Contents

1 BASIC

1.1 bigInt.h

```cpp
namespace BigIntMiniNS {
const int32_t COMPRESS_MOD = 10000;
const uint32_t COMPRESS_DIGITS = 4;

const uint32_t BIGINT_MUL_THRESHOLD = 60;
const uint32_t BIGINT_DIVIDEDIV_THRESHOLD = BIGINT_MUL_THRESHOLD * 3;

template <typename T> inline T high_digit(T digit) { return digit / (T)COMPRESS_MOD; }

template <typename T> inline uint32_t low_digit(T digit) { return (uint32_t)(digit % (T)COMPRESS_MOD); }

class BigIntMini {
protected:
    typedef uint32_t base_t;
    typedef int32_t carry_t;
    typedef uint32_t ucarry_t;
    int sign;
    std::vector<base_t> v;
    typedef BigIntMini BigInt_t;
    template <typename _Tx, typename _Ty> static inline void carry(_Tx &add, _Ty &baseval, _Tx newval) {
        add += newval;
        baseval = low_digit(add);
        add = high_digit(add);
    }
    template <typename _Tx, typename _Ty> static inline void borrow(_Tx &add, _Ty &baseval, _Tx newval) {
        add += newval - COMPRESS_MOD + 1;
        baseval = (_Tx)low_digit(add) + COMPRESS_MOD - 1;
        add = high_digit(add);
    }

    bool raw_less(const BigInt_t &b) const {
        if (v.size() != b.size()) return v.size() < b.size();
        for (size_t i = v.size() - 1; i < v.size(); i--)
            if (v[i] != b.v[i]) return v[i] < b.v[i];
        return false; // eq
    }
    bool raw_eq(const BigInt_t &b) const {
        if (v.size() != b.size()) return false;
        for (size_t i = 0; i < v.size(); ++i)
            if (v[i] != b.v[i]) return false;
        return true;
    }
    BigInt_t &raw_add(const BigInt_t &b) {
        if (v.size() < b.size()) v.resize(b.size());
        ucarry_t add = 0;
        for (size_t i = 0; i < b.v.size(); i++)
            carry(add, v[i], (ucarry_t)(v[i] + b.v[i]));
        for (size_t i = b.v.size(); add && i < v.size(); i++)
            carry(add, v[i], (ucarry_t)v[i]);
        add ? v.push_back((base_t)add) : trim();
```

```
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) /
(double)COMPRESS_MOD) / COMPRESS_MOD;
135         else if (b.size() > 1)
136             db += b.v[b.size() - 2] / (double)COMPRESS_MOD;
137         db = 1 / db;
138         for (size_t i = a.size() - offset; i <= a.size();) {
139             carry_t rm = (carry_t)r.v[i + offset] * COMPRESS_MOD + r.v[i +
offset - 1], m;
140             m = std::max((carry_t)(rm * db), (carry_t)r.v[i + offset]);
141             if (m) {
142                 v[i] += (base_t)m;
143                 carry_t add = 0;
144                 for (size_t j = 0; j < b.size(); j++)
145                     borrow(add, r.v[i + j], (carry_t)r.v[i + j] -
(carry_t)b.v[j] * m);
146                 for (size_t j = i + b.size(); add && j < r.size(); ++j)
147                     borrow(add, r.v[j], (carry_t)r.v[j]);
148             }
149             i -= !r.v[i + offset];
150         }
151         r.trim();
152         carry_t add = 0;
153         while (!r.raw_less(b)) {
154             r.raw_sub(b);
155             ++add;
156         }
157 
158         for (size_t i = 0; i < v.size(); i++)
159             carry(add, v[i], (carry_t)v[i]);
160         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
&r) {
220         if (b.size() <= BIGINT_DIVIDEDIV_THRESHOLD) {
221             raw_div(a, b, r);
222             return *this;
223         }
224         if (b.size() * 2 - 2 > a.size()) {
225             BigInt_t ta = a, tb = b;
226             size_t ans_len = a.size() - b.size() + 2;
227             size_t shr = b.size() - ans_len;
228             ta.raw_shr(shr);
229             tb.raw_shr(shr);
230             return raw_dividediv(ta, tb, r);
231         }
232         carry_t mul = (carry_t)(((uint64_t)COMPRESS_MOD * COMPRESS_MOD -
1) /               //
                                    (*(b.v.begin() + b.v.size() - 1) *
233 (uint64_t)COMPRESS_MOD + //
                                     *(b.v.begin() + b.v.size() - 2) + 1));
234         BigInt_t ma = a * BigInt_t(mul), mb = b * BigInt_t(mul);
235         while (mb.v.back() < COMPRESS_MOD >> 1) {
236             int32_t m = 2;
237             ma.raw_mul(ma, BigInt_t(m));
238             mb.raw_mul(mb, BigInt_t(m));
239             mul *= m;
240         }
241         BigInt_t d;
242         ma.sign = mb.sign = 1;
243         raw_dividediv_recursion(ma, mb, d);
244         r.raw_div(d, BigInt_t((int)mul), ma);
245         return *this;
246     }
247     void trim() {
248         while (v.back() == 0 && v.size() > 1)
249             v.pop_back();
250     }
251     size_t size() const { return v.size(); }
252     BigInt_t &from_str_base10(const char *s) {
253         v.clear();
254         int32_t base = 10, sign = 1, digits = COMPRESS_DIGITS;
255         const char *p = s + strlen(s) - 1;
256         while (*s == '-')
257             sign *= -1, ++s;
258         while (*s == '0')
259             ++s;
260
261         int32_t d = digits, hdigit = 0, hdigit_mul = 1;
262         for (; p >= s; p--) {
263             hdigit += (*p - '0') * hdigit_mul;
264             hdigit_mul *= base;
265             if (--d == 0) {
266                 v.push_back(hdigit);
267                 d = digits;
268                 hdigit = 0;
269                 hdigit_mul = 1;
270
```

```
            }
        }
        if (hdigit || v.empty()) v.push_back(hdigit);
        this->sign = sign;
        return *this;
    }

public:
    BigIntMini() { set(0); }
    explicit BigIntMini(int n) { set(n); }
    explicit BigIntMini(intmax_t n) { set(n); }
    explicit BigIntMini(const char *s) { from_str(s); }
    BigInt_t &set(intmax_t n) {
        v.resize(1);
        v[0] = 0;
        uintmax_t s;
        if (n < 0) {
            sign = -1;
            s = -n;
        } else {
            sign = 1;
            s = n;
        }
        for (int i = 0; s; i++) {
            v.resize(i + 1);
            v[i] = low_digit(s);
            s = high_digit(s);
        }
        return *this;
    }
    BigInt_t &from_str(const char *s) { return from_str_base10(s); }
    bool is_zero() const { return v.size() == 1 && v[0] == 0; }
    bool operator<(const BigInt_t &b) const {
        if (sign * b.sign > 0) {
            if (sign > 0)
                return raw_less(b);
            else
                return b.raw_less(*this);
        } else {
            if (sign > 0)
                return false;
            else
                return true;
        }
    }
    bool operator==(const BigInt_t &b) const {
        if (is_zero() && b.is_zero()) return true;
        if (sign != b.sign) return false;
        return raw_eq(b);
    }
    LESS_THAN_AND_EQUAL_COMPARABLE(BigInt_t)

    BigInt_t &operator=(intmax_t n) { return set(n); }
    BigInt_t &operator=(const char *s) { return from_str(s); }
    BigInt_t operator+(const BigInt_t &b) const {
        BigInt_t r = *this;
        if (sign * b.sign > 0)
            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
    BIGINT_STD_MOVE(*this - *this / b * b); }
359         BigInt_t div(const BigInt_t &b, BigInt_t &r) {
360             if (this == &b) {
361                 r.set(0);
362                 return set(1);
363             }
364             BigInt_t d;
365             d.raw_dividediv(*this, b, r);
366             d.sign = sign * b.sign;
367             return BIGINT_STD_MOVE(d);
368         }
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

```python
from decimal import *
import sys
setcontext(Context(prec=2000000, Emax=2000000, Emin=0))
a = sys.stdin.readline()
b = sys.stdin.readline()
print(Decimal(a) * Decimal(b))
```

1.3 bitset.h

```cpp
#include<bitset>
int n;
auto bin = bitset<32>(n);

cout << bin;
//输出二进制位

cout << bin.to_ullong();
//输出十进制

bin[1] = 1
//随机访问

bin = !bin ^ (bin & bin | bitset<32>(1))
//位运算

bin != bin
//比较运算符

bin.count()
//1 的数量

bin.test(i)
//随机访问，类似 std::vector::pos()

bin.any()
//有一位 1 就 true

bin.none()
//全 0 就返回 true

bin.all()
//全 1 就返回 true

bin.flip()
//翻转全部

bin.flip(i)
//a[i] = !a[i]

bin._Find_first()
//第一个 1 的下标

bin._Find_next(i)
//从下标 n 往后第一个 1 的下标
```

1.4 pbds.h

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
using ord_set = tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>;
using ord_mset =  tree<int, null_type, less_equal<int>, rb_tree_tag,
tree_order_statistics_node_update>;
//find_by_order
//order_of_key
```

1.5 时间戳优化.h

```cpp
//时间戳优化：对付多组数据很常见的技巧。
int tag[N], t[N], Tag;
int lowbit(int x){
  return x&-x;
}
void reset(){
  ++Tag;
}
void add(int x,int val){
  while(x<=n){
    if(tag[x]!=Tag) t[x]=0;
    t[x]+=val;tag[x]=Tag;
    x+=lowbit(x);
  }
}
int getsum(int x){
  int ans=0;
  while(x){
    if(tag[x]==Tag) ans+=t[x];
    x-=lowbit(x);
  }
  return ans;
}
```

2 Ds

2.1 dsu.h

```cpp
class DSU {
    std::vector<int> f, siz;
public:
    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        for(int i = 0;i < n;i ++) f[i] = i;
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

2.2 dsu_classification.h

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+100;
#define ll long long
#define int long long

struct node{
  int x,y,z;
}s[N];
vector<int> a(N),b(N);
int find(int x){
  return (a[x]==x) ? x : a[x]=find(a[x]);
}
void merge(int x,int y){
  a[find(x)]=find(y);
}
void solve(){
  int n,m;
  cin>>n>>m;
  vector<int> v(n+1);
  for(int i=0;i<=n;i++) a[i]=i;
  for(int i=0;i<m;i++) cin>>s[i].x>>s[i].y>>s[i].z;
  sort(s,s+m,[&](node a,node b){
    return a.z>b.z;
  });
  for(int i=0;i<m;i++){
    if(find(s[i].x)==find(s[i].y)){
      cout<<s[i].z;
      return;
    }
    if(!b[s[i].x]) b[s[i].x]=s[i].y;
    else merge(s[i].y,b[s[i].x]);
    if(!b[s[i].y]) b[s[i].y]=s[i].x;
    else merge(s[i].x,b[s[i].y]);
  }
  cout<<0;
  return;
}


signed main(){
  ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
  int t=1;
//  cin>>t;
  while(t--) solve();
  return 0;
}
```

2.3 dsu_weighted.h

```cpp
#include <cassert>
#include <iostream>
#include <numeric>
#include <vector>

using namespace std;

struct dsu {
  vector<size_t> pa, size, sum;

  explicit dsu(size_t size_)
      : pa(size_ * 2), size(size_ * 2, 1), sum(size_ * 2) {
    // 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
    iota(pa.begin(), pa.begin() + size_, size_);
    iota(pa.begin() + size_, pa.end(), size_);
    iota(sum.begin() + size_, sum.end(), 0);
  }

  void unite(size_t x, size_t y) {
    x = find(x), y = find(y);
    if (x == y) return;
    if (size[x] < size[y]) swap(x, y);
    pa[y] = x;
    size[x] += size[y];
    sum[x] += sum[y];
  }

  void move(size_t x, size_t y) {
    auto fx = find(x), fy = find(y);
    if (fx == fy) return;
    pa[x] = fy;
    --size[fx], ++size[fy];
    sum[fx] -= x, sum[fy] += x;
  }

  size_t find(size_t x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }
};

int main() {
  size_t n, m, op, x, y;
  while (cin >> n >> m) {
    dsu dsu(n + 1);  // 元素范围是 1..n
    while (m--) {
      cin >> op;
      switch (op) {
        case 1:
          cin >> x >> y;
          dsu.unite(x, y);
          break;
        case 2:
          cin >> x >> y;
          dsu.move(x, y);
          break;
        case 3:
          cin >> x;
          x = dsu.find(x);
          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

```cpp
// AC 带懒惰标记线段树
template <class TYPE_NAME>
class lazyTree
{
private:
    vector<TYPE_NAME> d;
    vector<TYPE_NAME> a;
    vector<TYPE_NAME> b;
    int n;
    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

    void subbuild(int s, int t, int p)
    {
        if (s == t)
        {
            d[p] = a[s];
            return;
        }
        int m = s + ((t - s) >> 1); //  (s+t)/2
        subbuild(s, m, p * 2);
        subbuild(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
        //     合并运算符 ↑
    }

    TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d[p];
        int m = s + ((t - s) >> 1);
        if (b[p] != 0)
        {
            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
            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
            b[p * 2] += b[p];               // 下传标记，无需修改
            b[p * 2 + 1] += b[p];           // 下传标记，无需修改
            b[p] = 0;
        }
        TYPE_NAME ansl = INI;
        TYPE_NAME ansr = INI;
        if (l <= m)
            ansl = subGetSum(l, r, s, m, p * 2);
        if (r > m)
            ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);
        return ansl + ansr;
        // 合并运算符↑
    }

    void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
    {
        if (l <= s && t <= r)
        {
```

```
            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
            b[p] += c;               // 记录懒惰标记，无需修改
            return;
        }
        int m = s + ((t - s) >> 1);
        if (b[p] != 0 && s != t)
        {
            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
            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
            b[p * 2] += b[p];               // 下传标记，无需修改
            b[p * 2 + 1] += b[p];           // 下传标记，无需修改
            b[p] = 0;
        }
        if (l <= m)
            subUpdate(l, r, c, s, m, p * 2);
        if (r > m)
            subUpdate(l, r, c, m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
        //    合并运算符 ↑
    }

public:
    lazyTree(TYPE_NAME _n)
    {
        n = _n;
        d.resize(4 * n + 5);
        a.resize(4 * n + 5);
        b.resize(4 * n + 5);
    }

    void build(vector<TYPE_NAME> _a)
    {
        a = _a;
        subbuild(1, n, 1);
    }

    TYPE_NAME getsum(int l, int r)
    {
        return subGetSum(l, r, 1, n, 1);
    }

    void update(int l, int r, TYPE_NAME c) // 修改区间
    {
        subUpdate(l, r, c, 1, n, 1);
    }

    void update(int idx, TYPE_NAME tar)
    { // 修改单点，未测试
        TYPE_NAME tmp = getsum(idx, idx);
        tar -= tmp;
        subUpdate(idx, idx, tar, 1, n, 1);
    }
};
```

2.5 segTree_add_setto.h

```cpp
template <typename T>
class SegTreeLazyRangeSet {
  vector<T> tree, lazy;
  vector<T> *arr;
  vector<bool> ifLazy;
  int n, root, n4, end;

  void maintain(int cl, int cr, int p) {
    int cm = cl + (cr - cl) / 2;
    if (cl != cr && ifLazy[p]) {
      lazy[p * 2] = lazy[p],ifLazy[p*2] = 1;
      lazy[p * 2 + 1] = lazy[p],ifLazy[p*2+1] = 1;
      tree[p * 2] = lazy[p] * (cm - cl + 1);
      tree[p * 2 + 1] = lazy[p] * (cr - cm);
      lazy[p] = 0;
      ifLazy[p] = 0;
    }
  }

  T range_sum(int l, int r, int cl, int cr, int p) {
    if (l <= cl && cr <= r) return tree[p];
    int m = cl + (cr - cl) / 2;
    T sum = 0;
    maintain(cl, cr, p);
    if (l <= m) sum += range_sum(l, r, cl, m, p * 2);
    if (r > m) sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
    return sum;
  }

  void range_set(int l, int r, T val, int cl, int cr, int p) {
    if (l <= cl && cr <= r) {
      lazy[p] = val;
      ifLazy[p] = 1;
      tree[p] = (cr - cl + 1) * val;
      return;
    }
    int m = cl + (cr - cl) / 2;
    maintain(cl, cr, p);
    if (l <= m) range_set(l, r, val, cl, m, p * 2);
    if (r > m) range_set(l, r, val, m + 1, cr, p * 2 + 1);
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
  }

  void build(int s, int t, int p) {
    if (s == t) {
      tree[p] = (*arr)[s];
      return;
    }
    int m = s + (t - s) / 2;
    build(s, m, p * 2);
    build(m + 1, t, p * 2 + 1);
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
  }

 public:
  explicit SegTreeLazyRangeSet<T>(vector<T> v) {
    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

```cpp
/*
三种操作:（模数为 mod）
1.区间乘 val
2.区间加 val
3.区间和
*/
#define int long long
#define lc p<<1
#define rc p<<1|1
const int N=1e5+10;
int mod=1e9+7;
struct node{
  int l,r,sum,mul,add;
}tr[4*N];
vector<int> a(N);
void pushup(int p){
  tr[p].sum=tr[lc].sum+tr[rc].sum;
  tr[p].sum%=mod;
}
void build(int p,int l,int r){
  tr[p].l=l;tr[p].r=r;
  tr[p].mul=1;tr[p].add=0;
  if(l==r){
    tr[p].sum=a[l];
    return;
  }
  int m=l+r>>1;
  build(lc,l,m);
  build(rc,m+1,r);
  pushup(p);
}
void pushdown(int p){
  tr[lc].sum=(tr[lc].sum*tr[p].mul+tr[p].add*(tr[lc].r-tr[lc].l+1))%mod;
  tr[rc].sum=(tr[rc].sum*tr[p].mul+tr[p].add*(tr[rc].r-tr[rc].l+1))%mod;
  tr[lc].mul=tr[lc].mul*tr[p].mul%mod;
  tr[rc].mul=tr[rc].mul*tr[p].mul%mod;
  tr[lc].add=(tr[lc].add*tr[p].mul+tr[p].add)%mod;
  tr[rc].add=(tr[rc].add*tr[p].mul+tr[p].add)%mod;
  tr[p].mul=1;tr[p].add=0;
}
void update_mul(int p,int l,int r,int val){
  if(l<=tr[p].l&&tr[p].r<=r){
    tr[p].sum=tr[p].sum*val%mod;
    tr[p].mul=tr[p].mul*val%mod;
    tr[p].add=tr[p].add*val%mod;
    return;
  }
  pushdown(p);
  int m=tr[p].l+tr[p].r>>1;
  if(l<=m) update_mul(lc,l,r,val);
  if(m<r) update_mul(rc,l,r,val);
  pushup(p);
  return;
}
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

```cpp
//AC MJ 的 MIN/MAX 树
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
```

```
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
const int inf = 1E9;
struct Info
{
```

```
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

```cpp
#include <algorithm>
#include <climits>
#include <iostream>
using namespace std;
using ll = long long;

constexpr int N = 1e5 + 7;

struct Tree {
  int mx, _mx;  // 区间最大值 区间历史最大值
  int ad, _ad;  // 区间加标记 区间阶段历史最大加标记
  int st, _st;  // 区间修改值 区间阶段历史最大修改标记
} g[N * 4];

int a[N];
int n, m;
#define ls u * 2
#define rs u * 2 + 1
#define mid (l + r) / 2

void pushup(int u) {
  g[u].mx = max(g[ls].mx, g[rs].mx);
  g[u]._mx = max(g[ls]._mx, g[rs]._mx);
}

void pushadd(int u, int v, int _v) {
  g[u]._mx = max(g[u]._mx, g[u].mx + _v), g[u].mx += v;
  if (g[u].st == INT_MIN)
    g[u]._ad = max(g[u]._ad, g[u].ad + _v), g[u].ad += v;
  else
    g[u]._st = max(g[u]._st, g[u].st + _v), g[u].st += v;
}

void pushset(int u, int v, int _v) {
  g[u]._mx = max(g[u]._mx, _v), g[u].mx = v;
  g[u]._st = max(g[u]._st, _v), g[u].st = v;
}

void pushdown(int u, int l, int r) {
  if (g[u].ad || g[u]._ad)
    pushadd(ls, g[u].ad, g[u]._ad), pushadd(rs, g[u].ad, g[u]._ad),
        g[u].ad = g[u]._ad = 0;
  if (g[u].st != INT_MIN || g[u]._st != INT_MIN)
    pushset(ls, g[u].st, g[u]._st), pushset(rs, g[u].st, g[u]._st),
        g[u].st = g[u]._st = INT_MIN;
}

void build(int u = 1, int l = 1, int r = n) {
  g[u]._st = g[u].st = INT_MIN;
  if (l == r) {
    g[u].mx = a[l];
    g[u]._mx = a[l];
    return;
  }
  build(ls, l, mid), build(rs, mid + 1, r);
  pushup(u);
}
```

```cpp

int L, R;

void add(int v, int u = 1, int l = 1, int r = n) {
  if (R < l || r < L) return;
  if (L <= l && r <= R) return pushadd(u, v, max(v, 0));
  pushdown(u, l, r);
  add(v, ls, l, mid), add(v, rs, mid + 1, r);
  pushup(u);
}

void tset(int v, int u = 1, int l = 1, int r = n) {
  if (R < l || r < L) return;
  if (L <= l && r <= R) return pushset(u, v, v);
  pushdown(u, l, r);
  tset(v, ls, l, mid), tset(v, rs, mid + 1, r);
  pushup(u);
}

int qmax(int u = 1, int l = 1, int r = n) {
  if (R < l || r < L) return INT_MIN;
  if (L <= l && r <= R) return g[u].mx;
  pushdown(u, l, r);
  return max(qmax(ls, l, mid), qmax(rs, mid + 1, r));
}

int qmaxh(int u = 1, int l = 1, int r = n) {
  if (R < l || r < L) return INT_MIN;
  if (L <= l && r <= R) return g[u]._mx;
  pushdown(u, l, r);
  return max(qmaxh(ls, l, mid), qmaxh(rs, mid + 1, r));
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n;
  for (int i = 1; i <= n; ++i) cin >> a[i];
  build();
  int m, z;
  cin >> m;
  for (int i = 1; i <= m; ++i) {
    char op;
    cin >> op;
    while (op == ' ' || op == '\r' || op == '\n') cin >> op;
    cin >> L >> R;
    int x;
    if (op == 'Q')
      cout << qmax() << '\n';
    else if (op == 'A')
      cout << qmaxh() << '\n';
    else if (op == 'P')
      cin >> x, add(x);
    else
      cin >> x, tset(x);
  }
  return 0;
}
```

2.9 SparseTable.h

```cpp
class SparseTable
{
    using func_type = function<int(const int &, const int &)>;

    vector<vector<int>> ST;
    int len;
    vector<int> preLog;
    func_type op;
    static int default_func(const int &t1, const int &t2) { return max(t1, t2); }

public:
    void build(const vector<int> &v, func_type _func = default_func)
    {
        op = _func;
        len = v.size();
        int l1 = ceil(log2(len)) + 1;
        ST.assign(len, vector<int>(l1, 0));
        for (int i = 0; i < len; ++i)
        {
            ST[i][0] = v[i];
        }
        for (int j = 1; j < l1; ++j)
        {
            int pj = (1 << (j - 1));
            for (int i = 0; i + pj < len; ++i)
            {
                ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
            }
        }
        preLog.resize(len + 1);
        lop(i, 1, len + 1) preLog[i] = floor(log2(i));
    }

    int getsum(int l, int r)
    {
        if (r < l)
            return 0;
        l = max(0, l), r = min(len - 1, r);
        if (r == 0)
            return 0;
        int lt = r - l + 1;
        int q = preLog[lt];
        return op(ST[l][q], ST[r - (1 << q) + 1][q]);
    }
};
```

2.10 twoDimPrfxSum.h

```cpp
class prfx_2{
public:
    vector<vector<int>> mtx;
    int n,m;
    public:
    prfx_2(vector<vector<int>> _mtx){init(_mtx);};
    prfx_2() { };
    void init(vector<vector<int>> _mtx)
    {
        n = _mtx.size();
        m = _mtx[0].size();
        mtx.resize(n+1);
        for(auto &x:mtx) x.resize(m+1);
        lop(i,1,n+1)
            lop(j,1,m+1)
                mtx[i][j] = mtx[i-1][j] + mtx[i][j-1] - mtx[i-1][j-1] + _mtx[i-1][j-1];
    }

    int getsum(int x1,int y1,int x2,int y2)
    {
        x1 ++,x2 ++,y1 ++,y2 ++;
        return mtx[x2][y2] - mtx[x1-1][y2] - mtx[x2][y1-1] + mtx[x1-1][y1-1];
    }

    int getsum(pii d1,pii d2)
    {
        auto [x1,y1] = d1;
        auto [x2,y2] = d2;
        x1 ++,x2 ++,y1 ++,y2 ++;
        return mtx[x2][y2] - mtx[x1-1][y2] - mtx[x2][y1-1] + mtx[x1-1][y1-1];
    }

    vector<int> & operator[](std::size_t i)
    {
        return mtx[i];
    }

};

class conj_diff_2{
    vector<vector<int>> mtx;
    vector<vector<int>> prf;
    int n,m;
    public:

    conj_diff_2(int _n,int _m)
    {
        n = _n+1,m = _m+1;
        vector<vector<int>> initmp(n,vector<int> (m,0));
        init(initmp);
    }

    conj_diff_2(vector<vector<int>> _mtx)
    {
        this->init(_mtx);
    }
```

```
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

```cpp
#define int long long

const int N=2e5+10;
#define lc(x) tr[x].l
#define rc(x) tr[x].r
struct node{
  int l,r,s; //l,r->son  s->cnt
}tr[N*20];
int root[N],idx;
int n,m,a[N];
vector<int> b;
void build(int &x,int l,int r){
  x=++idx;
  if(l==r) return;
  int m=l+r>>1;
  build(lc(x),l,m);
  build(rc(x),m+1,r);
}
void insert(int x,int &y,int l,int r,int k){
  y=++idx; tr[y]=tr[x]; tr[y].s++;
  if(l==r) return;
  int m=l+r>>1;
  if(k<=m) insert(lc(x),lc(y),l,m,k);
  else insert(rc(x),rc(y),m+1,r,k);
}
int query(int x,int y,int l,int r,int k){
  if(l==r) return l;
  int m=l+r>>1;
  int s=tr[lc(y)].s-tr[lc(x)].s;
  if(k<=s) return query(lc(x),lc(y),l,m,k);
  else return query(rc(x),rc(y),m+1,r,k-s);
}

void solve(){
  int n,m;
  cin>>n>>m;
  for(int i=1;i<=n;i++){
    cin>>a[i];
    b.push_back(a[i]);
  }
  sort(b.begin(),b.end());
  b.erase(unique(b.begin(),b.end()),b.end());
  int bn=b.size();
  build(root[0],1,bn);
  for(int i=1;i<=n;i++){
    int id=lower_bound(b.begin(),b.end(),a[i])-b.begin()+1;
    insert(root[i-1],root[i],1,bn,id);
  }
  while(m--){
    int l,r,k;
    cin>>l>>r>>k;
    int id=query(root[l-1],root[r],1,bn,k)-1;
    cout<<b[id]<<"\n";
  }
}
```

2.12 单调队列.h

```cpp
void solve(){
  int n,k;
  cin>>n>>k;
  vector<int> a(n+1),ans(n+1);
  deque<int> q;
  for(int i=1;i<=n;i++){
    cin>>a[i];
    while(!q.empty()&&(q.front()<=i-k)) q.pop_front();
    while(!q.empty()&&a[q.back()]<=a[i]) q.pop_back();
    q.push_back(i);
    ans[i]=a[q.front()];
  }
}
```

2.13 左偏树.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define N 100005
const ll inf=1e17+7;
signed main()
{
  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  void solve();
  int t=1;
    while(t--)solve();
    return 0;
}
vector<int> a;
int fa[N];
int find(int u)
{
    return (fa[u]==u)?u:fa[u]=find(fa[u]);
}
struct left_tree
{
    int rs,ls,fa;
    ll val;
    int d=-1;
}tree[N];
void init(int x,ll v)
{
    tree[x].val=v;
    tree[x].d=tree[0].d+1;
    tree[x].fa=tree[x].ls=tree[x].rs=0;
}
int merge(int x,int y)
{
    if(!x||!y)return x|y;//存在空堆
    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
    tree[x].rs=merge(tree[x].rs,y);  //合并根的右儿子和 y
    if(tree[tree[x].rs].d>tree[tree[x].ls].d)swap(tree[x].rs,tree[x].ls);//维护左偏性
    tree[x].d=tree[tree[x].rs].d+1;  //距离增加
    tree[tree[x].rs].fa=x;  //更新父子关系
    return x;
}
void pushup(int x)
{
    if(!x)return;  //无可更新
    if(tree[tree[x].rs].d>tree[tree[x].ls].d)swap(tree[x].rs,tree[x].ls);//维护左偏性
    if(tree[x].d!=tree[tree[x].rs].d+1)  //维护距离
    {
        tree[x].d=tree[tree[x].rs].d+1;
        pushup(tree[x].fa);  //向上更新
    }
}
```

```
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      fa[tree[fa1].ls]=fa[tree[fa1].rs]=fa[fa1]=merge(tree[fa1].ls,tree[fa1].rs);
76      tree[fa[tree[fa1].ls]].fa=0;
77      return ans;
78  }
79  void solve()
80  {
81      int n,m;
82      cin>>n>>m;
83      a.resize(n+10);
84      tree[0].d=-1;
85      for(int i=1;i<=n;i++)
86      {
87          cin>>a[i];
88          init(i,a[i]);
89      }
90      for(int i=1;i<=n;i++)
91      {
92          fa[i]=i;
93      }
94      while(m--)
95      {
96          int op;
97          cin>>op;
98          if(op==1)
99          {
100             int x,y;
101             cin>>x>>y;
102             uni(x,y);
103         }
104         else if(op==2)
105         {
106             int x;
107             cin>>x;
```

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

2.14 平衡树.h

```
const int N=1e5+100;
struct node{
  int s[2];
  int v,p,cnt,sz;
  void init(int p1,int v1){
    p=p1;v=v1;
    cnt=sz=1;
  }
}tr[N];
int root=0,idx=0;
void pushup(int x){
  tr[x].sz=tr[x].cnt+tr[tr[x].s[1]].sz+tr[tr[x].s[0]].sz;
}
void rotate(int x){
  int y=tr[x].p;
  int z=tr[y].p;
  int k=tr[y].s[1]==x;
  tr[y].s[k]=tr[x].s[k^1];
  tr[tr[x].s[k^1]].p=y;
  tr[z].s[tr[z].s[1]==y]=x;
  tr[x].p=z;
  tr[y].p=x;
  tr[x].s[k^1]=y;
  pushup(y);pushup(x);
}
void splay(int x,int k){s
  while(tr[x].p!=k){
    int y=tr[x].p;
    int z=tr[y].p;
    if(z!=k) (tr[y].s[0]==x)^(tr[z].s[0]==y) ? rotate(x) : rotate(y);
    rotate(x);
  }
  if(k==0) root=x;
}
void find(int v){
  int x=root;
  while(tr[x].v!=v && tr[x].s[v>tr[x].v] ) x=tr[x].s[v>tr[x].v];
  splay(x,0);
}
int get_pre(int v){
  find(v);
  int x=root;
  if(tr[x].v<v) return x;
  x=tr[x].s[0];
  while(tr[x].s[1]) x=tr[x].s[1];
  splay(x,0);
  return x;
}
int get_suc(int v){
  find(v);
  int x=root;
  if(tr[x].v>v) return x;
  x=tr[x].s[1];
  while(tr[x].s[0]) x=tr[x].s[0];
  splay(x,0);
  return x;
```

```
}
void del(int v){
  int pre=get_pre(v);
  int suc=get_suc(v);
  splay(pre,0);splay(suc,pre);
  int d=tr[suc].s[0];
  if(tr[d].cnt>1){
    tr[d].cnt--;splay(d,0);
  }
  else{
    tr[suc].s[0]=0;splay(suc,0);
  }
}
void insert(int v){
  int x=root;
  int p=0;
  while(x && tr[x].v!=v){
    p=x;x=tr[x].s[v>tr[x].v];
  }
  if(x) tr[x].cnt++;
  else{
    x=++idx;
    tr[p].s[v>tr[p].v]=x;
    tr[x].init(p,v);
  }
  splay(x,0);
}
int get_rank(int v){
  insert(v);
  int res=tr[tr[root].s[0]].sz;
  del(v);
  return res;
}
int get_val(int k){
  int x=root;
  while(1){
    int y=tr[x].s[0];
    if(tr[x].cnt+tr[y].sz<k){
      k-=tr[y].sz+tr[x].cnt;
      x=tr[x].s[1];
    }
    else{
      if(tr[y].sz>=k) x=tr[x].s[0];
      else break;
    }
  }
  splay(x,0);
  return tr[x].v;
}
```

2.15 树状数组.h

```cpp
//区间修改，区间查询 如果 N=4000，内存大概在 300 多 MB 注意空间
#define int long long
const int N=2050;
int t1[N][N],t2[N][N],t3[N][N],t4[N][N];
int n,m;
int lowbit(int x){
  return x&-x;
}
void add(int x,int y,int val){
  for(int i=x;i<=n;i+=lowbit(i)){
    for(int j=y;j<=m;j+=lowbit(j)){
      t1[i][j]+=val;
      t2[i][j]+=val*x;
      t3[i][j]+=val*y;
      t4[i][j]+=val*x*y;
    }
  }
}
void range_add(int xa,int ya,int xb,int yb,int val){
  add(xa,ya,val); add(xa,yb+1,-val);
  add(xb+1,ya,-val); add(xb+1,yb+1,val);
}
int ask(int x,int y){
  int ans=0;
  for(int i=x;i;i-=lowbit(i)){
    for(int j=y;j;j-=lowbit(j)){
      ans+=(x+1)*(y+1)*t1[i][j]-(y+1)*t2[i][j]-(x+1)*t3[i][j]+t4[i][j];
    }
  }
  return ans;
}
int range_ask(int xa,int ya,int xb,int yb){
  return ask(xb,yb)-ask(xb,ya-1)-ask(xa-1,yb)+ask(xa-1,ya-1);
}
void solve(){
  cin>>n>>m;
  int op;
  while(cin>>op){
    if(op==1){
      int a,b,c,d,val;
      cin>>a>>b>>c>>d>>val;
      range_add(a,b,c,d,val);
    }
    else{
      int a,b,c,d;
      cin>>a>>b>>c>>d;
      cout<<range_ask(a,b,c,d)<<"\n";
    }
  }
}


// 点修 区间查询 可以建一个数组，可以开更大的 N，防止 MLE
#define int long long
const int N=4100;
int t[N][N];
```

```cpp
int n,m;
int lowbit(int x){
  return x&-x;
}
void add(int x,int y,int val){
  for(int i=x;i<=n;i+=lowbit(i)){
    for(int j=y;j<=m;j+=lowbit(j)){
      t[i][j]+=val;
    }
  }
}
int sum(int x,int y){
  int ans=0;
  for(int i=x;i;i-=lowbit(i)){
    for(int j=y;j;j-=lowbit(j)){
      ans+=t[i][j];
    }
  }
  return ans;
}
int ask(int x1,int y1,int x2,int y2){
  return sum(x2,y2)-sum(x2,y1-1)-sum(x1-1,y2)+sum(x1-1,y1-1);
}
void solve(){
  cin>>n>>m;
  int op;
  while(cin>>op){
    if(op==1){
      int x,y,val;
      cin>>x>>y>>val;
      add(x,y,val);
    }
    else{
      int a,b,c,d;
      cin>>a>>b>>c>>d;
      cout<<ask(a,b,c,d)<<"\n";
    }
  }
}

// 第 k 大
int t[N];
int kth(int k){
  int ans=0,x=0;
  for(int i=log2(n);~i;i--){
    x+=(1<<i);
    if(x>=n||sum+t[x]>=k) x-=(1<<i);
    else sum+=t[x];
  }
  return x+1;
}

// 区间修改 单点查询
const int N=4100;
int t[N][N];
int n,m;
int lowbit(int x){
  return x&-x;
```

```
}
void add(int x,int y,int val){
  for(int i=x;i<=n;i+=lowbit(i)){
    for(int j=y;j<=m;j+=lowbit(j)){
      t[i][j]+=val;
    }
  }
}
void range_add(int x1,int y1,int x2,int y2,int val){
  add(x1,y1,val); add(x1,y2+1,-val);
  add(x2+1,y1,-val); add(x2+1,y2+1,val);
}
int sum(int x,int y){
  int ans=0;
  for(int i=x;i;i-=lowbit(i)){
    for(int j=y;j;j-=lowbit(j)){
      ans+=t[i][j];
    }
  }
  return ans;
}
```

3 GEO

3.1 Rotating_Calipers.h

```cpp
//Rotating_Calipers
template<typename VALUE_TYPE>
class Rotating_Calipers
{
public:
    using pv = pair<VALUE_TYPE, VALUE_TYPE>;
    using vec_pv = vector<pair<VALUE_TYPE, VALUE_TYPE>>;
    vec_pv p;

    static VALUE_TYPE cross(pv p1, pv p2, pv p0)
    {
        pv t1 = {p1.fi - p0.fi, p1.se - p0.se},
           t2 = {p2.fi - p0.fi, p2.se - p0.se};
        return t1.fi * t2.se - t1.se * t2.fi;
    }

    static VALUE_TYPE dis(const pv &p1,const pv &p2){
        return (p1.fi - p2.fi) * (p1.fi - p2.fi) + (p1.se - p2.se) * (p1.se - p2.se);
    };

public:

    Rotating_Calipers() {}

    Rotating_Calipers(vec_pv _A) {
        build(_A);
    }

    void build(const vec_pv & _A) {
        p = ConvexHull(_A);
    }

    static vec_pv ConvexHull(vec_pv A, VALUE_TYPE flag = 1)
    {
        int n = A.size();
        if (n <= 2) return A;
        vec_pv ans(n * 2);
        sort(A.begin(), A.end(),
        [](pv a,pv b) -> bool {
            if(fabs(a.fi - b.fi) < 1e-10)
                return a.se < b.se;
            else return a.fi < b.fi;}    );
        int now = -1;
        for (int i = 0; i < n; i++)
        { // 维护下凸包
            while (now > 0 && cross(A[i], ans[now], ans[now - 1]) < flag) now--;
            ans[++now] = A[i];
        }
        int pre = now;
        for (int i = n - 2; i >= 0; i--)
        { // 维护上凸包
```

```
            while (now > pre && cross(A[i], ans[now], ans[now - 1]) < flag)
now--;
            ans[++now] = A[i];
        }
        ans.resize(now);
        return ans;
    }

    VALUE_TYPE getDiameter() {
        int j = 1;
        VALUE_TYPE ans = 0;
        int m = p.size();
        p.push_back(p[0]);
        for(int i = 0;i < m;i ++)
        {
            while( cross(p[i+1],p[j],p[i]) > cross(p[i+1],p[j+1],p[i]) ) j =
(j+1)%m;
            ans = max(ans, max( dis(p[i],p[j]) , dis(p[i+1],p[j]) ) );
        }
        p.pop_back();
        return ans;
    }

    VALUE_TYPE getPerimeter() {
        VALUE_TYPE sum = 0;
        p.pb(p[0]);
        for(int i = 0;i < (int)p.size() - 1;i ++)
        {
            sum += sqrtl(dis(p[i],p[i+1]));
        }
        p.pop_back();
        return sum;
    }

};
```

3.2 三维封装.h

```cpp
using ld = long double;

struct Point3
{
    ld x, y, z;
    Point3(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
    Point3 &operator+=(Point3 p) &
    {
        return x += p.x, y += p.y, z += p.z, *this;
    }
    Point3 &operator-=(Point3 p) &
    {
        return x -= p.x, y -= p.y, z -= p.z, *this;
    }
    Point3 &operator*=(Point3 p) &
    {
        return x *= p.x, y *= p.y, z *= p.z, *this;
    }
    Point3 &operator*=(ld t) &
    {
        return x *= t, y *= t, z *= t, *this;
    }
    Point3 &operator/=(ld t) &
    {
        return x /= t, y /= t, z /= t, *this;
    }
    friend Point3 operator+(Point3 a, Point3 b) { return a += b; }
    friend Point3 operator-(Point3 a, Point3 b) { return a -= b; }
    friend Point3 operator*(Point3 a, Point3 b) { return a *= b; }
    friend Point3 operator*(Point3 a, ld b) { return a *= b; }
    friend Point3 operator*(ld a, Point3 b) { return b *= a; }
    friend Point3 operator/(Point3 a, ld b) { return a /= b; }
    friend auto &operator>>(istream &is, Point3 &p)
    {
        return is >> p.x >> p.y >> p.z;
    }
    friend auto &operator<<(ostream &os, Point3 p)
    {
        return os << "(" << p.x << ", " << p.y << ", " << p.z << ")";
    }
};
using P3 = Point3;
struct Line3
{
    Point3 a, b;
};
using L3 = Line3;
struct Plane
{
    Point3 u, v, w;
};

ld len(P3 p)
{ // 原点到当前点的距离计算
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}
```

```
P3 crossEx(P3 a, P3 b)
{ // 叉乘
    P3 ans;
    ans.x = a.y * b.z - a.z * b.y;
    ans.y = a.z * b.x - a.x * b.z;
    ans.z = a.x * b.y - a.y * b.x;
    return ans;
}
ld cross(P3 a, P3 b)
{
    return len(crossEx(a, b));
}
ld dot(P3 a, P3 b)
{ // 点乘
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
P3 getVec(Plane s)
{ // 获取平面法向量
    return crossEx(s.u - s.v, s.v - s.w);
}
ld dis(P3 a, P3 b)
{ // 三维欧几里得距离公式
    ld val = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) * (a.z - b.z);
    return sqrt(val);
}
P3 standardize(P3 vec)
{ // 将三维向量转换为单位向量
    return vec / len(vec);
}

bool onLine(P3 p1, P3 p2, P3 p3)
{ // 三点是否共线
    return sign(cross(p1 - p2, p3 - p2)) == 0;
}
bool onLine(Plane s)
{
    return onLine(s.u, s.v, s.w);
}
bool onPlane(P3 p1, P3 p2, P3 p3, P3 p4)
{ // 四点是否共面
    ld val = dot(getVec({p1, p2, p3}), p4 - p1);
    return sign(val) == 0;
}
bool pointOnSegment(P3 p, L3 l)
{
    return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) <= p.x &&
           p.x <= max(l.a.x, l.b.x) && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y) &&
           min(l.a.z, l.b.z) <= p.z && p.z <= max(l.a.z, l.b.z);
}
bool pointOnSegmentEx(P3 p, L3 l)
{ // pointOnSegment 去除端点版
    return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) < p.x &&
           p.x < max(l.a.x, l.b.x) && min(l.a.y, l.b.y) < p.y && p.y < max(l.a.y, l.b.y) &&
           min(l.a.z, l.b.z) < p.z && p.z < max(l.a.z, l.b.z);
```

```
}
bool pointOnSegmentSide(P3 p1, P3 p2, L3 l)
{
    if (!onPlane(p1, p2, l.a, l.b))
    { // 特判不共面
        return 0;
    }
    ld val = dot(crossEx(l.a - l.b, p1 - l.b), crossEx(l.a - l.b, p2 - l.b));
    return sign(val) == 1;
}
bool pointOnPlaneSide(P3 p1, P3 p2, Plane s)
{
    ld val = dot(getVec(s), p1 - s.u) * dot(getVec(s), p2 - s.u);
    return sign(val) == 1;
}
bool lineParallel(L3 l1, L3 l2)
{ // 平行
    return sign(cross(l1.a - l1.b, l2.a - l2.b)) == 0;
}
bool lineVertical(L3 l1, L3 l2)
{ // 垂直
    return sign(dot(l1.a - l1.b, l2.a - l2.b)) == 0;
}
```

3.3 距离转化.typ

3.3.0.1 曼哈顿距离

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

3.3.0.2 欧几里得距离

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

3.3.0.3 切比雪夫距离

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

3.3.0.4 闵可夫斯基距离

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

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

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

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

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

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:

$$\begin{aligned}
\text{曼哈顿距离}\Rightarrow\text{切比雪夫距离}&:\\
(x,y)&\Rightarrow(x+y,x-y)\\
\text{切比雪夫距离}\Rightarrow\text{曼哈顿距离}&:\\
(x,y)&\Rightarrow\left(\frac{x+y}{2},\frac{x-y}{2}\right)
\end{aligned} \tag{7}$$

3.4 跨立实验.h

```cpp
using pii = pair<int, int>

#define fi first
#define se second
    const long double EPS = 1e-9;

template <class T>
int sign(T x)
{
    if (-EPS < x && x < EPS)
        return 0;
    return x < 0 ? -1 : 1;
}

// 叉乘
template <class T>
T cross(pair<T, T> a, pair<T, T> b)
{
    return a.fi * b.se - a.se * b.fi;
}

// 二维快速跨立实验
template <class T>
bool segIntersection(pair<T, T> l1, pair<T, T> l2)
{
    auto [s1, e1] = l1;
    auto [s2, e2] = l2;
    auto A = max(s1.fi, e1.fi), AA = min(s1.fi, e1.fi);
    auto B = max(s1.se, e1.se), BB = min(s1.se, e1.se);
    auto C = max(s2.fi, e2.fi), CC = min(s2.fi, e2.fi);
    auto D = max(s2.se, e2.se), DD = min(s2.se, e2.se);
    return A >= CC && B >= DD && C >= AA && D >= BB &&
           sign(cross(s1, s2, e1) * cross(s1, e1, e2)) == 1 &&
           sign(cross(s2, s1, e2) * cross(s2, e2, e1)) == 1;
}

//三维线段交点，需要 P3 封装，不相交返回{0,{}}
pair<bool, P3> lineIntersection(L3 l1, L3 l2)
{
    if (!onPlane(l1.a, l1.b, l2.a, l2.b) || lineParallel(l1, l2))
    {
        return {0, {}};
    }
    auto [s1, e1] = l1;
    auto [s2, e2] = l2;
    ld val = 0;
    if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 0, 1}))
    {
        val = ((s1.x - s2.x) * (s2.y - e2.y) - (s1.y - s2.y) * (s2.x - e2.x)) /
              ((s1.x - e1.x) * (s2.y - e2.y) - (s1.y - e1.y) * (s2.x - e2.x));
    }
    else if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 1, 0}))
    {
        val = ((s1.x - s2.x) * (s2.z - e2.z) - (s1.z - s2.z) * (s2.x - 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 -
   e2.y)) /
60                 ((s1.y - e1.y) * (s2.z - e2.z) - (s1.z - e1.z) * (s2.y - e2.y));
61     }
62     return {1, s1 + (e1 - s1) * val};
63 }
```

4 GRAPH

4.1 2-SAT.h

```cpp
#include <bits/stdc++.h>
using namespace std;


using pii = pair<int,int>;
class graph {
public:
    vector<vector<pii>> cnj;
    vector<int> dfn,scc,sz;
    int sc;
    int n;

    graph() { }

    graph(int _n) {
        n = _n;
        cnj.resize(n+1);
    };

    graph(vector<vector<int>> _cnj){

    }
private:
    reset(auto &t){t.clear();t.resize(n+1);};
public:
    void addOrdEdge(int u,int v,int w = 1) {

    }

    void addUnordEdge(int u,int v,int w = 1) {

    }

    //tarjin 缩点
    void runSCC(){
        reset(dfn),reset(scc),reset(sz);
        vector<int> low(n+1),s,vst(n+1),sz(n+1);
        sc = 0;
        int tp = 0,dfncnt = 0;
        using itf = function<void(int)>;
        itf dfs = [&](int u) -> void {
            low[u] = dfn[u] = ++dfncnt;
            s.pb(u),vst[u] = 1;
            for(auto [v,_]:cnj[u])
            {
                if(!dfn[v])
                {
                    dfs(v);
                    low[u] = min(low[u],low[v]);
                } else if(vst[v]){
                    low[u] = min(low[u],dfn[v]);
                }
            }
            if(dfn[u] == low[u]){
                sc ++;
```

```
                while(s.back()!=u) {
                    scc[s.back()] = sc;
                    sz[sc]++;
                    vst[s.back()] = 0;
                    s.pop_back();
                }
                scc[s.back()] = sc;
                sz[sc]++;
                vst[s.back()] = 0;
                s.pop_back();
            }
        };
        for(int i = 1;i <= n;i ++)
            if(!dfn[i]) dfs(i);
    }

    /**
     * @param method:string
     * Kruskal:O(m log m) for 稠密图
     * Prim:O((n+m) log n) for 稀疏图
     * @return vector<vector<int>> 生成树邻接表
     */
    vector<vector<pii>> runMST(string method = "Kruskal"){
        if(method == "Kruskal") {

        } else if(method == "Prim"){

        }
    }


    vector<pii> findBridge(string method = "tarjin"){

    }

    vector<int> findCut(string method = "tarjin"){

    }

    void eraseEdge(vector<pii> lst) {

    }

    void eraseVertice(vector<int> lst) {

    }
};

class two_SAT{
public:
    graph g;
    int n;
    two_SAT(int _n) {
        assert(_n%2 == 0);
        g = graph(_n);
        n = _n;
    }

    int tr(int p){
    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

```cpp
/*************************
* 二分图匹配(Hopcroft-Carp 算法)
* 复杂度 O (sqrt(n) * E)
* 邻接表存图
* Nx 为左端的端点数,使用前先赋值
*************************/
const int N = 3005, INF = 0x3f3f3f3f;
vector<int> G[N];  //邻接表
int Nx,Ny,k;  //x、y 部的点数、k 边数
int Mx[N],My[N];  //记录匹配的点的序号
int dx[N],dy[N];  //标记数组
int dis,u,v;
bool used[N];
bool bfs()
{
    queue<int> Q;
    dis = INF;  //初始化增广路的最短长度
    memset(dx,-1,sizeof(dx)); //初始化标记
    memset(dy,-1,sizeof(dy));
    for(int i=1;i<=Nx;++i)
    {
        if(Mx[i] == -1)
        {
            Q.push(i), dx[i] = 0;//将 x 部未匹配的点加入到队列中
        }
    }
    while(!Q.empty())
    {
        int u = Q.front();Q.pop();
        if(dx[u] > dis) break;
        for(auto v:G[u])
        {
            if(dy[v] == -1)   //取未标记过的 y 部的点
            {
                dy[v] = dx[u] + 1;  //做上标记
                if(My[v] == -1) dis = dy[v];  //若点 v 未匹配，此即为最短增广路
                else dx[My[v]] = dy[v] + 1, Q.push(My[v]);  //若已经匹配，则将 v 的匹配点纳入到增广路，并入队
            }
        }
    }
    return dis != INF;  //若 dis==inf，说明无增广路
}
bool DFS(int u)
{
    for(auto v:G[u])
    {
        if(!used[v] && dy[v] == dx[u] + 1)//点 u 只能和增广路上的下一个点匹配
        {//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
            used[v] = true;
            if(My[v] != -1 && dy[v] == dis) continue; //若 v 匹配过，但是是增广路的终点，则忽略它
            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

```cpp
#include <bits/stdc++.h>
using namespace std;
using pii = pair<int,int>;
class graph {
public:
using tii = array<int,3>;
    vector<vector<pii>> cnj;
    vector<pii> links;
    vector<tii> edges;
    vector<vector<int>> bcc, h;
    vector<int> dfn, scc, sz, fa;
    int sc,dc;
    int n;
    graph(int _n) {
        n = _n;
        reset(cnj);
        reset(h);
    };

    graph(vector<vector<pii>> _cnj) {
        n = _cnj.size();
        cnj = _cnj;
    }

private:
    void reset(auto &t) {
        t.clear();
        t.resize(n + 1);
    };

public:
    void addOrdEdge(int u, int v, int w = 1) {
        cnj[u].push_back({v, w});
        edges.pb({u,v,w});
    }

    void addUnordEdge(int u, int v, int w = 1) {
        cnj[u].push_back({v, w});
        cnj[v].push_back({u, w});
        //邻接表↑--链式前向星↓
        links.pb({u,v});
        h[u].pb(links.size()-1);
        links.pb({v,u});
        h[v].pb(links.size()-1);
        edges.pb({u,v,w});
        edges.pb({v,u,w});
    }

    // 强连通分量缩点
    void runSCC() {
        reset(dfn), reset(scc), reset(sz);
        vector<int> low(n + 1), s, vst(n + 1), sz(n + 1);
        sc = 0;
        int tp = 0, dfncnt = 0;
        using itf = function<void(int)>;
        itf dfs = [&](int u) -> void
        {
```

```
            low[u] = dfn[u] = ++dfncnt;
            s.pb(u), vst[u] = 1;
            for (auto [v, _] : cnj[u])
            {
                if (!dfn[v])
                {
                    dfs(v);
                    low[u] = min(low[u], low[v]);
                }
                else if (vst[v])
                {
                    low[u] = min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u])
            {
                sc++;
                while (s.back() != u)
                {
                    scc[s.back()] = sc;
                    sz[sc]++;
                    vst[s.back()] = 0;
                    s.pop_back();
                }
                scc[s.back()] = sc;
                sz[sc]++;
                vst[s.back()] = 0;
                s.pop_back();
            }
        };
        for (int i = 1; i <= n; i++)
            if (!dfn[i])
                dfs(i);
    }

    /**双联通分量缩点
     * E: 边双: 支持重边
     * D: 点双
     */
    void runBCC(char mod = 'E') {
        reset(bcc),reset(sz);
        if(mod == 'E') {
            auto [ib,_] = this->findBridge();
            using itf = function<void(int,int)>;
            dc = 0;
            itf dfs = [&](int u,int tv) -> void {
                bcc[u] = {dc};
                for(auto rv:h[u]){
                    int v = links[rv].se;
                    if(ib[rv] || bcc[v].size() || rv == (tv^1)) continue;
                    else dfs(v,rv);
                }
            };
            for(int i = 1;i <= n;i ++) if(!bcc[i].size()) ++dc,dfs(i,0);
        } else if(mod == 'D') {
            auto ic = this->findCut();
            using itf = function<void(int,int)>;
```

```
            dc = 0;
            itf dfs = [&](int u,int f) -> void {
                for(auto [v,_]:cnj[u]) {
                    if(v == f || (!ic[v] && bcc[v].size())) continue;
                    else if(ic[v] || bcc[v].size()) bcc[v].pb(dc);
                    else bcc[v].pb(dc),dfs(v,u);
                }
            };
            for(int i = 1;i <= n;i ++) if(!bcc[i].size()) +
+dc,bcc[i].pb(dc),dfs(i,0);
        } else assert(0);
    }

    /**最小生成树，默认 Kruskal
     * Kruskal:O(m log m) for 稀疏图
     * Prim:O((n+m) log n) for 稠密图
     * 返回邻接表
     */
    vector<vector<pii>> runMST(string method = "Kruskal") {
        graph ng(n);
        using tii = array<int,3>;
        const int INF = 1e17;
        if (method == "Kruskal") {
            priority_queue<tii,vector<tii>,greater<tii>> q;
            DSU dsu(n+1);
            for(auto [u,v,w]:edges) q.push({w,u,v});
            while(q.size())
            {
                auto [w,u,v] = q.top();
                q.pop();
                if(!dsu.same(u,v)) {
                    ng.addUnordEdge(u,v,w);
                    dsu.merge(u,v);
                }
            }
            return ng.cnj;
        }
        else if (method == "Prim") {
            vector<int> dis(n+1,INF);
            using fii = array<int,4>;
            priority_queue<fii,vector<fii>,greater<fii>> q;
            q.push({0,0,1});
            dis[1] = 0;
            vector<int> LOCK(n+1);
            while(q.size())
            {
                auto [_,w,f,u] = q.top();
                q.pop();
                if(LOCK[u]) continue;
                else LOCK[u] = 1;
                if(f) ng.addUnordEdge(f,u,w);
                for(auto [v,wv]:cnj[u])
                {
                    if(dis[v] > dis[u] + wv) {
                        dis[v] = dis[u] + wv;
                        q.push({dis[v],wv,u,v});
                    }
```

```
                }
            }
            return ng.cnj;
        } else assert(0);
    }

    /**
     * tarjin 找桥
     * 返回{res,ib}中 ib[x]!=0 代表某点的出边 x 是桥，res 为桥的列表
     */
    pair<vector<int>,vector<pii>> findBridge() {
        reset(dfn),reset(fa);
        vector<int> low(n+1),ib(links.size()+2);
        vector<pii> res;
        int dfncnt = 0;
        using itf = function<void(int,int)>;
        itf dfs = [&](int u,int tv) -> void {
            fa[u] = tv;
            low[u] = dfn[u] = ++dfncnt;
            // for(auto [v,_]:cnj[u]) {
            for(auto rv:h[u]){
                int v = links[rv].se;
                if(!dfn[v]){
                    dfs(v,rv);
                    low[u] = min(low[u],low[v]);
                    if(low[v] > dfn[u]) ib[rv] = ib[rv^1] = 1,res.pb({u,v});
                } else if(dfn[v] < dfn[u] && rv != (tv^1))
                    low[u] = min(low[u],dfn[v]);
            }
        };
        for(int i = 1;i <= n;i++) if(!dfn[i]) dfs(i,0);
        return {ib,res};
    }

    // tarjin 找割点
    vector<int> findCut() {
        reset(dfn);
        vector<int> low(n+1),ic(n+1);
        vector<bool> vis(n+1);//能不能去掉?
        int dfscnt = 0;
        using itf = function<void(int,int)>;
        itf dfs = [&](int u,int f) {
            // cout << "et:" << u << " ";
            vis[u] = 1;
            low[u] = dfn[u] = ++dfscnt;
            int ch = 0;
            for(auto [v,_]:cnj[u]){
                if(!vis[v]){
                    ch ++;
                    dfs(v,u);
                    low[u] = min(low[u],low[v]);
                    if(f != u && low[v] >= dfn[u] && !ic[u]) ic[u] = 1;
                } else if(v != f) low[u] = min(low[u],dfn[v]);
            }
            if(f == u && ch >= 2 && !ic[u]) ic[u] = 1;
        };
        for(int i = 1;i <= n;i ++) if(!vis[i]) dfscnt = 0, dfs(i,i);
        // 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

```cpp
//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

//涉及：树的直径、二分图染色
//      貌似只能判定、不能求数量

struct node  //存边
{
  int u;
  int v;
  int id;
};
struct D  //用于求树的直径的结构体
{
  int d;   //直径   d = maxn + submaxn
  int maxn=0;    //最大子树结点数
  int submaxn=0;  //次大子树结点数
}d[N];
vector<node> edge;  //邻接表

int odd=0,even=0;  //是否存在奇/偶环

vector<int> e[N];  //存边
int col[N]={0};  //二分图染色法
int vis[N]={0};  //
int dfn[N]={0},cnt=0;  //深搜序

int fa[N]={0};   //并查集
int siz[N]={0};  //存连通块的直径

int pre[N]={0},cov[N]={0};  //前驱点、染色体
int pre_edge[N]={0};   //前驱边

int find(int u) //并查集
{
  if(fa[u]==u)return u;
  else return fa[u]=find(fa[u]);
}
void uni(int u,int v)
{
  int fa1=find(u);
  int fa2=find(v);
  fa[fa1]=min(fa1,fa2);
  fa[fa2]=min(fa1,fa2);
}

void dfs(int u,int fa)
{
  if(fa!=0)uni(u,fa);  //深搜时顺便建立父子关系

  dfn[u]=++cnt;   //记录深搜序
  col[u]=col[fa]^1;   //染色
  vis[u]++;     //搜过的点

  d[u].maxn=0;  //系列初始化
  d[u].submaxn=0;
```

```
  d[u].d=0;

  for(auto re:e[u])
  {
    int v=u^edge[re].u^edge[re].v;   //取儿子结点

    if(v==fa)continue;  //跳过父亲

    if(col[v]<0)   //儿子未染色
    {
      pre[re]=u;      //记录 边 的前驱点
      pre_edge[v]=re;  //记录 点 的前驱边
      dfs(v,u);

      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
      {
        d[u].submaxn=d[u].maxn;
        d[u].maxn=d[v].maxn+1;
      }
      else if(d[u].maxn==d[v].maxn+1)
      {
        d[u].submaxn=d[u].maxn;
      }
      else if(d[u].submaxn<d[v].maxn+1)
      {
        d[u].submaxn=d[v].maxn+1;
      }

    }
    else if(col[v]==col[u]^1) //找到偶环
    {
        even++;//记录
    }
    else if(col[v]==col[u])//找到奇环
    {
      odd++;//记录
      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
      //否则遇到了祖宗
      cov[re]++;    //标记这条边

      int pnt=u;//记录当前点

      int pe=pre_edge[u];//取出当前点的前驱边，准备走返祖边

      while(!even)    //若没有发现偶环，则尝试用两个奇环制造偶环
      {
        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

        cov[pe]++;  // 标记沿途的边

        pnt=pre[pe];// 取出 边 的前驱点
        pe=pre_edge[pnt];  //再取 前驱点 的 前驱边

        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
```

```
        }
      }
    }
    d[u].d=d[u].maxn+d[u].submaxn;  //求直径
  }
  void solve()
  {
    //这里要补初始化//
    edge.clear();
      int n,m;
      cin>>n>>m;

      edge.push_back((node){-1,-1,0});//这个是凑数用的
      pre[0]=-1;

      for(int i=1;i<=m;i++)//存边
    {
        int u,v;
        cin>>u>>v;
        edge.push_back((node){u,v,i});

        e[u].push_back(i);  //注意:邻接表存边号
        e[v].push_back(i);
    }

    for(int i=1;i<=n;i++)col[i]=-1,fa[i]=i;  //初始化
    col[0]=0;//0 点不存在，但初始化为 0 备用

    for(int i=1;i<=n;i++)//深搜，找环，顺便求连通块的直径
    {
      if(!vis[i])dfs(i,0);
    }

    for(int i=1;i<=n;i++)
    {
      int f=find(i);
      siz[f]=max(siz[f],d[i].d);  //更新连通块的直径
    }

    if(n<4)cout<<-1<<"\n"; //n<4 is -1

    else if(odd&&even)  //若有奇又有偶，直接输出
    {
      cout<<0<<"\n";
    }
    else if(even)   //仅有偶环，只需再加一条边
    {
      cout<<1<<"\n";
    }
    else // 若无偶环，则需要尝试用若干直径制造偶环
    {
      map<int,int> is;
      priority_queue<int> q;
      for(int i=1;i<=n;i++)  //找连通块
      {
        int f=find(i);
        if(!is[f])
        {
```

```
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

```cpp
//V1
//Dinic 算法，求解最大流
//O（M*N^2）
//是对 EK 算法的优化
//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
//先用 BFS 求分层图，操作和 EK 算法一样，但不计算，只分层
//然后用 DFS 在分层图上找增广路，并计算流量，更新残量图
//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
const ll M=1000000000000;
int n,m,s,t,si;
ll sum=0;
struct edge
{
    int to;
    int nex;
    ll w;
}a[500005];
int head[500005]={0},cnt=1;  //链式前向星相关
void add(int u,int v,ll w)  //链式前向星
{
    //a[++cnt].from=u;
    a[++cnt].to=v;
    a[cnt].w=w;
    a[cnt].nex=head[u];
    head[u]=cnt;

    //a[++cnt].from=v;   //regard edge
    a[++cnt].to=u;
    a[cnt].w=0;
    a[cnt].nex=head[v];
    head[v]=cnt;
}
ll dis[500005]={0};
int now[500005]={0};

int bfs() //在残量网络中构造分层图
{     //事实上，深度标记 dis[i]就是分层图，无穷深代表不在图内
    for(int i=1;i<=n;i++)dis[i]=M;//重置分层图
    queue<int> q;
    dis[s]=0;     //标记源点为 0 深度
    now[s]=head[s];
    q.push(s);
    while(!q.empty())
    {
        int temp=q.front();
        q.pop();
        for(int i=head[temp];i;i=a[i].nex)
        {
            int v=a[i].to;
            if(dis[v]!=M||a[i].w<=0)continue;
            q.push(v);
            dis[v]=dis[temp]+1;  //分层，标记深度

```

```
                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

                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
            }
        }
        return 0;
    }
    ll dfs(int x,ll delta)    //深搜求解 父节点->子节点  的流量
    {                         //源点的父节点->源点  就是最大流

        if(x==t)return delta;      //delta 最终会是某条路的最小权
        ll k,res=0;

        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
        {
            now[x]=i;//更新

            int v=a[i].to;//取儿子
            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
            k=dfs(v,min(a[i].w,delta));  //求出 x->v 的流量

            if(k==0)dis[v]=M;    //发现流量 x->v 等于 0，删点

            a[i].w-=k;        //更新容量
            a[i^1].w+=k;

            res+=k;         //更新总和
            delta-=k;     //更新剩余容量
        }
        return res;    //返回流入点 x 的流量
    }
    void Dinic()
    {
        while(bfs())  //bfs 构建分层图
        {
            sum+=dfs(s,M);     //dfs 找增广路并计算
        }
    }
    void solve()
    {
        cin>>n>>m>>s>>t;
        for(int i=1;i<=m;i++)
        {
            int u,v;
            ll w;
            cin>>u>>v>>w;
            add(u,v,w);
        }
        while(bfs())  //bfs 构建分层图
        {
            sum+=dfs(s,M);     //dfs 找增广路并计算
        }
        cout<<sum<<"\n";
    }
```

```
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,
    edgeList[_v].size() - 1);
147         edgeList[_v][edgeList[_v].size() - 1].setRevIdx(_u,
    edgeList[_u].size() - 1);
148     }
149
150     bool bfs()
151     {
152         queue<int> que;
153         for (auto x = dep.begin(); x != dep.end(); x++) (*x) = 0;
154         dep[S] = 1;
155         que.push(S);
156         while (que.size())
157         {
158             ll int at = que.front();
159             que.pop();
160             for (int i = 0; i < edgeList[at].size(); i++)
161             {
162                 auto tar = edgeList[at][i];
163                 if ((!dep[tar.nxt]) && (tar.flow < tar.cap))
164                 {
```

```
                    dep[tar.nxt] = dep[at] + 1;
                    que.push(tar.nxt);
                }
            }
        }
        return dep[T];
    }

    ll int dfs(int at, ll int flow)
    {
        if ((at == T) || (!flow))
            return flow; // 到了或者没了
        ll int ret = 0;  // 本节点最大流
        for (int &i = fir[at]; i < edgeList[at].size(); i++)
        {
            auto tar = edgeList[at][i];   // 目前遍历的边
            int tlFlow = 0;   // 目前边的最大流
            if (dep[at] == dep[tar.nxt] - 1) // 遍历到的边为合法边
            {
                tlFlow = dfs(tar.nxt, min((ll)tar.cap - tar.flow, flow - ret));
                if (!tlFlow)
                    continue;   // 若最大流为 0，什么都不做
                ret += tlFlow;   // 本节点最大流累加
                edgeList[at][i].flow += tlFlow;  // 本节点实时流量
                edgeList[tar.revNodeIdx.first][tar.revNodeIdx.second].flow -= tlFlow; // 反向边流量
                if (ret == flow)
                    return ret; // 充满了就不继续扫了
            }
        }
        return ret;
    }

    ll int dinic()
    {
        if (maxFlowAns)
            return maxFlowAns;
        while (bfs())
        {
            for(auto x = fir.begin();x != fir.end();x ++) (*x) = 0;
            maxFlowAns += dfs(S, INT_MAX);
        }
        return maxFlowAns;
    }
};
```

4.5.3 DJ.h

```
int n,m,s,t;
ll max_flow=0,min_cost=0;
struct p3381
{
    int to;
    int nex;
    ll w;
    ll c;
}a[200005];
int head[5005]={0},cnt=1,x[5005][2]={0};
void add(int u,int v,ll w,ll c)
{
    a[++cnt].to=v;
    a[cnt].nex=head[u];
    a[cnt].w=w;
    a[cnt].c=c;
    head[u]=cnt;

    a[++cnt].to=u;
    a[cnt].nex=head[v];
    a[cnt].w=0;
    a[cnt].c=-c;
    head[v]=cnt;
}
ll dist[5005]={0};   //最小费用数组
ll h[5005]={0};    //势数组，dj 不能跑负边，所以要操作
struct node
{
    int id;
    bool operator < (const node &x) const {return dist[x.id]<dist[id];}
};
////////////////////////////////////
bool DJ()
{
    int vis[5005]={0};
    priority_queue<node> q;
    q.push({s});
    for(int i=1;i<=n;i++)dist[i]=inf;
    dist[s]=0;
    while(!q.empty())
    {
        node temp=q.top();
        q.pop();
        vis[temp.id]=0;
        for(int i=head[temp.id];i;i=a[i].nex)
        {
            int v=a[i].to;
            if(dist[v]>dist[temp.id]+a[i].c+h[temp.id]-h[v]&&a[i].w>0)
            {
                dist[v]=dist[temp.id]+a[i].c+h[temp.id]-h[v];
                if(!vis[v])
                {
                    vis[v]=1;
                    q.push({v});
                }
                x[v][0]=temp.id;
```

```
                x[v][1]=i;
            }
        }
    }
    if(dist[t]==inf)return false;
    return true;
}
void EK()//EK 算法求解
{
    ll cost=0,minn=inf;
    for(int p=t;p!=s;p=x[p][0])
    {
        minn=min(minn,a[x[p][1]].w);
    }
    for(int p=t;p!=s;p=x[p][0])
    {
        a[x[p][1]].w-=minn;
        a[x[p][1]^1].w+=minn;
    }
    cost=(dist[t]-(h[s]-h[t]))*minn;
    min_cost+=cost;
    max_flow+=minn;
    for(int i=1;i<=n;i++)h[i]+=dist[i];
}
void solve()
{
    cin>>n>>m>>s>>t;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        ll w,c;
        cin>>u>>v>>w>>c;
        add(u,v,w,c);
    }
    while(DJ())
    {
        EK();
    }
    cout<<max_flow<<" "<<min_cost<<"\n";
}
```

4.5.4 EK.h

```cpp
int n,m,s,t;
ll max_flow=0,min_cost=0;
struct p3381
{
    int to;
    int nex;
    ll w;
    ll c;
}a[200005];
int head[5005]={0},cnt=1,x[5005][2]={0};
void add(int u,int v,ll w,ll c)
{
    a[++cnt].to=v;
    a[cnt].nex=head[u];
    a[cnt].w=w;
    a[cnt].c=c;
    head[u]=cnt;

    a[++cnt].to=u;
    a[cnt].nex=head[v];
    a[cnt].w=0;
    a[cnt].c=-c;
    head[v]=cnt;
}
ll dist[5005]={0};   //最小费用数组
////////////////////////////////////
//基于 EK 算法的最小费用最大流
//用 SPFA 对费用求增广路
//用 EK 算法中的更新操作求解
bool SPFA() //SPFA 找增广路
{
    int vis[5005]={0};
    queue<int> q;
    q.push(s);
    for(int i=1;i<=n;i++)dist[i]=inf;
    dist[s]=0;
    while(!q.empty())
    {
        int temp=q.front();
        q.pop();
        vis[temp]=0;
        for(int i=head[temp];i;i=a[i].nex)
        {
            int v=a[i].to;
            if(dist[v]>dist[temp]+a[i].c&&a[i].w>0)
            {
                dist[v]=dist[temp]+a[i].c;
                if(!vis[v])
                {
                    vis[v]=1;
                    q.push(v);
                }
                x[v][0]=temp;
                x[v][1]=i;
            }
        }
```

```
    }
    if(dist[t]==inf)return false;
    return true;
}
void EK()//EK 算法求解
{
    ll cost=0,minn=inf;
    for(int p=t;p!=s;p=x[p][0])
    {
        minn=min(minn,a[x[p][1]].w);
    }
    for(int p=t;p!=s;p=x[p][0])
    {
        a[x[p][1]].w-=minn;
        a[x[p][1]^1].w+=minn;
    }
    cost=dist[t]*minn;
    min_cost+=cost;
    max_flow+=minn;
}
void solve()
{
    cin>>n>>m>>s>>t;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        ll w,c;
        cin>>u>>v>>w>>c;
        add(u,v,w,c);
    }
    while(SPFA())
    {
        EK();
    }
    cout<<max_flow<<" "<<min_cost<<"\n";
}
```

4.5.5 HLPP.h

```cpp
//比 ISAP 和 DINIC 快一点，但不稳定 O(根(m)*n^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
//且节点会伺机将超额量推向深度低的节点
int n,m,s,t;
int vis[1500]={0};
int head[1500]={0},cnt=1;
int h[1500]={0},gap[1500]={0};
int sum=0;
int e[1500]={0};
struct p4722
{
    int to,nex;
    int w;
}a[540005];
struct node
{
    bool operator() (int x,int y)const{return h[x]<h[y];}
};
priority_queue<int,vector<int>,node> q;
queue<int> que;
void add(int u,int v,int w)   //链式前向星
{
    a[++cnt].to=v;
    a[cnt].nex=head[u];
    a[cnt].w=w;
    head[u]=cnt;
    a[++cnt].to=u;
    a[cnt].nex=head[v];
    a[cnt].w=0;
    head[v]=cnt;
}
void bfs_rebel()  //深度标签初始化，从汇点广搜
{
    que.push(t);
    h[t]=0;
    while(!que.empty())
    {
        int temp=que.front();
        que.pop();
        gap[h[temp]]++;
        for(int i=head[temp];i;i=a[i].nex)
        {
            int v=a[i].to;
            if(h[v]==0&&v!=t&&a[i^1].w)
            {
                h[v]=h[temp]+1;
                que.push(v);
            }
        }
    }
}
void push(int u)   //推流
{
    int i;
```

```
    for(i=head[u];i;i=a[i].nex)
    {
        int v=a[i].to;
        if(a[i].w<=0||h[u]!=h[v]+1)continue;//跳过 0 容边和深度不递减的点

        int f=min(e[u],a[i].w); //取残留容量和超额量中的小者

        e[v]+=f;    //推流，更新
        e[u]-=f;
        a[i].w-=f;
        a[i^1].w+=f;

        if(!vis[v]&&v!=t&&v!=s)  //子节点返回队列
        {
            vis[v]=1;
            q.push(v);
        }

        if(e[u]==0)break;   //超额量分发完毕退出函数
    }
}
void rebel(int u)   //重贴标签
{
    int i;
    h[u]=inf;   //标签初始化
    for(i=head[u];i;i=a[i].nex)
    {
        int v=a[i].to;
        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 下一次恰好可以给它推流
    }
}
void hlpp()   //核心函数
{
    h[s]=n;e[s]=inf; //源点 s 深度为 n，超额量为无穷
    for(int i=head[s];i;i=a[i].nex)  //源点不入队，因此在外处理
    {
        int v=a[i].to;
        if(int f=a[i].w)
        {
            a[i].w-=f;     //更新网络
            a[i^1].w+=f;
            e[s]-=f;
            e[v]+=f;
            if(v!=s&&v!=t&&!vis[v])  //注意汇点和重复点不可入队
            {
                q.push(v);
                vis[v]=1;
            }
        }
    }
    while(!q.empty()) //计算最大流
    {
        int u=q.top();
        q.pop();
        vis[u]=0;  //出队取消标记
        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

```cpp
int n,m;
struct p3376
{
    int to;
    ll w=0;
    int nex;
}a[540005];
map<pair<int,int>,int> key;
int head[1580]={0},cnt=1,dept[1580]={0};
void add(int u,int v,ll w)
{
    if(key[{u,v}]!=0)     //去除重边
    {
        a[key[{u,v}]].w+=w;
        return;
    }
    a[++cnt].to=v;
    a[cnt].w=w;
    a[cnt].nex=head[u];
    key[{u,v}]=cnt;
    head[u]=cnt;
    a[++cnt].to=u;
    a[cnt].w=0;
    a[cnt].nex=head[v];
    key[{v,u}]=cnt;
    head[v]=cnt;
}
//.........................//
//Dinic 算法会调用太多次 bfs，考虑优化
//于是就有了 ISPA 算法
//从汇点开始 bfs，标记深度
//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
ll ans=0;
int g[1580]={0},maxn=0;

void bfs(int t)  //从汇点 bfs
{
    queue<int> q;
    bool vis[1580];
    memset(vis,0,sizeof(vis));
    q.push(t);
    vis[t]=true;
    //g[0]++;
    while(!q.empty())
    {
        int temp=q.front();
        q.pop();
        g[dept[temp]]++;
        maxn=max(dept[temp],maxn);
        //vis[temp]=true;
        for(int i=head[temp];i;i=a[i].nex)
        {
            int v=a[i].to;
            if(!vis[v])
            {
                vis[v]=true;
```

```
                dept[v]=dept[temp]+1;
                q.push(v);
            }
        }
    }
}
ll dfs(int p,int s,int t,ll tot)
{
    int i;
    if(p==t)  //到达汇点
    {
        ans+=tot;  //更新答案
        return tot; //返回
    }
    ll k=0,res=0;
    for(i=head[p];i&&tot;i=a[i].nex)
    {
        int v=a[i].to;
        if(dept[v]!=dept[p]-1||a[i].w<=0)continue;  //残量为 0，或非递减的深度，跳过
        k=dfs(v,s,t,min(a[i].w,tot)); //取 p->v 的流量

        a[i].w-=k; //更新残量
        a[i^1].w+=k;

        res+=k;  //增加总流量

        tot-=k;  //残量减少
    }
    if(tot>0)   //流过来的流量还有余，使深度增加
    {
        dept[p]++;
        g[dept[p]-1]--;
        g[dept[p]]++;
        if(g[dept[p]-1]<=0)dept[s]=n+1;     //出现断层，对 s 的深度进行标记
    }
    return res;
}
void isap(int s,int t)
{
    while(dept[s]<=n)dfs(s,s,t,100000000000);  //未出现断层就反复 dfs
}
void solve()
{
    int s,t;
    cin>>n>>m>>s>>t;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        ll w;
        cin>>u>>v>>w;
        add(u,v,w);
    }
    //ll temp;
    bfs(t);
    isap(s,t);
    cout<<ans<<"\n";
```

```
113 }
```

4.5.7 min_Cost.h

```
const int INF = 0x3f3f3f3f

class PD//AC
{
public:
    class edge
    {
    public:
        int v, f, c, next;
        edge(int _v,int _f,int _c,int _next)
        {
            v = _v,f = _f,c = _c,next = _next;
        }
        edge() { }
    } ;

    void vecset(int value,vector<int> &arr)
    {
        for(int i = 0;i < arr.size();i ++) arr[i] = value;
        return;
    }

    class node
    {
    public:
        int v, e;
    } ;

    class mypair
    {
    public:
        int dis, id;

        bool operator<(const mypair &a) const { return dis > a.dis; }

        mypair(int d, int x)
        {
            dis = d;
            id = x;
        }
    };

    vector<int> head,dis,vis,h;
    vector<edge> e;
    vector<node> p;
    int n, m, s, t, cnt = 1, maxf, minc;

    PD(int _n,int _m,int _s,int _t)
    {
        n = _n, m = _m,s = _s,t = _t;
        maxf = 0, minc = 0;
        head.resize(n+2), dis.resize(n+2),vis.resize(n+2);
        e.resize(2);
        h.resize(n+2), p.resize(m+2);
    }

    void addedge(int u, int v, int f, int c)
    {
```

```
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                     }
```

```
                }
            }
        }
    }

    int pd()
    {
        spfa();
        while (dijkstra())
        {
            int minf = INF;
            for (int i = 1; i <= n; i++)
                h[i] += dis[i];
            for (int i = t; i != s; i = p[i].v)
                minf = min(minf, e[p[i].e].f);
            for (int i = t; i != s; i = p[i].v)
            {
                e[p[i].e].f -= minf;
                e[p[i].e ^ 1].f += minf;
            }
            maxf += minf;
            minc += minf * h[t];
        }
        return 0;
    }

    pair<int,int> get()
    {
        return {maxf,minc};
    }
};
```

4.6 PATH

4.6.1 johonson.h

```cpp
// Johnson 算法

// 带负边的全源最短路

// 这个算法最重要的功能是使得 dj 能用来处理负边

// 只跑一次的话复杂度和 spfa 同阶
// 跑多次的话会比 spfa 优秀

// 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]
// 再将原边权 w(u,v),改造为 w(u,v)+h[u]-h[v]
// 再以每个点作为起点，跑 n 轮 dj 即可

// 最终 d'(s,t)=d(s,t)+h[s]-h[t]
#define ll long long
#define inf 0x3f3f3f3f
int main()
{
    void solve();
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    solve();
    return 0;
}
//Johnson 算法
//带负边的全源最短路
//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]
//再将原边权 w(u,v),改造为 w(u,v)+h[u]-h[v]
//再以每个点作为起点，跑 n 轮 dj 即可
int n,m;
struct p5905
{
    int to;
    ll w;
};
vector<p5905> edge[3005];
ll dis[3005]={0},h[3005]={0};
int cnt[3005]={0};
int vis[3005]={0};
queue<int> q;
void spfa()
{
    memset(h,inf,sizeof(h));
    h[0]=0;
    q.push(0);
    vis[0]++;
    while(!q.empty())
    {
        int t=q.front();
        q.pop();
        cnt[t]++;
        if(cnt[t]>n)
```

```
        {
            cout<<-1<<"\n";
            exit(0);
        }
        for(auto re:edge[t])
        {
            int v=re.to;
            ll w=re.w;
            if(h[v]>h[t]+w)
            {
                h[v]=h[t]+w;
                if(!vis[v])
                {
                    vis[v]++;
                    q.push(v);
                }
            }
        }
        vis[t]--;
    }
}
struct node
{
    int id;
    ll w;
    bool operator < (const node &x) const {return x.w<w;}
};
void dj(int f)
{
    memset(dis,inf,sizeof(dis));
    int vv[3005]={0};
    priority_queue<node> qq;
    dis[f]=0;
    qq.push((node){f,dis[f]});
    while(!qq.empty())
    {
        node temp=qq.top();
        qq.pop();
        int t=temp.id;
        ll w=temp.w;
        if(vv[t])continue;
        vv[t]++;
        dis[t]=w;
        for(auto re:edge[t])
        {
            int v=re.to;
            ll tw=re.w;
            if(vv[v])continue;
            if(dis[v]>w+tw)
            {
                dis[v]=w+tw;
                qq.push((node){v,dis[v]});
            }
        }
    }
}
void solve()
{
```

```
110     cin>>n>>m;
111     while(m--)
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

```cpp
#define ll long long
#define N 5050
const ll inf=1e17+7;
int n;
struct node  //边
{
  int v;
  ll w;
  int is=0;
};
struct ed  //用于优先队列
{
  int v;
  ll w;
  ll d;
  bool operator < (const ed x) const
  {
    return x.w<w;
  }
};
queue<ll> ans;  //答案队列
vector<node> e[N];  //邻接表
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
void dj(int s)
{
  priority_queue<ed> q;
  vector<int> vis(n+10,0);
  for(int i=1;i<=n;i++)d[i]=inf;//初始化
  d[s]=0;
  q.push((ed){s,d[s],0});  //终点入队
  while(!q.empty())
  {
    auto r=q.top();
    q.pop();
    int u=r.v;

    if(vis[u])continue;
    vis[u]=1;
    d[u]=r.w;

    for(auto re:e[u])
    {
      int v=re.v;
      ll w=re.w;
      if(re.is)continue;//仅搜反边
      if(vis[v]||d[v]<=w+d[u])continue;
      d[v]=w+d[u];  //可松弛
      q.push((ed){v,d[v],0});
    }
  }
}
void astar(int s,int t,ll &k) //A*算法
{
  priority_queue<ed> q;
  vector<int> vis(n+10,0);
```

```
    q.push((ed){s,d[s],0});  //起点入队
    while(!q.empty())
    {
      auto temp=q.top();
      q.pop();

      int u=temp.v;

      vis[u]++;//统计出队次数

      if(vis[t]<=k&&u==t)//若只求第 k 短，只需 vis[t]==k，记录并退出
      {
        ans.push(temp.d);
      }
      if(vis[t]==k)return;

      for(auto re:e[u])
      {
        if(!re.is)continue;
        if(vis[u]>k)continue;  //跳过出队次数大于 k 的

        int v=re.v;
        ll w=re.w;
        ll dis=temp.d;
        dis=dis+w;  //此为从起点到 v 的距离

        q.push((ed){v,d[v]+dis,dis});
        //d[v]+dis 是估计距离，dis 是起点到 v 的距离
        //按估计距离升序排列
      }
    }
  }
  void solve()
  {
    int s,t; //起点和终点
    int m;  //边数
    ll k;    //第 k 短
    cin>>n>>m>>k;

    s=n;t=1;   //处理终点和起点

    for(int i=1;i<=m;i++)
    {
      ll c;
      int u,v;
      cin>>u>>v>>c;
      e[u].push_back((node){v,c,1});//建立正边
      e[v].push_back((node){u,c,0});//建立反边
    }

    if(s==t)k++;  //起点与终点重合

    dj(t);//对 t 求 dj，走反边

    astar(s,t,k);  //求 s 到 t 的最短的 k 条路

    for(int i=1;i<=k;i++)
    {
      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

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int inf =1e9+7;
////////////////////
struct p3008  //链式前向星
{
    int to;
    int nex;
    int v;
}a[300000];
int head[25006]={0},cnt=0;

int dis[25006]={0};  //距离

void add(int u,int v,int w)
{
    a[++cnt].nex=head[u];
    a[cnt].to=v;
    a[cnt].v=w;
    head[u]=cnt;
}
////////////////////////////////
//SPFA，但是双端队列优化
//小的放队头，大的放队尾
void spfa(int t,int s)
{
    for(int i=1;i<=t;i++)dis[i]=inf;  //初始化距离
    deque<int> q;
    vector<int> vis(t+1),cou(t+1);

    q.push_front(s);  //起点入队

    dis[s]=0;  //起点初始化
    vis[s]++;

    while(!q.empty())
    {
        int u=q.front();  //取出队头
        q.pop_front();

        cou[u]++;

        if(cou[u]>t)return;  //遍历次数过大，出现负环

        for(int i=head[u];i;i=a[i].nex)  //遍历儿子
        {
            int v=a[i].to;
            int w=a[i].v;
            if(dis[v]>dis[u]+w)  //可松弛
            {
                dis[v]=dis[u]+w;  //松弛操作

                if(!vis[v])
                {
                    if(q.empty())q.push_back(v);       //特判！队空则随便入队
```

```
                    else if(dis[q.front()]>=dis[v])q.push_front(v);  //小的放队
头
                    else q.push_back(v);  //大的放队尾
                    vis[v]++;
                }
            }
        }
        vis[u]--;  //取消入队标记
    }
}
void solve()
{
    int t,r,p,s;
    cin>>t>>r>>p>>s;
    for(int i=1;i<=r;i++)
    {
        int u,v;
        int w;
        cin>>u>>v>>w;
        add(u,v,w);
        add(v,u,w);
    }
    for(int i=1;i<=p;i++)
    {
        int u,v;
        int w;
        cin>>u>>v>>w;
        add(u,v,w);
    }

    spfa(t,s);  //建好边调用即可，t 是点数，s 是起点

    for(int i=1;i<=t;i++)
    {
        if(dis[i]>=inf)cout<<"NO PATH\n";
        else cout<<dis[i]<<"\n";
    }
}
```

4.6.4 SPFA.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
////////////////////////////////////
struct p4779SPFA  //链式前向星
{
    int to;
    int nex;
    ll w;
}star[550000];
int head[100005]={0},cnt=0;
void add(int u,int v,ll w)
{
    star[++cnt].to=v;
    star[cnt].nex=head[u];
    star[cnt].w=w;
    head[u]=cnt;
}
////////////////////////////////

//SPFA,一款暴力的最短路算法
//常用于网络流、判负环、带负边的最短路
//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
int n,m,s;
ll dis[100005]={0};   //距离
int times[100005]={0}; //遍历次数
bool vis[100005];  //入队情况
void SPFA()
{
    memset(vis, 0 ,sizeof(vis)); //清空 vis
    for(int i=1;i<=n;i++)dis[i]=2147483647;  //距离初始化为无穷大
    dis[s]=0;  //起点归零
    queue<int> q;  //队列
    q.push(s);
    while(!q.empty())
    {
        int temp=q.front();
        q.pop();
        times[temp]++;
        vis[temp]=false;  //出队判定

        if(times[temp]>2+n)  //判负环  ，遍历次数过多说明出现了负环
        {
            times[s]=n+10;  //做标记
            break;
        }

        for(int i=head[temp];i;i=star[i].nex)  //遍历儿子
        {
            int v=star[i].to;
            if(dis[v]>dis[temp]+star[i].w)  //可松弛
            {
                dis[v]=dis[temp]+star[i].w;//松弛
                if(!vis[v])  //未入队则入队
                {
```

```
                    q.push(v);
                    vis[v]=true;
                }
            }
        }
    }

}
void solve()
{
    //spfa 用于求最短路
    cin>>n>>m>>s;
    for(int i=1;i<=m;i++)
    {
        int u,v;
        ll w;
        cin>>u>>v>>w;
        add(u,v,w);
    }

    SPFA();//建边后调用即可

    if(times[s]>n)  //找负环
    {
        cout<<"minus circle!\n";
        return;
    }
    for(int i=1;i<=n;i++)cout<<dis[i]<<" ";
    cout<<"\n";
}
```

4.6.5 判断欧拉回路或通路.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll N=250000+7;
int main()
{
  // ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  void solve();
  int t=1;
  while(t--)solve();
  return 0;
}
//无向图
//判断欧拉回路和通路
map<string,int> point;
int cnt=0,tot=0;
struct node{int u,v,i;};
vector<node> edge;
vector<int> e[N<<1];
int ind[N<<1],vis[N];
void yes(){cout<<"Possible\n";}
void no(){cout<<"Impossible\n";}
int dfs(int u)  //深搜找欧拉回路/通路
{
    int ans=0;
    for(auto re:e[u])
    {
        int v=edge[re].u^edge[re].v^u;
        if(!vis[re])continue;
        vis[re]--;
        ans++;
        ans+=dfs(v);
    }
    return ans;   //这里在数经过的边数
    //如果要找路，可以在这将点加入栈
}
bool check(int m)
{
    for(int i=1;i<=m;i++)  //建图、数度数
    {
        ind[edge[i].u]++;    //有向图则要分出度和入度
        ind[edge[i].v]++;

        vis[i]++;  //每条边只走一次

        e[edge[i].u].push_back(i);  //邻接表存边号
        e[edge[i].v].push_back(i);
    }

    int st=1;   //起点

    int flag=3; //

    for(int i=1;i<=tot;i++)
    {
        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

$2 \le n \le 2e5$ $1 \le m, q \le 2e5$ $1 \le wi,xi \le 1e9$

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 到 u 的距离+边长+v 到 n 的距离,1 到 v 的距离+边长+u 到 n 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(原最短路长度，min(1 到 u 的距离+新边长+v 到 n 的距离,1 到 v 的距离+新边长+u 到 n 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 E1,E2,E3……Ek. 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 Lu,1 到 u 的最短路经过 E1,E2,……E(Lu)。

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 Ru,u 到 n 的最短路经过 E1,E2,……E(Ru)。

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 的长度-原边长+新边长，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 Lu+1,Rv 的区间用 len 更新,比个 min

同样地，可以把 Lv+1,Ru 的区间用 len 更新

由于代码中点 1 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(logn)的时间内回答每个问题

总的复杂度为 O((m+n+q)logn)

4.6.6.1 solution:

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll inf=1e18;
#define N 200505
#define mod 1000000007
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int t=1;
    void solve();
    // cin>>t;
    while(t--)solve();
}
//换边最短路
//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 的权值修改为 ch 后的最短路，
//       询问对原图无影响

//       可用于删边、换边的最短路

//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
//此模板建议原封不动地抄
int n,m,qu;   //n 点数，m 边数，qu 询问次数
struct Original_edge{int u,v;ll len;}OE[N];  //存原边
struct tabl{int to,id;ll len;};   //用于邻接表的结构体、存边
struct node{int id; ll len; bool operator < (const node &x) const {return x.len<len;}};//用于优先队列的结构体，存点和距离
vector<tabl> edge[N<<1];   //邻接表
priority_queue<node> q;    //用于 dj 的优先队列
int lstvis[N]={0};
int l[N]={0};
int r[N]={0};    //
int ind[N]={0};   //标记，ind[i]==j，表示边 i 在原始最短路上，且是路上的第 j 条边
int vis[N];       //用于 dj 的 vis 数组

ll t[N<<4]={0};       //这个是线段树

```

```cpp
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
bool on_path[N];      //记录原始最短路，on_path[i]==true,表示点 i 在最短路上
int path_cnt=0;      //表示原始最短路上的边数

void dj(int p,ll dis[],int f)//起点，  对应数组，  操作编号
{
    for(int i=1;i<=n;i++)//初始化
    {
        vis[i]=0;
        dis[i]=inf;
    }
    dis[p]=0;
    q.push((node){p,0});  //加入起点
    while(!q.empty())
    {
        node temp=q.top();
        q.pop();
        int u=temp.id;
        ll w=temp.len;
        if(vis[u])continue;//跳过处理过的点
        vis[u]++;
        dis[u]=w;//记录距离
        for(auto re:edge[u])
        {
            int v=re.to;
            int id=re.id;
            ll tw=re.len;

            if(dis[v]<=w+tw)continue;
            dis[v]=w+tw;  //松弛
            lstvis[v]=id;  //记前驱边，表示点 v 从边 id 到达
            q.push((node){v,w+tw});

            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
            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
        }
    }
}
void trace()
{
    int u=1;    //u 初始为起点
    on_path[u]=true;  //做好起点的初始化
    l[u]=r[u]=0;

    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
    {
        int e_id=lstvis[u];  //取前驱边
        ind[e_id]=i;      //给前驱标记

        u^=OE[e_id].u^OE[e_id].v;  //取前驱点
        on_path[u]=true;        //前驱点做标记
        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
一
160         update(1,path_cnt,l[v]+1,r[u],1,disT[v]+w+disN[u]);
161     }
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
过 ti 边的最短路
179             }
180         }
181         else   //若换了最短路外的边
182         {
183             ans=disT[n];   //最理想是原最短路
184                            //换的边比原来的边大，则无需考虑
185
186             if(OE[ti].len>ch)  //若换的边更小，则需要比较
187             {
188                 int u=OE[ti].u,v=OE[ti].v;
189
190                 ans=min(ans,min(disT[u]+ch+disN[v],disT[v]+ch+disN[u]));
                    //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
191 u，再经过边 ti，再从 v 沿着最优路径到达终点
                    //         也可能是    disT[v]+ch+disN[u]  表示从起点沿着最优路径到
192 达 v，再经过边 ti，再从 u 沿着最优路径到达终点
193             }
194         }
195         cout<<ans<<"\n";
196     }
197 }
```

4.6.7 求欧拉回路或通路.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll N=1055;
int main()
{
  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  void solve();
  int t=1;
  while(t--)solve();
  return 0;
}
//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
int m;
struct node{int u,v,i;};
vector<node> edge;
vector<pair<int,int>> e[N];
int vis[N]={0};
int ind[N]={0};
stack<int> stk;
bool cmp(pair<int,int> x,pair<int,int> y)
{
    int ix=x.second,iy=y.second;
    int u=x.first,v=y.first;
    return (edge[ix].v^edge[ix].u^u)<(edge[iy].u^edge[iy].v^v);
}
void dfs(int u)
{
    for(auto re:e[u])
    {
        int v=edge[re.second].v^edge[re.second].u^u;
        if(!vis[re.second])continue;
        vis[re.second]--;
        dfs(v);
    }
    stk.push(u);
}
void solve()
{
    memset(vis,0,sizeof(vis));
    int st=550;
    cin>>m;
    edge.clear();
    edge.push_back({-1,-1,0});
    for(int i=1;i<=500;i++)
    {
        ind[i]=0;
        e[i].clear();
    }
    for(int i=1;i<=m;i++)
    {
        int u,v;
        cin>>u>>v;
        edge.push_back({u,v,i});
        e[u].push_back({u,i});  //邻接表，但是存编号
        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

```cpp
/*
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
*/
void solve(){
  int n,m;
  cin>>n>>m;
  vector<int> e[n*2+1];
  priority_queue<array<int,3>> q;
  for(int i=1;i<=m;i++){
    int u,v,w;
    cin>>u>>v>>w;
    q.push({w,u,v});
  }
  vector<int> fa(n*2+10),w(n*2+10);
  iota(fa.begin(),fa.end(),0);
  function<int(int)> find=[&](int x){
    return fa[x]==x ? x : fa[x]=find(fa[x]);
  };
  auto merge=[&](int x,int y){
    int fx=find(x),fy=find(y);
    fa[fx]=fy;
  };
  int cnt;
  auto Kruskal=[&]()->void {
    cnt=n;
    while(!q.empty()){
      auto a=q.top();q.pop();
      int fx=find(a[1]),fy=find(a[2]);
      if(fx==fy) continue;
      cnt++;
      merge(fx,cnt);
      merge(fy,cnt);
      w[cnt]=a[0];
      e[cnt].push_back(fy);
      e[cnt].push_back(fx);
    }
  };
  Kruskal();
  vector<vector<int>> faa(n*2+10,vector<int>(19));
  vector<int> dep(n*2+10);
  vector<bool> vis(n*2+10);
  function<void(int,int)> dfs=[&](int id,int u){
    faa[id][0]=u;dep[id]=dep[u]+1;vis[id]=1;
    for(int i=1;i<=19;i++){
      faa[id][i]=faa[faa[id][i-1]][i-1];
    }
    for(auto x:e[id]) dfs(x,id);
  };
  for(int i=cnt;i>=1;i--){
    if(!vis[i]) dfs(i,i);
  }
  auto lca=[&](int x,int y)->int{
    if(find(x)!=find(y)) return 0;
```

```
    if(dep[x]<dep[y]) swap(x,y);
    int tmp=dep[x]-dep[y];
    for(int i=0;i<19;i++){
      if((tmp>>i&1)) x=faa[x][i];
    }
    if(x==y) return x;
    for(int i=18;i>=0;i--){
      if(faa[x][i]!=faa[y][i]){
        x=faa[x][i];
        y=faa[y][i];
      }
    }
    return faa[x][0];
  };
  w[0]=-1;
  int qq;cin>>qq;
  while(qq--){
    int x,y;
    cin>>x>>y;
    int l=lca(x,y);
    cout<<w[l]<<"\n";
  }
}
```

4.7.2 lca.h

```cpp
class LCA{
public:
    vector<vector<pii>> cnj;
    vector<int> lg,dep;
    vector<array<int,32>> fa,wei;
    int n;

    LCA(int _n) {
        n = _n;
        cnj.resize(n+1);
        lg.resize(n+1),fa.resize(n+1),dep.resize(n+1),wei.resize(n+1);
        for(int i = 1; i <= n; i ++)
            lg[i] = lg[i-1] + (1 << lg[i-1] == i);
    }

    void addEdge(int u,int v,int w) {
        cnj[u].push_back({v,w});
        cnj[v].push_back({u,w});
    }

    void build(int rt = 1) {
        using itf = function<void(int,int)>;
        itf dfs = [&](int p,int f) -> void {
            fa[p][0] = f,dep[p] = dep[f] + 1;
            // wei[p][0] = 0;
            for(int i = 1;i <= lg[dep[p]];i ++) fa[p][i] = fa[fa[p][i-1]][i-1];
            for(int i = 1;i <= lg[dep[p]];i ++) wei[p][i] = max(wei[p][i-1],wei[fa[p][i-1]][i-1]);
            for(auto [x,w]:cnj[p]) if(x == f) continue;
            else wei[x][0] = w,dfs(x,p);
        };
        dfs(rt,0);
    }

    int get(int x,int y) {
        if(dep[x] < dep[y]) swap(x,y);
        while(dep[x] > dep[y]) x = fa[x][lg[dep[x] - dep[y]] - 1];
        if(x == y) return x;
        for(int k = lg[dep[x]]-1;k >= 0;k --) if(fa[x][k] != fa[y][k]) x = fa[x][k],y = fa[y][k];
        return fa[x][0];
    }

    int getmaxw(int x,int y) {
        int curmx = 0;
        if(dep[x] < dep[y]) swap(x,y);
        while(dep[x] > dep[y]) curmx = max(curmx,wei[x][lg[dep[x] - dep[y]] - 1]), x = fa[x][lg[dep[x] - dep[y]] - 1];
        if(x == y) return curmx;
        for(int k = lg[dep[x]]-1;k >= 0;k --)
            if(fa[x][k] != fa[y][k])
                curmx = max(curmx,wei[x][k]),x = fa[x][k],
                curmx = max(curmx,wei[y][k]),y = fa[y][k];
        return max({curmx,wei[x][0],wei[y][0]});
    }
```

```cpp
};

//----------VERSION 2----------


void solve(){
  int n,m,root;
  cin>>n>>m>>root;
  vector<int> e[n+1],dep(n+1);
  vector<vector<int>> fa(n+1,vector<int>(21));
  for(int i=1;i<n;i++){
    int u,v;
    cin>>u>>v;
    e[u].push_back(v);
    e[v].push_back(u);
  }
  function<void(int,int)> dfs=[&](int id,int u){
    fa[id][0]=u;
    dep[id]=dep[u]+1;
    for(int i=1;i<=20;i++) fa[id][i]=fa[fa[id][i-1]][i-1];
    for(auto x:e[id]){
      if(x==u) continue;
      dfs(x,id);
    }
  };
  dfs(root,root);
  function<int(int,int)> lca=[&](int x,int y){
    if(dep[x]<dep[y]) swap(x,y);
    int tmp=dep[x]-dep[y];
    for(int i=0;i<=20;i++){
      if(tmp>>i&1) x=fa[x][i];
    }
    if(x==y) return x;
    for(int i=20;i>=0;i--){
      if(fa[x][i]!=fa[y][i]){
        x=fa[x][i];
        y=fa[y][i];
      }
    }
    return fa[x][0];
  };
  while(m--){
    int a,b;
    cin>>a>>b;
    cout<<lca(a,b)<<"\n";
  }
  return;
}
```

4.7.3 中心.h

```cpp
// 这份代码默认节点编号从 1 开始，即 i ∈ [1,n]，使用 vector 存图
int d1[N], d2[N], up[N], x, y, mini = 1e9;  // d1,d2 对应上文中的 len1,len2

struct node {
  int to, val;  // to 为边指向的节点，val 为边权
};

vector<node> nbr[N];

void dfsd(int cur, int fa) {  // 求取 len1 和 len2
  for (node nxtn : nbr[cur]) {
    int nxt = nxtn.to, w = nxtn.val;  // nxt 为这条边通向的节点，val 为边权
    if (nxt == fa) {
      continue;
    }
    dfsd(nxt, cur);
    if (d1[nxt] + w > d1[cur]) {  // 可以更新最长链
      d2[cur] = d1[cur];
      d1[cur] = d1[nxt] + w;
    } else if (d1[nxt] + w > d2[cur]) {  // 不能更新最长链，但可更新次长链
      d2[cur] = d1[nxt] + w;
    }
  }
}

void dfsu(int cur, int fa) {
  for (node nxtn : nbr[cur]) {
    int nxt = nxtn.to, w = nxtn.val;
    if (nxt == fa) {
      continue;
    }
    up[nxt] = up[cur] + w;
    if (d1[nxt] + w != d1[cur]) {  // 如果自己子树里的最长链不在 nxt 子树里
      up[nxt] = max(up[nxt], d1[cur] + w);
    } else {  // 自己子树里的最长链在 nxt 子树里，只能使用次长链
      up[nxt] = max(up[nxt], d2[cur] + w);
    }
    dfsu(nxt, cur);
  }
}

void GetTreeCenter() {  // 统计树的中心，记为 x 和 y（若存在）
  dfsd(1, 0);
  dfsu(1, 0);
  for (int i = 1; i <= n; i++) {
    if (max(d1[i], up[i]) < mini) {  // 找到了当前 max(len1[x],up[x])最小点
      mini = max(d1[i], up[i]);
      x = i;
      y = 0;
    } else if (max(d1[i], up[i]) == mini) {  // 另一个中心
      y = i;
    }
  }
}
```

4.7.4 支配树.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
#define emp empty
#define fi first
#define se second
const int N=3e5+7;
const ll inf=1e16+7;
const ll mod=998244353;
signed main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    void solve();
  int t=1;
//    cin>>t;
  while(t--)solve();
    return 0;
}
struct node
{
  int v;
  int nex;
}a[N<<2];
int n,m,u,v,cnt=0,dfc=0;
int siz[N],dfn[N],pos[N],sdm[N],idm[N],fa[N],fth[N],mn[N];
//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
int h[3][N<<1];//三幅图的表头，分别是原图、反图、支配树图
void clear()
{
  memset(h,0,sizeof(h));
  memset(dfn,0,sizeof(dfn));
  memset(siz,0,sizeof(siz));
  cnt=0;
  dfc=0;
}
void add(int u,int v,int x)
{
  a[++cnt]=(node){v,h[x][u]};
  h[x][u]=cnt;
}
void dfs(int u)
{
  dfn[u]=++dfc;
  pos[dfc]=u;
  for(int i=h[0][u];i;i=a[i].nex)
  {
    int v=a[i].v;
    if(dfn[v])continue;
    dfs(v);
    fth[v]=u;
  }
}
int find(int u)//这是一个带权并查集?
{
```

```
  if(fa[u]==u)return u;
  int temp=fa[u];
  fa[u]=find(fa[u]);
  if(dfn[sdm[mn[temp]]]<dfn[sdm[mn[u]]])
  {
    mn[u]=mn[temp];
  }
  return fa[u];
}
void tar(int st)
{
  dfs(st);
  for(int i=1;i<=n;i++)
  {
    sdm[i]=fa[i]=mn[i]=i;
  }
  for(int i=dfc;i>=2;i--)
  {
    int u=pos[i];
    int res=mod;
    for(int j=h[1][u];j;j=a[j].nex)
    {
      int v=a[j].v;
      if(!dfn[v])continue;
      find(v);
      if(dfn[v]<dfn[u])res=min(res,dfn[v]);
      else res=min(res,dfn[sdm[mn[v]]]);
    }
    sdm[u]=pos[res];
    fa[u]=fth[u];
    add(sdm[u],u,2);
    u=fth[u];
    for(int j=h[2][u];j;j=a[j].nex)
    {
      int v=a[j].v;
      find(v);
      if(u==sdm[mn[v]])
      {
        idm[v]=u;
      }
      else
      {
        idm[v]=mn[v];
      }
    }
    h[2][u]=0;
  }
  for(int i=2;i<=dfc;i++)
  {
    int u=pos[i];
    if(idm[u]!=sdm[u])idm[u]=idm[idm[u]];
  }
  for(int i=dfc;i>=2;i--)
  {
    int u=pos[i];
    ++siz[u];
    siz[idm[u]]+=siz[u];
  }
```

```
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

```cpp
/*
n(<=100)个点 m(<=500)条带权无向边 G
给定 k（<=10）个节点的点集 S，选出 G 的子图 G1
使得 S 属于 G1，G1 是联通图  边权和最小

*/

#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=4e3;
const int inf=2e9;
int n,m,k,p[N],state;
int dp[N][N];
int head[N],to[N],ne[N],w[N],tot;
void add(int x,int y,int z){
  ne[++tot]=head[x];
  to[tot]=y,w[tot]=z;
  head[x]=tot;
}
queue<int>q;
bool vis[N];
void spfa(int s){
  while(!q.empty()){
    int u=q.front();q.pop();
    vis[u]=0;
    for(int i=head[u];i;i=ne[i]){
      int v=to[i];
      if(dp[v][s]>dp[u][s]+w[i]){
        dp[v][s]=dp[u][s]+w[i];
        if(!vis[v]) q.push(v),vis[v]=1;
      }
    }
  }
}
signed main(){
  cin>>n>>m>>k;
  for(int i=1;i<=m;i++){
    int x,y,z;cin>>x>>y>>z;
    add(x,y,z),add(y,x,z);
  }
  state=(1<<k)-1;
  for(int i=1;i<=n;i++) for(int s=0;s<=state;s++) dp[i][s]=inf;
  for(int i=1;i<=k;i++){
    cin>>p[i];
    dp[p[i]][1<<(i-1)]=0;
  }
  for(int s1=1;s1<=state;s1++){
    for(int i=1;i<=n;i++){
      for(int s2=s1&(s1-1);s2;s2=s1&(s2-1)) dp[i][s1]=min(dp[i][s1],dp[i]
[s2]+dp[i][s1^s2]);//枚举子集
      if(dp[i][s1]<inf) q.push(i),vis[i]=1;//将这个点看成出发点
    }
    spfa(s1);
  }
  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

```cpp
vector<int> dep(n+1);
int mx=0,tmp=-1;
function<void(int,int)> dfs=[&](int id,int fa){
  dep[id]=dep[fa]+1;
  for(auto x:e[id]){
    if(x==fa) continue;
    dfs(x,id);
  }
  if(dep[id]>mx){
    mx=dep[id];
    tmp=id;
  }
};
dfs(1,0);
int d1=tmp;
mx=0;
dfs(d1,0);
int d2=tmp; //d1,d2
```

4.7.7 虚树.h

```cpp
int dfn[N];
int h[N],m,a[N],len;
bool cmp(int x,int y){
  return dfn[x]<dfn[y];
}
void build(){
  sort(h+1,h+1+m,cmp);  //dfn 排序
  for(int i=1;i<m;i++){
    a[++len]=h[i];
    a[++len]=lca(h[i],h[i+1]);  //插入 LCA
  }
  a[++len]=h[m];
  sort(a+1,a+1+len,cmp);  //DFN 排序
  len=unique(a+1,a+len+1)-a+1;  //去重
  for(int i=1,lc;i<len;i++){
    lc=lca(a[i],a[i+1]);
    conn(lc,a[i+1]);  //连边
  }
}
```

4.7.8 重心.h

```cpp
int sz[N];  //鍵値対C
int weight[N]; //鍵値対の重さC
int centroid[2]; //重心の鍵値対
void dfs(int id,int fa){
  sz[id]=1;
  weight[id]=0;
  for(auto x:e[id]){
    if(x==fa) continue;
    dfs(x,id);
    sz[id]+=sz[x];
    weight[id]=max(weight[id],sz[x]);
  }
  weight[id]=max(weight[id],n-sz[id]);
  if(weight[id]<=n/2){
    centroid[centroid[0]!=0]=id;
  }
}
```

4.7.9 重链剖分.h

```cpp
/*
n 个节点树，节点有权值，q 次询问，四种操作
op1：x 到 y 最短路径上所有节点的值加上 val
op2：输出 x 到 y 最短路径上所有节点的权值和
op3：将以 x 为根节点的子树内所有节点值都加上 val
op4：输出以 x 为根节点的子树内所有节点的权值和
op5：输入 x,y 输出 lca(x,y)
*/
const int N=5e5+10;
vector<int> e[N];
//如果开 vector 有爆空间的可能
int fa[N],son[N],sz[N],dep[N],dfn[N],rkn[N],top[N];
int fi[N],la[N];
int idx=0;
void dfs1(int id,int u){
  fa[id]=u;
  sz[id]=1;
  dep[id]=dep[u]+1;
  for(auto x:e[id]){
    if(x==u) continue;
    dfs1(x,id);
    sz[id]+=sz[x];
    if(sz[x]>sz[son[id]]) son[id]=x;
  }
}
void dfs2(int id,int tp){
  top[id]=tp;
  dfn[id]=++idx;
  fi[id]=la[id]=idx;
  rkn[idx]=id;
  if(son[id]) dfs2(son[id],tp);
  for(auto x:e[id]){
    if(x==fa[id]||x==son[id]) continue;
    dfs2(x,x);
  }
  for(auto x:e[id]){
    if(x==fa[id]) continue;
    la[id]=max(la[id],la[x]);
  }
}
vector<int> a(N);
#define lc p<<1
#define rc p<<1|1
struct node{
  int l,r,sum,mx;
  int lz;
}tr[4*N];
void pushup(int p){
  tr[p].sum=tr[lc].sum+tr[rc].sum;
  tr[p].mx=max(tr[lc].mx,tr[rc].mx);
}
void build(int p,int l,int r){  // (1,1,n)
  tr[p].l=l;tr[p].r=r;tr[p].lz=0;
  if(l==r){
    tr[p].sum=tr[p].mx=a[rkn[l]];
```

```
    return;
  }
  int m=l+r>>1;
  build(lc,l,m);
  build(rc,m+1,r);
  pushup(p);
}
void pushdown(int p){
  if(tr[p].lz){
    tr[lc].mx+=tr[p].lz;
    tr[rc].mx+=tr[p].lz;
    tr[lc].sum+=(tr[lc].r-tr[lc].l+1)*tr[p].lz;
    tr[rc].sum+=(tr[rc].r-tr[rc].l+1)*tr[p].lz;
    tr[lc].lz+=tr[p].lz;
    tr[rc].lz+=tr[p].lz;
    tr[p].lz=0;
  }
}
void update1(int p,int x,int val){  //单点修改  传(1, dfn[id],val)
  if(tr[p].l==tr[p].r){
    // tr[p].mx=tr[p].sum=val;
    tr[p].mx+=val;
    tr[p].sum+=(tr[p].r-tr[p].l+1)*val;
    return;
  }
  pushdown(p);
  int m=tr[p].l+tr[p].r>>1;
  if(x<=m) update1(lc,x,val);
  else update1(rc,x,val);
  pushup(p);
}
void update2(int p,int l,int r,int val){  //区间修改  传(1,dfn[x],dfn[y],val)
  if(l<=tr[p].l&&tr[p].r<=r){
    tr[p].sum+=(tr[p].r-tr[p].l+1)*val;
    tr[p].mx+=val;
    tr[p].lz+=val;
    return;
  }
  pushdown(p);
  int m=tr[p].l+tr[p].r>>1;
  if(l<=m) update2(lc,l,r,val);
  if(m<r) update2(rc,l,r,val);
  pushup(p);
}
int query1(int p,int l,int r){  // 查询 sum  传(1,dfn[x],dfn[y])
  if(l<=tr[p].l&&tr[p].r<=r) return tr[p].sum;
  pushdown(p);
  int sum=0;
  int m=tr[p].l+tr[p].r>>1;
  if(l<=m) sum+=query1(lc,l,r);
  if(m<r) sum+=query1(rc,l,r);
  return sum;
}
int query2(int p,int l,int r){  //查询 mx 传(1,dfn[x],dfn[y])
  if(l<=tr[p].l&&tr[p].r<=r) return tr[p].mx;
  pushdown(p);
  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

```cpp
#include <algorithm>
#include <iostream>
using ull = unsigned long long;

ull p[64];

void insert(ull x) {
  for (int i = 63; ~i; --i) {
    if (!(x >> i))  // x 的第 i 位是 0
      continue;
    if (!p[i]) {
      p[i] = x;
      break;
    }
    x ^= p[i];
  }
}

using std::cin;
using std::cout;

int main() {
  int n;
  cin >> n;
  ull a;
  for (int i = 1; i <= n; ++i) {
    cin >> a;
    insert(a);
  }
  ull ans = 0;
  for (int i = 63; ~i; --i) {
    ans = std::max(ans, ans ^ p[i]);
  }
  cout << ans << '\n';
  return 0;
}
/// ================VERSION 2==========
#include <iostream>
using ull = unsigned long long;
constexpr int MAXN = 1e5 + 5;

ull deg(ull num, int deg) { return num & (1ull << deg); }

ull a[MAXN];
using std::cin;
using std::cout;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n;
  cin >> n;
  for (int i = 1; i <= n; ++i) cin >> a[i];
```

```
  int row = 1;
  for (int col = 63; ~col && row <= n; --col) {
    for (int i = row; i <= n; ++i) {
      if (deg(a[i], col)) {
        std::swap(a[row], a[i]);
        break;
      }
    }
    if (!deg(a[row], col)) continue;
    for (int i = 1; i <= n; ++i) {
      if (i == row) continue;
      if (deg(a[i], col)) {
        a[i] ^= a[row];
      }
    }
    ++row;
  }
  ull ans = 0;
  for (int i = 1; i < row; ++i) {
    ans ^= a[i];
  }
  cout << ans << '\n';
  return 0;
}
```

5.1.2 `det.h`

```cpp
constexpr double EPS = 1E-9;
int n;
vector<vector<double>> a(n, vector<double>(n));

double det = 1;
for (int i = 0; i < n; ++i) {
  int k = i;
  for (int j = i + 1; j < n; ++j)
    if (abs(a[j][i]) > abs(a[k][i])) k = j;
  if (abs(a[k][i]) < EPS) {
    det = 0;
    break;
  }
  swap(a[i], a[k]);
  if (i != k) det = -det;
  det *= a[i][i];
  for (int j = i + 1; j < n; ++j) a[i][j] /= a[i][i];
  for (int j = 0; j < n; ++j)
    if (j != i && abs(a[j][i]) > EPS)
      for (int k = i + 1; k < n; ++k) a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

5.1.3 GaussianELI.h

```cpp
using ld = long double;
// A:[NxN] b[1xN]
//返回: {1,[]}代表有唯一解, {-1,[]}代表无解或无穷解
pair<int,vector<ld>> GaussianELI(vector<vector<ld>> a,vector<ld> b) {
    assert(a.size()),assert(a.size() == a[0].size()),assert(a.size() == b.size());
    int n = a.size();
    vector<int> p(n);
    iota(all(p),0);
    for(int i = 0;i < n;i ++){
        sort(p.begin()+i,p.end(),[&](int x,int y) -> bool {return abs(a[x][i]) > abs(a[y][i]); });
        if(a[p[i]][i] == 0) continue;
        for(int j = i+1;j < n;j ++)
        {
            ld k = a[p[j]][i]/a[p[i]][i] ;
            for(int f = i;f < n;f ++) a[p[j]][f] -= a[p[i]][f] * k;
            b[p[j]] -= b[p[i]] * k;
        }
    }
    vector<ld> res(n);
    for(int i = n-1;i >= 0;i --)
    {
        ld dev = 0;
        for(int f = i+1;f < n;f ++) dev += a[p[i]][f] * res[f];
        if(fabs(a[p[i]][i]) <= 1e-8){ return {-1,{}};} else res[i] = (b[p[i]] - dev)/a[p[i]][i];
    }
    return {f,res};
}
```

5.1.4 单纯形法.cpp

```cpp
#include <cmath>
#include <cstring>
#include <iostream>
using namespace std;
constexpr int M = 10005, N = 1005, INF = 1e9;

int n, m;
double a[M][N], b[M], c[N], v;

void pivot(int l, int e) {  // 转轴操作函数
  b[l] /= a[l][e];
  for (int j = 1; j <= n; j++)
    if (j != e) a[l][j] /= a[l][e];
  a[l][e] = 1 / a[l][e];

  for (int i = 1; i <= m; i++)
    if (i != l && fabs(a[i][e]) > 0) {
      b[i] -= a[i][e] * b[l];
      for (int j = 1; j <= n; j++)
        if (j != e) a[i][j] -= a[i][e] * a[l][j];
      a[i][e] = -a[i][e] * a[l][e];
    }

  v += c[e] * b[l];
  for (int j = 1; j <= n; j++)
    if (j != e) c[j] -= c[e] * a[l][j];
  c[e] = -c[e] * a[l][e];

  // swap(B[l],N[e])
}

double simplex() {
  while (true) {
    int e = 0, l = 0;
    for (e = 1; e <= n; e++)
      if (c[e] > (double)0) break;
    if (e == n + 1) return v;  // 此时 v 即为最优解
    double mn = INF;
    for (int i = 1; i <= m; i++) {
      if (a[i][e] > (double)0 && mn > b[i] / a[i][e]) {
        mn = b[i] / a[i][e];  // 找对这个 e 限制最紧的 l
        l = i;
      }
    }
    if (mn == INF) return INF;  // unbounded
    pivot(l, e);                // 转动 l,e
  }
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> m;
  for (int i = 1; i <= n; i++) cin >> c[i];
  for (int i = 1; i <= m; i++) {
    int s, t;
    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

```
__builtin_ffsll(x)
// 返回 x 的二进制末尾最后一个 1 的位置

__builtin_clzll(x)
// 返回 x 的二进制的前导 0 的个数。

__builtin_ctzll(x)
// 返回 x 的二进制末尾连续 0 的个数。

__builtin_clrsbll(x)
// 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 1 的个数减一。

__builtin_popcountll(x)
// 返回 x 的二进制中 1 的个数。

__builtin_parity(x)
// 判断 x 的二进制中 1 的个数的奇偶性。

int binpow(int x, int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y & 1)
            res = res * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return res;
}

void exgcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1, y = 0;
    return;
  }
  exgcd(b, a % b, y, x);
  y -= a / b * x;
}

binpow(x,mod-2)
```

5.2.2 CRT.h

```cpp
int CRT(vector<int> &r, vector<int> &a)
{ // % r === a
    int n = a.size();
    __int128 k = 1, ans = 0;
    for (int i = 0; i < n; i++) k *= r[i];
    for (int i = 0; i < n; i++)
    {
        __int128 m = k / r[i];
        int b, y;
        exgcd(m, r[i], b, y); // b * m mod r[i] = 1
        ans = (ans + a[i] * m * b % k) % k;
    }
    return (ans % k + k) % k;
}



int mul(int a, int b, int m) {
    return (__int128)a * b % m;
}

int exgcd (int a,int b,int &x,int &y) {
    if (b == 0) { x = 1, y = 0; return a; }
    int g = exgcd(b, a % b, x, y), tp = x;
    x = y, y = tp - a / b * y;
    return g;
};

int EXCRT(vector<int> &a,vector<int> &r) { // % r == a
    int x, y, k;
    int n = r.size();
    int M = a[0], ans = r[0];
    for (int i = 1; i < n; ++ i) {
        int ca = M, cb = a[i], cc = (r[i] - ans % cb + cb) % cb;
        int gcd = exgcd(ca, cb, x, y), bg = cb / gcd;
        if (cc % gcd != 0) return -1;
        x = mul(x, cc / gcd, bg);
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}
```

5.2.3 Eular_phi.h

```cpp
int euler_phi(int n) {
  int ans = n;
  for (int i = 2; i * i <= n; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
```

5.2.4 Eular_sieve.h

```cpp
vector<int> init(int n)
{
    vector<int> pri;
    vector<bool> vis(n, 0);
    for (int i = 2; i <= n; i++)
    {
        if (!vis[i])
            pri.push_back(i);
        for (int j = 0; j < pri.size(); j++)
        {
            if (i * pri[j] > n)
                break;
            vis[pri[j] * i] = 1;
            if (i % pri[j] == 0)
                break;
        }
    }
    return pri;
}
```

5.2.5 factor_pr.h

```cpp
#define int long long
#define pii pair<int, int>
const int INF = 1145141919810LL;
using namespace std;

class Pollard_Rho
{
private:

    vector<int> B;

    int mul(int a, int b, int m)
    {
        int r = a * b - m * (int)(1.L / m * a * b);
        return r - m * (r >= m) + m * (r < 0);
    }

    int mypow(int a, int b, int m)
    {
        int res = 1 % m;
        for (; b; b >>= 1, a = mul(a, a, m))
        {
            if (b & 1)
            {
                res = mul(res, a, m);
            }
        }
        return res;
    }

    bool MR(int n)
    {
        if (n <= 1)
            return 0;
        for (int p : B)
        {
            if (n == p)
                return 1;
            if (n % p == 0)
                return 0;
        }
        int m = (n - 1) >> __builtin_ctz(n - 1);
        for (int p : B)
        {
            int t = m, a = mypow(p, m, n);
            while (t != n - 1 && a != 1 && a != n - 1)
            {
                a = mul(a, a, n);
                t *= 2;
            }
            if (a != n - 1 && t % 2 == 0)
                return 0;
        }
        return 1;
    }

    inline const int getfecsum(int _n)
```

```
    {
        int sum = 0;
        while (_n)
        {
            sum += _n % 10;
            _n /= 10;
        }
        return sum;
    };

    int PR(int n)
    {
        for (int p : B)
        {
            if (n % p == 0)
                return p;
        }
        auto f = [&](int x) -> int
        {
            x = mul(x, x, n) + 1;
            return x >= n ? x - n : x;
        };
        int x = 0, y = 0, tot = 0, p = 1, q, g;
        for (int i = 0; (i & 255) || (g = gcd(p, n)) == 1; i++, x = f(x), y = f(f(y)))
        {
            if (x == y)
            {
                x = tot++;
                y = f(x);
            }
            q = mul(p, abs(x - y), n);
            if (q)
                p = q;
        }
        return g;
    }

    vector<int> fac(int n)
    {
        // if(n == 0)
        // #define pb emplace_back
        if (n <= 1)
            return {};
        if (MR(n))
            return {n};
        int d = PR(n);
        auto v1 = fac(d), v2 = fac(n / d);
        auto i1 = v1.begin(), i2 = v2.begin();
        vector<int> ans;
        while (i1 != v1.end() || i2 != v2.end())
        {
            if (i1 == v1.end())
            {
                ans.pb(*i2++);
            }
            else if (i2 == v2.end())
            {
```

```
                ans.pb(*i1++);
            }
            else
            {
                if (*i1 < *i2)
                {
                    ans.pb(*i1++);
                }
                else
                {
                    ans.pb(*i2++);
                }
            }
        }
        return ans;
    }

public:

    Pollard_Rho(){
        B = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    }

    vector<pii> fac_Comp(int n)
    {
        auto srt = fac(n);
        map<int, int> cnt;
        for (auto x : srt)
            cnt[x]++;
        vector<pii> rt;
        for (auto x : cnt)
            rt.push_back(x);
        return rt;
    }

    vector<int> fac_pri(int n)
    {
        return fac(n);
    }

    vector<int> fac_all(int n)
    {
        vector<pii> rt = fac_Comp(n);
        vector<int> v;
        function<void(int, int)> dfs = [&](int id, int x)
        {
            if (id == rt.size())
            {
                v.push_back(x);
                return;
            }
            for(int i = 0;i <= rt[id].se;i ++)
            {
                dfs(id + 1, x * (mypow(rt[id].fi, i, INF)));
            }
        };
        dfs(0, 1);
        return v;
```

```
173         }
174 };
```

5.2.6 factror_pri.h

```cpp
int n, m, p, b[10000005], prime[1000005], t, min_prime[10000005];
void euler_Prime(int n)
{ // 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
    for (int i = 2; i <= n; i++)
    {
        if (b[i] == 0)
        {
            prime[++t] = i;
            min_prime[i] = t;
        }
        for (int j = 1; j <= t && i * prime[j] <= n; j++)
        {
            b[prime[j] * i] = 1;
            min_prime[prime[j] * i] = j;
            if (i % prime[j] == 0)
                break;
        }
    }
}
long long c(int n, int m, int p)
{ // 计算 C(n,m)%p 的值
    euler_Prime(n);
    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
    for (int i = 1; i <= t; i++)
        a[i] = 0; // 注意清 0，一开始是随机数
    for (int i = n; i >= n - m + 1; i--)
    { // 处理分子
        int x = i;
        while (x != 1)
        {
            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）
            x /= prime[min_prime[x]];
        }
    }
    for (int i = 1; i <= m; i++)
    { // 处理分母
        int x = i;
        while (x != 1)
        {
            a[min_prime[x]]--;
            x /= prime[min_prime[x]];
        }
    }
    long long ans = 1;
    for (int i = 1; i <= t; i++)
    { // 枚举质数的编号，看它出现了几次
        while (a[i] > 0)
        {
            ans = ans * prime[i] % p;
            a[i]--;
        }
    }
    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 $[l, r] = [l, \left\lfloor \frac{n}{\lfloor \frac{n}{l} \rfloor} \right\rfloor]$ 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 $\quad r \leftarrow \left\lfloor \frac{n}{\lfloor n/l \rfloor} \right\rfloor$
7 $\quad result \leftarrow result + [s(r) - s(l-1)] \times \left\lfloor \frac{n}{l} \right\rfloor$
8 $\quad l \leftarrow r + 1$
9 **end while**

最终得到的 $result$ 即为所求的和式。

Figure 1: 整除分块.png

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define inf 100000000000000
const int N=2e6+7;
//题目，在 1<=a<b<=n 的条件下，求 gcd(a,b)的和

//这里用到了欧拉函数
//欧拉函数也可用欧拉筛求出
int main()
{
    void solve();
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    solve();
    return 0;
}
bool isprime[N];
```

```cpp
vector<ll> p;
ll phi[N]={0,1};//边界条件
void eular(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!isprime[i])
        {
            p.push_back(i);
            phi[i]=i-1;
        }
        for(auto re:p)
        {
            if(i*re>n)break;
            isprime[i*re]=1;
            if(i%re==0){phi[i*re]=phi[i]*re;break;}
            phi[i*re]=phi[i]*(re-1);
        }
    }
    //到此，欧拉函数就求出来了
    for(int i=1;i<=n;i++)phi[i]+=phi[i-1];  //此处在求函数的前缀和，用于整除分块
}

ll cal(ll n,ll m)
{
    ll l=1,r=0,ans=0;
    while(l<=n)
    {
        r=min((n/(n/l)),(m/(m/l)));
        ans+=(phi[r]-phi[l-1])*(n/l)*(m/l);
        l=r+1;
    }
    return ans;
}
void solve()
{
    ll n;
    cin>>n;
    eular(n);
    ll ans=(cal(n,n)-n*(n+1)/2)/2;
    cout<<ans<<"\n";
}
```

5.2.8 组合数.h

```cpp
const int N = 1e6;
const int mod = 1e9+7;

int binpow(int x, int y)
{
    int ans = 1;
    while (y)
    {
        if (y & 1) ans = ans * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return ans;
}

vector<int> fac(N), inv(N);

void init()
{
    fac[0] = inv[0] = 1;
    for (int i = 1; i < N; i++) fac[i] = fac[i - 1] * i % mod;
    inv[N - 1] = binpow(fac[N - 1], mod - 2);
    for (int i = N - 2; i >= 1; i--)
    {
        inv[i] = inv[i + 1] * (i + 1) % mod;
    }
}

auto C = [&](int x, int y) -> int
{
    return (fac[x] * inv[y] % mod) * inv[x - y] % mod;
};
```

5.2.9 莫反.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=5e4+7;
//题目：在 1<=a<=n,1<=b<=m 的条件下，求满足 gcd(a,b)==d 的(a,b)的个数

//莫比乌斯反演
//莫比乌斯函数可以用欧拉筛求出
bool isprime[N];
vector<ll> prime;
int mu[N];
void eular()
{
  mu[1]=1;//边界条件
  for(int i=2;i<N;i++)
  {
    if(!isprime[i])
    {
      prime.push_back(i);
      mu[i]=-1;
    }
    for(auto re:prime)
    {
      if(i*re>=N)break;
      isprime[i*re]=1;
      if(i%re==0)
      {
        mu[i*re]=0;
        break;
      }
      mu[i*re]=-mu[i];
    }
  }
  //到此，莫比乌斯函数已经求出

  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
}
int main()
{
  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  int t=1;
  cin>>t;
  eular();
  void solve();
  while(t--)solve();
  return 0;
}
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
{
  ll ans=0;
  n/=d;
  m/=d;
  for(int l=1,r=0;l<=min(n,m);)
  {
    r=min(min(n/(n/l),m/(m/l)),min(n,m));
    ans+=(ll)(n/l)*(m/l)*(mu[r]-mu[l-1]);
```

```
57      //用莫比乌斯函数解题常常需要求出形如:    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`

```cpp
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
```

```cpp
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator>(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator<=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
    friend bool operator>=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend std::ostream &operator<<(std::ostream &os, Frac x) {
        T g = std::gcd(x.num, x.den);
        if (x.den == g) {
            return os << x.num / g;
        } else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};

using F = Frac<int>;
```

5.4 POLYNOMIAL

前置知识：多项式求逆 + NTT

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

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

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

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

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

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

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

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

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

除去$F(x)$,得

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

两边平方，得

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

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

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

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$O(n \log_2^2 n)$

Figure 4: 分治 fft.png

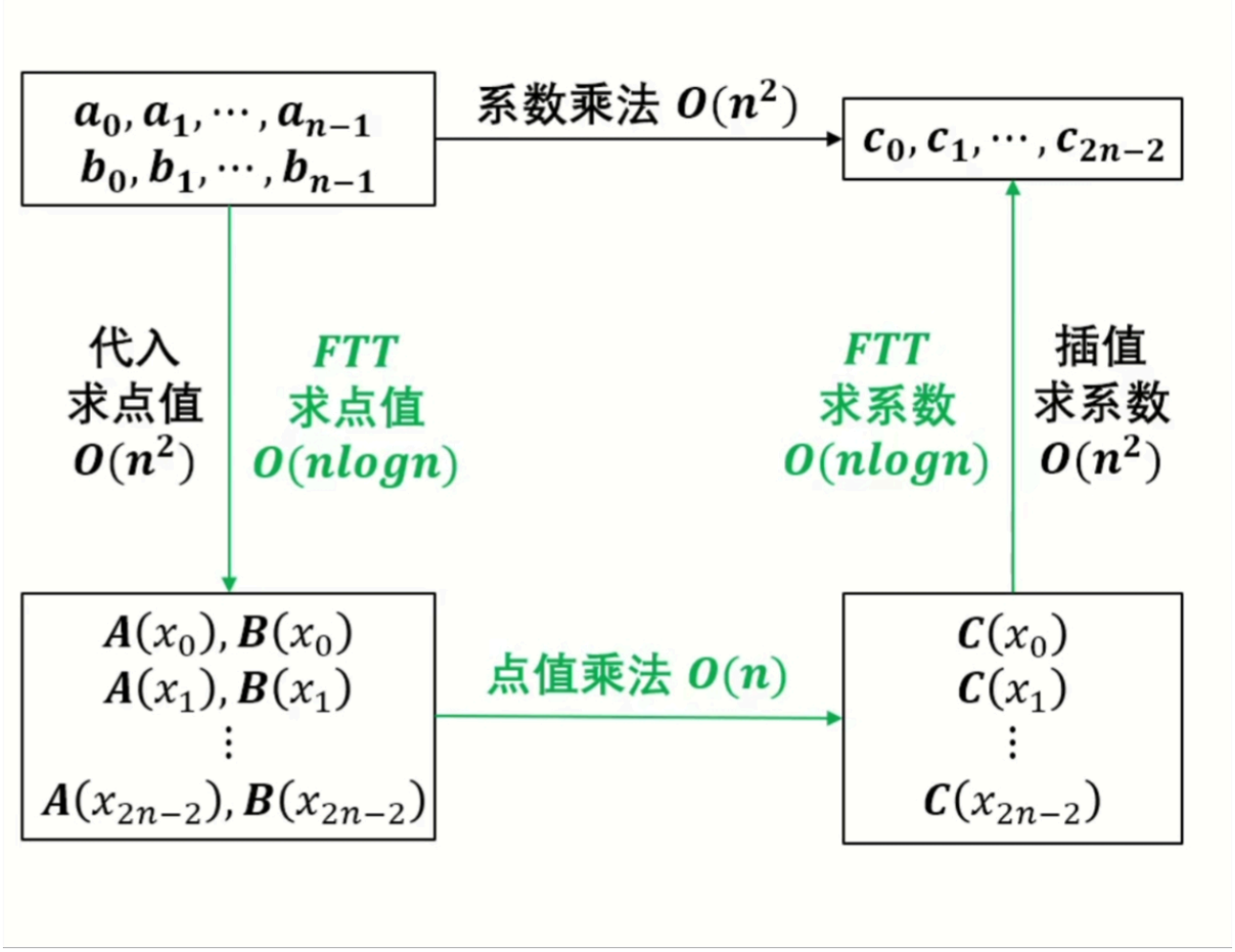

Figure 5: FFT 思想图解.png

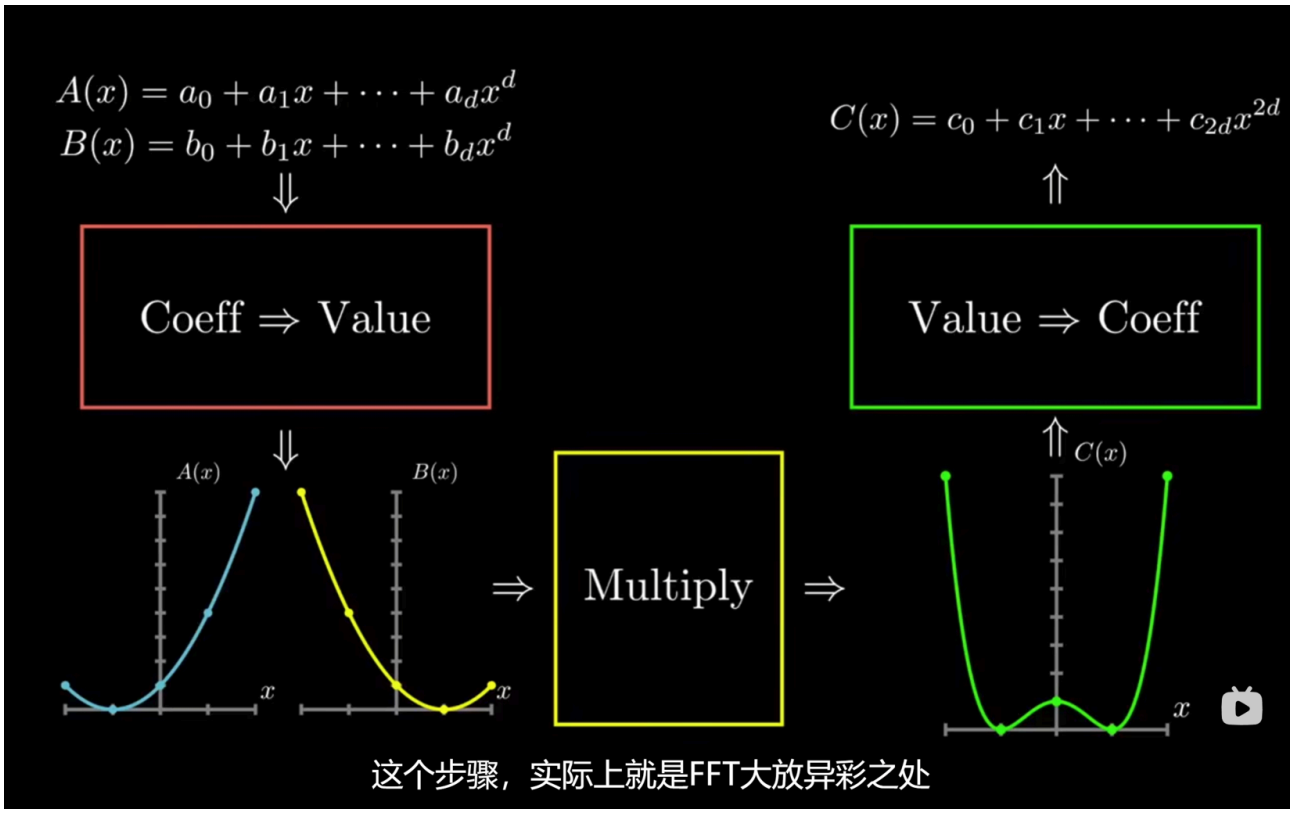

Figure 6: FFT 思想图解 2.png

5.4.1 AxB_prob.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define ld double
#define inf 10000000
#define mod 998244353
const ld pi=acos(-1);
signed main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    void solve();
    solve();
    return 0;
}
//高精乘法
//但是 FFT
//注意只能 FFT，不能 NTT,NTT 会乱
//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
const int N=(1<<22)|10;
string s1,s2;
int n,m;
int rev[N]={0};
struct comp    //手搓复数
{
    ld r=0,i=0;
}a[N],b[N];
ll c[N];
comp mul(comp x,comp y)
{
    comp res;
    res.r=x.r*y.r-x.i*y.i;
    res.i=x.r*y.i+x.i*y.r;
    return res;
}
comp add(comp x,comp y,int op)
{
    comp res;
    res.r=x.r+op*y.r;
    res.i=x.i+op*y.i;
    return res;
}

void chang(comp *A,int n)
{
    for(int i=0;i<n;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
    for(int i=0;i<n;i++)if(i<rev[i])swap(A[i],A[rev[i]]);
}
void FFT(comp *A,int n,int op)
{
    chang(A,n);
    for(int mid=1;mid<n;mid<<=1)
    {
        comp g1;
        g1.r=cos(pi/(ld)mid);
        g1.i=sin(op*pi/(ld)mid);
        for(int j=0;j<n;j+=(mid<<1))
```

```
        {
            comp gk;
            gk.r=1;gk.i=0;
            for(int k=0;k<mid;k++,gk=mul(gk,g1))
            {
                comp x=A[j+k],y=mul(A[j+k+mid],gk);
                A[j+k]=add(x,y,1);
                A[j+k+mid]=add(x,y,-1);
            }
        }
    }
    if(op==1)return ;
    for(int i=0;i<n;i++)c[i]=(ll)(A[i].r/n+0.5);
}

void solve()
{
    stack<int> ans;
    cin>>s1;
    cin>>s2;
    n=s1.length();
    m=s2.length();
    for(int i=0;i<n;i++)a[n-1-i].r=s1[i]-'0',a[n-1-i].i=0;
    for(int i=0;i<m;i++)b[m-1-i].r=s2[i]-'0',b[n-1-i].i=0;
    int mx=1;
    while(mx<n+m)mx<<=1;
    for(int i=n;i<mx;i++)a[i].i=0,a[i].r=0;
    for(int i=m;i<mx;i++)b[i].i=0,b[i].r=0;
    FFT(a,mx,1);FFT(b,mx,1);
    for(int i=0;i<mx;i++)a[i]=mul(a[i],b[i]);
    FFT(a,mx,-1);
    for(int i=0;i<mx;i++)
    {
        if(c[i]>=10)
        {
            c[i+1]+=(ll)c[i]/10;
            c[i]%=10;
        }
        ans.push(c[i]);
    }
    bool flag=true;
    while(!ans.empty())
    {
        int temp=ans.top();
        ans.pop();
        if(temp==0&&flag)continue;
        flag=false;
        cout<<temp;
    }
    cout<<"\n";
}
```

5.4.2 CDQ+NTT_FTT.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
#define mod 998244353
const int N=1<<22;
ll ge=3,gi;
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int t =1;
    void solve();
    // cin>>t;
    while(t--)solve();
}

//CDQ+NTT/FFT
//可用于处理卷积

//若我们求得了左区间，即可求出左区间对于右区间的贡献
//而这个贡献可通过 NTT 求出
//这就是所谓的分治 FFT（用 NTT 精度更高）
ll f[N],g[N];
int rev[N]={0};
ll ksm(ll a,ll b)
{
    ll ans=1;
    while(b)
    {
        if(b&1)ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}
void chang(ll *A,int n)
{
    for(int i=0;i<n;i++)
    {
        rev[i]=rev[i>>1]>>1;
        rev[i]|=(i&1)?(n>>1):0;
    }
    for(int i=0;i<n;i++)
    {
        if(i<rev[i])swap(A[i],A[rev[i]]);
    }
}
void NTT(ll *A,int n,int opt)
{
    chang(A,n);
    for(int mid=1;mid<n;mid<<=1)
    {
        ll g1=ksm((opt==1)?ge:gi,(mod-1)/(mid<<1));
        for(int R=mid<<1,j=0;j<n;j+=R)
        {
            ll gk=1;
```

```
            for(int k=0;k<mid;k++,gk=gk*g1%mod)
            {
                ll x=A[j+k],y=A[j+k+mid]*gk%mod;
                A[j+k]=(x+y)%mod;
                A[j+k+mid]=(x-y+mod)%mod;
            }
        }
    }
}
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
{
    NTT(A,n,1);NTT(B,n,1);

    for(int i=0;i<n;i++)A[i]=A[i]*B[i]%mod;

    NTT(A,n,-1);

    ll inv=ksm(n,mod-2);

    for(int i=0;i<n;i++)A[i]=A[i]*inv%mod;
}
ll ta[N],tb[N];
void CDQ(int l,int r)
{
    int mid=(l+r)>>1;
    if(l==r)return ;
    CDQ(l,mid);
    ll mx=1;
    while(mx<(mid-l+r-l)+1)mx<<=1;  //取不小于区间长度的 二的幂

    for(int i=0;i<mx;i++)ta[i]=tb[i]=0;  //初始化

    for(int i=l;i<=mid;i++)ta[i-l]=f[i];

    for(int i=1;i<=r-l;i++)tb[i]=g[i];

    mul(ta,tb,mx);  //ta 乘 tb

    for(int i=mid+1;i<=r;i++)f[i]=(f[i]+ta[i-l])%mod;//算贡献

    CDQ(mid+1,r);
}
void solve()
{
    gi=ksm(ge,mod-2);
    int n;
    cin>>n;
    for(int i=1;i<n;i++)
    {
        cin>>g[i];
        f[i]=0;
    }
    f[0]=1;  //这是 f 的边界
    g[0]=0;
    CDQ(0,n-1);  //分治
    for(int i=0;i<n;i++)cout<<f[i]<<" ";
    cout<<"\n";
```

```
114 }
```

5.4.3 FFT_butterfly.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define ld long double
#define inf 0x3f3f3f3f
#define mod 1000000007
const ld PI=acos(-1.0);  //取 PI
const ll N=1<<22;  //注意这个 N
typedef complex<ld> comp;
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int t =1;
    void solve();
    // cin>>t;
    while(t--)solve();
}
//FFT 最基础的板子
//FFT+蝴蝶优化
//规避了动态数组带来的时间复杂度
//O(n*logn)
//FFT 需要运用到复数，建议手写复数，会更快
//FFT 会有精度问题
int mx=0;
int rev[N]={0};
ll p1[N],p2[N];
comp tmp1[N],tmp2[N];

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
{
    for(int i=0;i<len;i++) //二进制翻转
    {
        rev[i]=rev[i>>1]>>1;
        if(i&1)rev[i]|=len>>1;
    }
    for(int i=0;i<len;i++)
    {
        if(i<rev[i])swap(tmp[i],tmp[rev[i]]);  //交换
    }
}

void FFT(comp *f,int n,int op)
{
    chang(f,n);  //先变换
    //采用非递归方法求解
    for(int mid=1;mid<n;mid<<=1)   //遍历交换中点，亦是半周期
    {
        comp w(cos(PI/mid),sin(PI*op/mid));   //单位根

        for(int R=mid<<1,j=0;j<n;j+=R)   //周期为 R,区间起点 j
        {
            comp cur(1,0);  //变化的自变量

            for(int k=0;k<mid;k++,cur*=w)   //傅里叶变换
            {

```

```
                comp x=f[j+k],y=cur*f[j+k+mid];

                f[j+k]=x+y;      //直接覆写

                f[j+k+mid]=x-y;  //直接覆写
            }
        }
    }
}

void solve()
{
    int n,m;
    cin>>n>>m;
    for(int i=0;i<=n;i++)
    {
        cin>>p1[i];
        tmp1[i]=p1[i];
    }
    for(int i=0;i<=m;i++)
    {
        cin>>p2[i];
        tmp2[i]=p2[i];
    }
    mx=1;
    while(mx<n+m+1)mx<<=1;

    for(int i=n+1;i<=mx;i++)
    {
        p1[i]=0;  // 高位补上 0,保证是 2 的幂次
        tmp1[i]=p1[i];
    }
    for(int i=m+1;i<=mx;i++)
    {
        p2[i]=0;
        tmp2[i]=p2[i];
    }

    FFT(tmp1,mx,1);  //两个 fft 求出点值式
    FFT(tmp2,mx,1);

    for(int i=0;i<=mx;i++)tmp1[i]=tmp1[i]*tmp2[i];  //值相乘

    FFT(tmp1,mx,-1);  //乘完后用 fft 转化为系数式

    for(int i=0;i<n+m+1;i++)cout<<(ll)((ld)tmp1[i].real()/(ld)mx+0.5)<<"
";  //注意实部取四舍五入

    cout<<"\n";
}
```

5.4.4 NTT.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define ld long double
#define inf 0x3f3f3f3f
#define mod 998244353
const ll N=1<<22;  //注意这个 N
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int t =1;
    void solve();
    // cin>>t;
    while(t--)solve();
}
//NTT 最基础的板子
//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
//由此诞生了快速数论变化 NTT
//换个根就行了
ll g=1,gi;  //g 是原根，一般取 3，gi 是 g 的乘法逆元
ll tmp1[N],tmp2[N];
ll mx;
int rev[N]={0};

ll ksm(ll a,ll b)  //快速幂
{
    ll ans=1;
    while(b)
    {
        if(b&1)ans=ans*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ans;
}
void chang(ll *f,int n)  //蝴蝶变换
{
    for(int i=0;i<n;i++)
    {
        rev[i]=rev[i>>1]>>1;
        rev[i]|=(i&1)?(n>>1):0;
    }
    for(int i=0;i<n;i++)
     if(i<rev[i])swap(f[i],f[rev[i]]);
}

void NTT(ll *f,int n,int opt) //快速数论变换
{
    chang(f,n);//先变换
    for(int mid=1;mid<n;mid<<=1)  //枚举半周期
    {
        ll g1=ksm((opt==1)?g:gi,(mod-1)/(mid*2));  //取单位根

        for(int R=mid<<1,j=0;j<n;j+=R)
        {

```

```cpp
            ll w=1;  //变化的自变量

            for(int k=0;k<mid;k++,w=(w*g1)%mod)
            {
                ll a1=f[j+k],a2=f[j+k+mid]*w%mod;

                f[j+k]=(a1+a2)%mod;

                f[j+k+mid]=(a1-a2+mod)%mod;
            }
        }
    }
}
void solve()
{
    int n,m;
    cin>>n>>m;

    g=3;gi=ksm(g,mod-2);  //原根 和 它的倒数

    for(int i=0;i<=n;i++)cin>>tmp1[i];
    for(int i=0;i<=m;i++)cin>>tmp2[i];

    mx=1;

    while(mx<n+m+1)mx<<=1;  //取不小于最高次数的二的幂

    for(int i=n+1;i<mx;i++)tmp1[i]=0;  //高位补 0
    for(int i=m+1;i<mx;i++)tmp2[i]=0;

    NTT(tmp1,mx,1);  //分别求点值式
    NTT(tmp2,mx,1);

    for(int i=0;i<mx;i++)tmp1[i]=tmp1[i]*tmp2[i]%mod;  //值相乘

    NTT(tmp1,mx,-1);  //点值式转系数式

    ll inv=ksm(mx,(mod-2));  //取 mx 的逆元
    for(int i=0;i<n+m+1;i++)
    {
        cout<<tmp1[i]*inv%mod<<" ";  //这里不要用除法，用乘法逆元
    }
    cout<<"\n";
}
```

5.4.5 NTT_INV.h

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define mod 998244353
const int N=(1<<22)|10;
int gen=3,gi;
int ksm(int a,int b)
{
    int ans=1;
    while(b)
    {
        if(b&1)ans=(ll)ans*a%mod;
        a=(ll)a*a%mod;
        b>>=1;
    }
    return ans;
}
signed main()
{
    void solve();
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    gi=ksm(gen,mod-2);
    solve();
    return 0;
}
//多项式求逆元，只能用 NTT
int a[N],rev[N]={0},b[N],tmp[N];
void chang(int *A,int n)  //蝴蝶变换
{
    for(int i=0;i<n;i++)
    {
        rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
    }
    for(int i=0;i<n;i++)
    {
        if(i<rev[i])swap(A[i],A[rev[i]]);
    }
}
void NTT(int *A,int n,int opt)
{
    chang(A,n);  //先变换
    for(int mid=1;mid<n;mid<<=1)
    {
        int g1=ksm((opt==1)?gen:gi,(mod-1)/(mid<<1));
        for(int j=0;j<n;j+=(mid<<1))
        {
            int gk=1;
            for(int k=0;k<mid;k++,gk=(ll)gk*g1%mod)
            {
                int x=A[j+k],y=(ll)gk*A[j+k+mid]%mod;
                A[j+k]=((ll)x+(ll)y)%mod;
                A[j+k+mid]=((ll)x-(ll)y+mod)%mod;
            }
        }
    }
    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]=(ll)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-
    (ll)tmp[i]*B[i]%mod+mod)%mod*B[i]%mod;  //套公式计算答案值
83
84          NTT(B,mx,-1);  //点值式转系数式
85
86          for(int i=n;i<mx;i++)B[i]=0;  //高位补 0
87      }
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

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define inf 0x3f3f3f3f
#define mod 998244353
int ksm(int a,int b)
{
    int ans=1;
    while(b)
    {
        if(b&1)ans=(ll)ans*a%mod;
        a=(ll)a*a%mod;
        b>>=1;
    }
    return ans;
}
//多项式开根号，需要 NTT+INV+二次剩余(cipolla)
const int gen=3,gi=ksm(gen,mod-2),N=(1<<22)|10,ny2=ksm(2,mod-2);
int main()
{
    void solve();
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    solve();
    return 0;
}
struct comp{int r,i;};
comp image_mul(comp a,comp b,int w)
{
    comp ans;
    ans.r=((ll)a.r*b.r%mod+(ll)a.i*b.i%mod*w%mod)%mod;
    ans.i=((ll)a.r*b.i%mod+(ll)a.i*b.r%mod)%mod;
    return ans;
}
comp ksm_image(comp a,int b,int w)
{
    comp res;
    res.r=1;res.i=0;
    while(b)
    {
        if(b&1)res=image_mul(res,a,w);
        a=image_mul(a,a,w);
        b>>=1;
    }
    return res;
}
int cipolla(int n)  //求二次剩余，仅用于求常数 mod 意义下的二次方根
{
    int w,a=rand()%mod;
    w=((ll)a*a-n+mod)%mod;
    if(ksm(n,(mod-1)>>1)==0)return 0;
    while(ksm(w,(mod-1)>>1)!=mod-1)a=rand()%mod,w=((ll)a*a-n+mod)%mod;
    comp ans;
    ans.r=a;
    ans.i=1;
    ans=ksm_image(ans,((ll)mod+1)>>1,w);
    int a1=ans.r,a2=(mod-a1)%mod;
```

```
    if(a1>a2)swap(a1,a2);
    return a1;
}

int a[N],b[N],c[N],d[N],e[N],rev[N]={0};
void chang(int *A,int n)
{
    for(int i=0;i<n;i++)rev[i]=(rev[i>>1]>>1)|((i&1)?(n>>1):0);
    for(int i=0;i<n;i++)if(i<rev[i])swap(A[i],A[rev[i]]);
}
void NTT(int *A,int n,int f)  //多项式相乘
{
    chang(A,n);
    for(int mid=1;mid<n;mid<<=1)
    {
        int g1=ksm((f==1)?gen:gi,(mod-1)/(mid<<1));
        for(int j=0;j<n;j+=(mid<<1))
        {
            int gk=1;
            for(int k=0;k<mid;k++,gk=(ll)gk*g1%mod)
            {
                int x=A[j+k],y=(ll)A[j+k+mid]*gk%mod;
                A[j+k]=((ll)x+(ll)y)%mod;
                A[j+k+mid]=((ll)x-(ll)y+mod)%mod;
            }
        }
    }
    if(f==1)return ;
    int inv=ksm(n,mod-2);
    for(int i=0;i<n;i++)A[i]=(ll)A[i]*inv%mod;
}
void INV(int n,int *A,int *B)//多项式求逆
{
    stack<int> stk;
    while(n!=1){stk.push(n);n=(n+1)>>1;}
    stk.push(1);
    int mx=1;
    while(!stk.empty())
    {
        n=stk.top();
        stk.pop();
        if(n==1){B[0]=ksm(A[0],mod-2);continue;}
        while(mx<(n<<1))mx<<=1;
        for(int i=0;i<n;i++)c[i]=A[i];
        for(int i=n;i<mx;i++)c[i]=0;
        NTT(c,mx,1);NTT(B,mx,1);
        for(int i=0;i<mx;i++)B[i]=(2ll-(ll)c[i]*B[i]%mod+mod)%mod*B[i]%mod;
        NTT(B,mx,-1);
        for(int i=n;i<mx;i++)B[i]=0;
    }
}
void SQRT(int n,int *A,int *B)  //多项式开根
{
    //思想和 INV 相同：分治、套公式
    stack<int> stk;
    while(n!=1){stk.push(n);n=(n+1)>>1;}
    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

```
// sz =int(sqrt(n)) 注意考虑 sqrtl(n)
struct node{
  int l,r,id;
  bool operator<(const node &x) const{
    if(l/sz!=x.l/sz) return l<x.l;
    if((l/sz)&1) return r<x.r;
    else return r>x.r;
  }
};
```

7 STRING

7.1 AC_automaton.h

```cpp
struct ACAutomaton
{
    static constexpr int N = 1e6 + 10;
    int ch[N][26], fail[N], cntNodes;
    int cnt[N];
    ACAutomaton()
    {
        cntNodes = 1;
    }
    void insert(string s)
    {
        int u = 1;
        for (auto c : s)
        {
            int &v = ch[u][c - 'a'];
            if (!v)
                v = ++cntNodes;
            u = v;
        }
        cnt[u]++;
    }
    void build()
    {
        fill(ch[0], ch[0] + 26, 1);
        queue<int> q;
        q.push(1);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
            {
                int &v = ch[u][i];
                if (!v)
                    v = ch[fail[u]][i];
                else
                {
                    fail[v] = ch[fail[u]][i];
                    q.push(v);
                }
            }
        }
    }
    LL query(string t)
    {
        LL ans = 0;
        int u = 1;
        for (auto c : t)
        {
            u = ch[u][c - 'a'];
            for (int v = u; v && ~cnt[v]; v = fail[v])
            {
                ans += cnt[v];
```

```
                    cnt[v] = -1;
                }
            }
            return ans;
        }
};
```

7.2 compress_print.h

```cpp
const int N = 1 << 21;
static const int mod1 = 1E9 + 7, base1 = 127;
static const int mod2 = 1E9 + 9, base2 = 131;
vector<int> val1;
vector<int> val2;
void init(int n = N)
{
    val1.resize(n + 1), val2.resize(n + 2);
    val1[0] = 1, val2[0] = 1;
    for (int i = 1; i <= n; i++)
    {
        val1[i] = val1[i - 1] * base1 % mod1;
        val2[i] = val2[i - 1] * base2 % mod2;
    }
}

string compress(vector<string> in)
{ // 前后缀压缩
    vector<int> h1{1};
    vector<int> h2{1};
    string ans = "#";
    for (auto s : in)
    {
        s = "#" + s;
        int st = 0;
        int chk1 = 0;
        int chk2 = 0;
        for (int j = 1; j < s.size() && j < ans.size(); j++)
        {
            chk1 = (chk1 * base1 % mod1 + s[j]) % mod1;
            chk2 = (chk2 * base2 % mod2 + s[j]) % mod2;
            if ((h1.back() == (h1[ans.size() - 1 - j] * val1[j] % mod1+ chk1) % mod1) &&
                (h2.back() == (h2[ans.size() - 1 - j] * val2[j] % mod2+ chk2) % mod2)    )
                st = j;
        }
        for (int j = st + 1; j < s.size(); j++)
        {
            ans += s[j];
            h1.push_back((h1.back() * base1 % mod1 + s[j]) % mod1);
            h2.push_back((h2.back() * base2 % mod2 + s[j]) % mod2);
        }
    }
    return ans.substr(1);
}
```

7.3 get_occr.h

```cpp
#include <template_overAll.h>

/*
 * 找到某一堆短字符串在长字符串中的出现位置
 * 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
 * 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|
 * 则时间复杂度为 O(max(|_s|log(|_s|),|s|))
 */
class get_occr
{
private:
    string s;
public:
    get_occr(string _s) { s = _s; }
    vector<int> locate(vector<string> _s,bool dira = 1)
    {
        int n = _s.size();
        vector<int> occr(n,-1);
        map<char,vector<pair<int,int>>> gncing;
        if(dira == 1)
        {
            for(int i = 0;i < n;i++)
                gncing[_s[i][0]].push_back({i,0});
            for(int i = 0;i < s.size();i ++)
            {
                vector<pair<int,int>> gnctmp = gncing[s[i]];
                gncing[s[i]].clear();
                for(int j = 0;j < gnctmp.size();j ++)
                {
                    if(gnctmp[j].se+1 < _s[gnctmp[j].fi].size())
                            gncing[_s[gnctmp[j].fi]
[gnctmp[j].se+1]].push_back({gnctmp[j].fi,gnctmp[j].se+1});
                    else occr[gnctmp[j].fi] = i;
                }
            }
        } else {
            for(int i = 0;i < n;i++) gncing[_s[i]
[_s[i].size()-1]].push_back({i,_s[i].size()-1});
            for(int i= s.size()-1;i >=0;i --)
            {
                vector<pair<int,int>> gnctmp = gncing[s[i]];
                gncing[s[i]].clear();
                for(int j = 0;j < gnctmp.size();j ++)
                {
                    if(gnctmp[j].se -1 >= 0)
                            gncing[_s[gnctmp[j].fi]
[gnctmp[j].se-1]].push_back({gnctmp[j].fi,gnctmp[j].se-1});
                    else occr[gnctmp[j].fi] = i;
                }
            }
        }
        return occr;
    }
};
```

7.4 hash_print.h

```cpp
#define int long long
const int N = 1 << 21;
static const int mod1 = 1E9 + 7, base1 = 127;
static const int mod2 = 1E9 + 9, base2 = 131;
vector<int> val1;
vector<int> val2;
using puv = pair<int,int>;
void init(int n = N)
{
    val1.resize(n + 1), val2.resize(n + 2);
    val1[0] = 1, val2[0] = 1;
    for (int i = 1; i <= n; i++)
    {
        val1[i] = val1[i - 1] * base1 % mod1;
        val2[i] = val2[i - 1] * base2 % mod2;
    }
}
class hstring
{
public:
    vector<int> h1;
    vector<int> h2;
    string s;

    hstring(string s_) : s(s_), h1{0}, h2{0}
    {
        build();
    }

    void build()
    {
        for (auto it : s)
        {
            h1.push_back((h1.back() * base1 % mod1 + it) % mod1);
            h2.push_back((h2.back() * base2 % mod2 + it) % mod2);
        }
    }

    puv get()
    { // 输出整串的哈希值
        return {h1.back(), h2.back()};
    }

    puv substring(int l, int r)
    { // 输出子串的哈希值
        if (l > r) swap(l, r);
        int ans1 = (mod1 + h1[r + 1] - h1[l] * val1[r - l + 1] % mod1) % mod1;
        int ans2 = (mod2 + h2[r + 1] - h2[l] * val2[r - l + 1] % mod2) % mod2;
        return {ans1, ans2};
    }

    puv modify(int idx, char x)
    { //修改 idx 位为 x
        int n = s.size() - 1;
        int ans1 = (h1.back() + val1[n - idx] * (x - s[idx]) % mod1) % mod1;
        int ans2 = (h2.back() + val2[n - idx] * (x - s[idx]) % mod2) % mod2;
        return {ans1, ans2};
```

```
58         }
59 };
```

7.5 KMP.h

```cpp
#include <template_overAll.h>

class KMP
{
private:
    string s;
    string inis;
public:
    vector<int> pi;
    KMP(string _s)
    {
        s = _s;
        inis = s;
    }
    void prefix_function()
    {
        pi.clear();
        int n = (int)s.length();
        pi.resize(n);
        for (int i = 1; i < n; i++)
        {
            int j = pi[i - 1];
            while (j > 0 && s[i] != s[j])
                j = pi[j - 1];
            if (s[i] == s[j])
                j++;
            pi[i] = j;
        }
        return;
    }
    vector<int> find_occr(string p)
    {
        s = inis;
        s = p + "#" + s;
        prefix_function();
        vector<int> v;
        for (int i = p.size() + 1; i < s.size(); i++)
            if (pi[i] == p.size())
                v.push_back(i - 2 * p.size());
        return v;
    }
};
```

7.6 Manacher.h

```cpp
pair<vector<int>,vector<int>> Manacher(string s){
    // d1: a b [c:3] b a
    // d2: a b [b:2] a
    int n = s.size();
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;
        d1[i] = k--;
        if (i + k > r) l = i - k,r = i + k;
    }
    vector<int> d2(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) k++;
        d2[i] = k--;
        if (i + k > r) l = i - k - 1, r = i + k;
    }
    return {d1,d2};
}
```

7.7 Palindromic_automaton.h

```cpp
class PA {
 private:
  static const int N = 100010;
  struct Node {
    int len;
    int ptr[26], fail;
    Node(int len = 0) : len(len), fail(0) { memset(ptr, 0, sizeof(ptr)); }
  } nd[N];

  int size, cnt;  // size 为字符串长度，cnt 为节点个数
  int cur;  //当前指针停留的位置，即最后插入字符所对应的节点
  char s[N];

  int getfail(int x)  //沿着 fail 指针找到第一个回文后缀
  {
    while (s[size - nd[x].len - 1] != s[size]) {
      x = nd[x].fail;
    }
    return x;
  }

 public:
  PA() : size(0), cnt(0), cur(0) {
    nd[cnt] = Node(0);
    nd[cnt].fail = 1;
    nd[++cnt] = Node(-1);
    nd[cnt].fail = 0;
    s[0] = '$';
  }

  void extend(char c) {
    s[++size] = c;
    int now = getfail(cur);  //找到插入的位置
    if (!nd[now].ptr[c - 'a'])  //若没有这个节点，则新建并求出它的 fail 指针
    {
      int tmp = ++cnt;
      nd[tmp] = Node(nd[now].len + 2);
      nd[tmp].fail = nd[getfail(nd[now].fail)].ptr[c - 'a'];
      nd[now].ptr[c - 'a'] = tmp;
    }
    cur = nd[now].ptr[c - 'a'];
  }

  int qlen() { return nd[cur].len; }
}
```

7.8 trie_Tree.h

```cpp
#include <template_overAll.h>

class Trie//AC
{
public:
    vector<map<char, int>> t;
    int root = 0;
    Trie()
    {
        t.resize(1);
    }
    void addedge(string _s)
    {
        int pvidx = root;
        _s.push_back('-');
        for (int i = 0; i < _s.size(); i++)
        {
            if (t[pvidx].find(_s[i]) != t[pvidx].end())
            {
                pvidx = t[pvidx][_s[i]];
            }
            else
            {
                t[pvidx][_s[i]] = t.size();
                t.push_back(map<char, int>());
                pvidx = t[pvidx][_s[i]];
            }
        }
    }
    bool ifcmp(string &s)
    {
        int pvidx = root;
        for(int i = 0;i < s.size();i ++)
        {
            if(t[pvidx].find(s[i]) != t[pvidx].end()) pvidx = t[pvidx][s[i]];
            else return 0;
        }
        return t[pvidx].find('-') != t[pvidx].end();
    }
};
```