
85     mul(ta,tb,mx); //ta 乘 tb
86     for(int i=mid+1;i<=r;i++)f[i]=(f[i]+ta[i-l])%mod;//算贡献
87     CDQ(mid+1,r);
88 }
89 void solve()
90 {
91     gi=ksm(ge,mod-2);
92     int n;
93     cin>>n;
94     for(int i=1;i<n;i++)
95     {
96         cin>>g[i];
97         f[i]=0;
98     }
99     f[0]=1; //这是 f 的边界
100     g[0]=0;
101     CDQ(0,n-1); //分治
102     for(int i=0;i<n;i++)cout<<f[i]<<" ";
103     cout<<"\n";

```

114

}

5.4.3 FFT_butterfly.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define ld long double
5  #define inf 0x3f3f3f3f
6  #define mod 1000000007
7  const ld PI=acos(-1.0); //取PI
8  const ll N=1<<22; //注意这个N
9  typedef complex<ld> comp;
10 int main()
11 {
12     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
13     int t =1;
14     void solve();
15     // cin>>t;
16     while(t--)<math>\text{solve}();</math>
17 }
18 //FFT 最基础的板子
19 //FFT+蝴蝶优化
20 //规避了动态数组带来的时间复杂度
21 //O(n*logn)
22 //FFT 需要运用到复数，建议手写复数，会更快
23 //FFT 会有精度问题
24 int mx=0;
25 int rev[N]={0};
26 ll p1[N],p2[N];
27 comp tmp1[N],tmp2[N];
28
29 void chang(comp *tmp,int len) //FFT 每次递归到底都有一定的规律，即下标二进制翻转
30 {
31     for(int i=0;i<len;i++) //二进制翻转
32     {
33         rev[i]=rev[i>>1]>>1;
34         if(i&1)rev[i]|=len>>1;
35     }
36     for(int i=0;i<len;i++)
37     {
38         if(i<rev[i])swap(tmp[i],tmp[rev[i]]); //交换
39     }
40 }
41
42 void FFT(comp *f,int n,int op)
43 {
44     chang(f,n); //先变换
45     //采用非递归方法求解
46     for(int mid=1;mid<n;mid<=1) //遍历交换中点，亦是半周期
47     {
48         comp w(cos(PI/mid),sin(PI*op/mid)); //单位根
49
50         for(int R=mid<<1,j=0;j<n;j+=R) //周期为 R,区间起点 j
51         {
52             comp cur(1,0); //变化的自变量
53
54             for(int k=0;k<mid;k++,cur*=w) //傅里叶变换
55             {
56

```



```

57         comp x=f[j+k],y=cur*f[j+k+mid];
58
59         f[j+k]=x+y;    //直接覆写
60
61         f[j+k+mid]=x-y; //直接覆写
62     }
63 }
64 }
65 }
66
67 void solve()
68 {
69     int n,m;
70     cin>>n>>m;
71     for(int i=0;i<=n;i++)
72     {
73         cin>>p1[i];
74         tmp1[i]=p1[i];
75     }
76     for(int i=0;i<=m;i++)
77     {
78         cin>>p2[i];
79         tmp2[i]=p2[i];
80     }
81     mx=1;
82     while(mx<n+m+1)mx<<=1;
83
84     for(int i=n+1;i<=mx;i++)
85     {
86         p1[i]=0; // 高位补上 0,保证是 2 的幂次
87         tmp1[i]=p1[i];
88     }
89     for(int i=m+1;i<=mx;i++)
90     {
91         p2[i]=0;
92         tmp2[i]=p2[i];
93     }
94
95     FFT(tmp1,mx,1); //两个 fft 求出点值式
96     FFT(tmp2,mx,1);
97
98     for(int i=0;i<=mx;i++)tmp1[i]=tmp1[i]*tmp2[i]; //值相乘
99
100    FFT(tmp1,mx,-1); //乘完后用 fft 转化为系数式
101
102    for(int i=0;i<n+m+1;i++)cout<<(ll)((ld)tmp1[i].real()/(ld)mx+0.5)<<"
103    "; //注意实部取四舍五入
104
105    cout<<"\n";
106 }

```

5.4.4 NTT.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define ld long double
5  #define inf 0x3f3f3f3f
6  #define mod 998244353
7  const ll N=1<<22; //注意这个 N
8  int main()
9  {
10     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
11     int t =1;
12     void solve();
13     // cin>>t;
14     while(t--)<math>\text{solve}();</math>
15 }
16 //NTT 最基础的板子
17 //FFT 涉及三角函数和复数，浮点计算导致运算的复杂度大，精度低
18 //由此诞生了快速数论变化 NTT
19 //换个根就行了
20 ll g=1,gi; //g 是原根，一般取 3, gi 是 g 的乘法逆元
21 ll tmp1[N],tmp2[N];
22 ll mx;
23 int rev[N]={0};
24
25 ll ksm(ll a,ll b) //快速幂
26 {
27     ll ans=1;
28     while(b)
29     {
30         if(b&1)ans=ans*a%mod;
31         a=a*a%mod;
32         b>>=1;
33     }
34     return ans;
35 }
36 void chang(ll *f,int n) //蝴蝶变换
37 {
38     for(int i=0;i<n;i++)
39     {
40         rev[i]=rev[i>>1]>>1;
41         rev[i]|=(i&1)?(n>>1):0;
42     }
43     for(int i=0;i<n;i++)
44         if(i<rev[i])swap(f[i],f[rev[i]]);
45 }
46
47 void NTT(ll *f,int n,int opt) //快速数论变换
48 {
49     chang(f,n);//先变换
50     for(int mid=1;mid<n;mid<<=1) //枚举半周期
51     {
52         ll g1=ksm((opt==1)?g:gi,(mod-1)/(mid*2)); //取单位根
53
54         for(int R=mid<<1,j=0;j<n;j+=R)
55         {
56

```

```

57         ll w=1; //变化的自变量
58
59         for(int k=0;k<mid;k++,w=(w*g1)%mod)
60         {
61             ll a1=f[j+k],a2=f[j+k+mid]*w%mod;
62
63             f[j+k]=(a1+a2)%mod;
64
65             f[j+k+mid]=(a1-a2+mod)%mod;
66         }
67     }
68 }
69
70 void solve()
71 {
72     int n,m;
73     cin>>n>>m;
74
75     g=3;gi=ksm(g,mod-2); //原根 和 它的倒数
76
77     for(int i=0;i<=n;i++)cin>>tmp1[i];
78     for(int i=0;i<=m;i++)cin>>tmp2[i];
79
80     mx=1;
81
82     while(mx<n+m+1)mx<<=1; //取不小于最高次数的二的幂
83
84     for(int i=n+1;i<mx;i++)tmp1[i]=0; //高位补0
85     for(int i=m+1;i<mx;i++)tmp2[i]=0;
86
87     NTT(tmp1,mx,1); //分别求点值式
88     NTT(tmp2,mx,1);
89
90     for(int i=0;i<mx;i++)tmp1[i]=tmp1[i]*tmp2[i]%mod; //值相乘
91
92     NTT(tmp1,mx,-1); //点值式转系数式
93
94     ll inv=ksm(mx,(mod-2)); //取mx的逆元
95     for(int i=0;i<n+m+1;i++)
96     {
97         cout<<tmp1[i]*inv%mod<<" "; //这里不要用除法,用乘法逆元
98     }
99     cout<<"\n";
100 }

```

5.4.5 NTT_INV.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define mod 998244353
5  const int N=(1<<22)|10;
6  int gen=3,gi;
7  int ksm(int a,int b)
8  {
9      int ans=1;
10     while(b)
11     {
12         if(b&1)ans=(ll)ans*a%mod;
13         a=(ll)a*a%mod;
14         b>>=1;
15     }
16     return ans;
17 }
18 signed main()
19 {
20     void solve();
21     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
22     gi=ksm(gen,mod-2);
23     solve();
24     return 0;
25 }
26 //多项式求逆元, 只能用 NTT
27 int a[N],rev[N]={0},b[N],tmp[N];
28 void chang(int *A,int n) //蝴蝶变换
29 {
30     for(int i=0;i<n;i++)
31     {
32         rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
33     }
34     for(int i=0;i<n;i++)
35     {
36         if(i<rev[i])swap(A[i],A[rev[i]]);
37     }
38 }
39 void NTT(int *A,int n,int opt)
40 {
41     chang(A,n); //先变换
42     for(int mid=1;mid<n;mid<<=1)
43     {
44         int g1=ksm((opt==1)?gen:gi,(mod-1)/(mid<<1));
45         for(int j=0;j<n;j+=(mid<<1))
46         {
47             int gk=1;
48             for(int k=0;k<mid;k++,gk=(ll)gk*g1%mod)
49             {
50                 int x=A[j+k],y=(ll)gk*A[j+k+mid]%mod;
51                 A[j+k]=((ll)x+(ll)y)%mod;
52                 A[j+k+mid]=((ll)x-(ll)y+mod)%mod;
53             }
54         }
55     }
56     if(opt==1)return ; //如果是系数式求点值式, 到这里即可

```

```

57
58     int inv=ksm(n,mod-2);          //否则这里直接处理出答案
59     for(int i=0;i<n;i++)A[i]=(1l)inv*A[i]%mod;
60 }
61 void INV(int n,int *A,int *B) //传入系数个数 n、原系数式 A、用于返回答案的系数式 B
62 {
63     //本质是一种分治的思想, n 的答案可由 n/2 求出, 而 n==1 的答案易得
64     stack<int> stk;
65     int mx=1;
66     while(n!=1){stk.push(n);n=(n+1)>>1;} //这里在用栈模拟递归
67     stk.push(1);
68     while(!stk.empty())
69     {
70         n=stk.top();
71         stk.pop(); //取栈顶, 先
72
73         if(n==1){B[0]=ksm(A[0],mod-2);continue;} //0 次即为常数的逆元
74
75         while(mx<(n<<1))mx<<=1; //处理出不小于 n*2 的 二的幂
76
77         for(int i=0;i<n;i++)tmp[i]=A[i]; //低位复制
78         for(int i=n;i<mx;i++)tmp[i]=0; //高位补 0
79
80         NTT(tmp,mx,1);NTT(B,mx,1); //求点值式
81
82         for(int i=0;i<mx;i++)B[i]=(2ll-
83 (1l)tmp[i]*B[i]%mod+mod)%mod*B[i]%mod; //套公式计算答案值
84
85         NTT(B,mx,-1); //点值式转系数式
86
87         for(int i=n;i<mx;i++)B[i]=0; //高位补 0
88     }
89 }
90 void solve()
91 {
92     int n;
93     cin>>n;
94     for(int i=0;i<n;i++)cin>>a[i];
95     INV(n,a,b);
96     for(int i=0;i<n;i++)cout<<b[i]<<" ";
97     cout<<"\n";
98 }

```

5.4.6 sqrt.h

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define inf 0x3f3f3f3f
5  #define mod 998244353
6  int ksm(int a,int b)
7  {
8      int ans=1;
9      while(b)
10     {
11         if(b&1)ans=(ll)ans*a%mod;
12         a=(ll)a*a%mod;
13         b>>=1;
14     }
15     return ans;
16 }
17 //多项式开根号, 需要 NTT+INV+二次剩余(cipolla)
18 const int gen=3,gi=ksm(gen,mod-2),N=(1<<22)|10,ny2=ksm(2,mod-2);
19 int main()
20 {
21     void solve();
22     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
23     solve();
24     return 0;
25 }
26 struct comp{int r,i;};
27 comp image_mul(comp a,comp b,int w)
28 {
29     comp ans;
30     ans.r=((ll)a.r*b.r%mod+(ll)a.i*b.i%mod*w%mod)%mod;
31     ans.i=((ll)a.r*b.i%mod+(ll)a.i*b.r%mod)%mod;
32     return ans;
33 }
34 comp ksm_image(comp a,int b,int w)
35 {
36     comp res;
37     res.r=1;res.i=0;
38     while(b)
39     {
40         if(b&1)res=image_mul(res,a,w);
41         a=image_mul(a,a,w);
42         b>>=1;
43     }
44     return res;
45 }
46 int cipolla(int n) //求二次剩余, 仅用于求常数 mod 意义下的二次方根
47 {
48     int w,a=rand()%mod;
49     w=((ll)a*a-n+mod)%mod;
50     if(ksm(w,(mod-1)>>1)==0)return 0;
51     while(ksm(w,(mod-1)>>1)!=mod-1)a=rand()%mod,w=((ll)a*a-n+mod)%mod;
52     comp ans;
53     ans.r=a;
54     ans.i=1;
55     ans=ksm_image(ans,((ll)mod+1)>>1,w);
56     int a1=ans.r,a2=(mod-a1)%mod;

```

```

57     if(a1>a2)swap(a1,a2);
58     return a1;
59 }
60
61 int a[N],b[N],c[N],d[N],e[N],rev[N]={0};
62 void chang(int *A,int n)
63 {
64     for(int i=0;i<n;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
65     for(int i=0;i<n;i++)if(i<rev[i])swap(A[i],A[rev[i]]);
66 }
67 void NTT(int *A,int n,int f) //多项式相乘
68 {
69     chang(A,n);
70     for(int mid=1;mid<n;mid<<=1)
71     {
72         int g1=ksm((f==1)?gen:gi,(mod-1)/(mid<<1));
73         for(int j=0;j<n;j+=(mid<<1))
74         {
75             int gk=1;
76             for(int k=0;k<mid;k++,gk=(1l)gk*g1%mod)
77             {
78                 int x=A[j+k],y=(1l)A[j+k+mid]*gk%mod;
79                 A[j+k]=((1l)x+(1l)y)%mod;
80                 A[j+k+mid]=((1l)x-(1l)y+mod)%mod;
81             }
82         }
83     }
84     if(f==1)return ;
85     int inv=ksm(n,mod-2);
86     for(int i=0;i<n;i++)A[i]=(1l)A[i]*inv%mod;
87 }
88 void INV(int n,int *A,int *B)//多项式求逆
89 {
90     stack<int> stk;
91     while(n!=1){stk.push(n);n=(n+1)>>1;}
92     stk.push(1);
93     int mx=1;
94     while(!stk.empty())
95     {
96         n=stk.top();
97         stk.pop();
98         if(n==1){B[0]=ksm(A[0],mod-2);continue;}
99         while(mx<(n<<1))mx<<=1;
100         for(int i=0;i<n;i++)c[i]=A[i];
101         for(int i=n;i<mx;i++)c[i]=0;
102         NTT(c,mx,1);NTT(B,mx,1);
103         for(int i=0;i<mx;i++)B[i]=(21l-(1l)c[i]*B[i]%mod+mod)%mod*B[i]%mod;
104         NTT(B,mx,-1);
105         for(int i=n;i<mx;i++)B[i]=0;
106     }
107 }
108 void Sqrt(int n,int *A,int *B) //多项式开根
109 {
110     //思想和 INV 相同：分治、套公式
111     stack<int> stk;
112     while(n!=1){stk.push(n);n=(n+1)>>1;}
113     stk.push(1);

```

```

114     int mx=1;
115     while(!stk.empty())
116     {
117         n=stk.top();
118         stk.pop();
119         if(n==1){B[0]=cipolla(A[0]);continue;} //常数求二次剩余
120         while(mx<(n<<1))mx<<=1;
121         for(int i=0;i<n;i++)d[i]=A[i],e[i]=0;
122         for(int i=n;i<mx;i++)d[i]=0,e[i]=0;
123         //e 是 B 的逆元
124
125         INV(n,B,e);NTT(d,mx,1);NTT(B,mx,1);NTT(e,mx,1);
126
127         for(int i=0;i<mx;i++)B[i]=((ll)d[i]*e[i]%mod+(ll)B[i])%mod*ny2%mod;//
套公式
128
129         NTT(B,mx,-1);
130
131         for(int i=n;i<mx;i++)B[i]=0;
132     }
133 }
134 void solve()
135 {
136     int n;
137     cin>>n;
138     for(int i=0;i<n;i++)cin>>a[i];
139     Sqrt(n,a,b);
140     for(int i=0;i<n;i++)cout<<b[i]<<" ";
141     cout<<"\n";
142 }

```


6 Misc

6.1 莫队.h

```

1 // sz =int(sqrt(n)) 注意考虑 sqrtl(n)
2 struct node{
3     int l,r,id;
4     bool operator<(const node &x) const{
5         if(l/sz!=x.l/sz) return l<x.l;
6         if((l/sz)&1) return r<x.r;
7         else return r>x.r;
8     }
9 };

```

7 STRING

7.1 AC_automaton.h

```

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

```

```

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

```

7.2 compress_print.h

```

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

```

7.3 get_occrr.h

```

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

```

7.4 hash_print.h

```

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

```

```
58     }  
59 };
```

7.5 KMP.h

```

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

```


7.6 Manacher.h

```

1  pair<vector<int>,vector<int>> Manacher(string s){
2      // d1: a b [c:3] b a
3      // d2: a b [b:2] a
4      int n = s.size();
5      vector<int> d1(n);
6      for (int i = 0, l = 0, r = -1; i < n; i++)
7      {
8          int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
9          while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;
10         d1[i] = k--;
11         if (i + k > r) l = i - k, r = i + k;
12     }
13     vector<int> d2(n);
14     for (int i = 0, l = 0, r = -1; i < n; i++)
15     {
16         int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
17         while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) k++;
18         d2[i] = k--;
19         if (i + k > r) l = i - k - 1, r = i + k;
20     }
21     return {d1,d2};
22 }
```

7.7 Palindromic_automaton.h

```

1  class PA {
2  private:
3      static const int N = 100010;
4      struct Node {
5          int len;
6          int ptr[26], fail;
7          Node(int len = 0) : len(len), fail(0) { memset(ptr, 0, sizeof(ptr)); }
8      } nd[N];
9
10     int size, cnt; // size 为字符串长度, cnt 为节点个数
11     int cur; // 当前指针停留的位置, 即最后插入字符所对应的节点
12     char s[N];
13
14     int getfail(int x) // 沿着 fail 指针找到第一个回文后缀
15     {
16         while (s[size - nd[x].len - 1] != s[size]) {
17             x = nd[x].fail;
18         }
19         return x;
20     }
21
22 public:
23     PA() : size(0), cnt(0), cur(0) {
24         nd[cnt] = Node(0);
25         nd[cnt].fail = 1;
26         nd[++cnt] = Node(-1);
27         nd[cnt].fail = 0;
28         s[0] = '$';
29     }
30
31     void extend(char c) {
32         s[++size] = c;
33         int now = getfail(cur); // 找到插入的位置
34         if (!nd[now].ptr[c - 'a']) // 若没有这个节点, 则新建并求出它的 fail 指针
35         {
36             int tmp = ++cnt;
37             nd[tmp] = Node(nd[now].len + 2);
38             nd[tmp].fail = nd[getfail(nd[now].fail)].ptr[c - 'a'];
39             nd[now].ptr[c - 'a'] = tmp;
40         }
41         cur = nd[now].ptr[c - 'a'];
42     }
43
44     int qlen() { return nd[cur].len; }
45 }

```

7.8 trie_Tree.h

```

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

```

