南通大學 Nantong University

xCPC Team
Standard Code Library

Compiled on December 9, 2023

Contents

1. 准备

1.1 头文件

一开始打一份保存起来。

```cpp
#include <bits/stdc++.h>
using i64 = int64_t;

int main() {
#ifndef NTU
  | cin.tie(nullptr) -> sync_with_stdio(false);
#endif
}
```

1.2 热身赛

- 检查所需身份证件：护照、学生证、胸牌以及现场所需通行证。
- 确认什么东西能带进场，什么不能。特别注意：智能手表、金属（钥匙）等等。
- 测试鼠标、键盘、显示器和座椅。如果有问题，立刻联系工作人员。
- 确认比赛前能动什么，不能动什么，能存储什么配置文件。
- 测试本地栈大小：如果 `ulimit -a` 是 unlimited，那么在 bashrc 里加上 `ulimit -s 65536; ulimit -m 1048576`，否则死递归会死机。
- 测试比赛提交方式。如果有 `submit` 命令，确认如何使用。讨论是否应该不用以避免 `submit > a.cpp`。
- 如果可以的话，设置定期备份文件。
- 测试 OJ 栈大小。如果不合常理，发 clar 问一下。
- 测试提交编译器版本。如 C++17 `auto [x, y]: a`; C++14 `[](auto x, auto y)`; C++11 `auto`; `bits/stdc++.h`; `pb_ds`。

```cpp
#include <ext/rope>
using namespace __gnu_cxx;
rope <int> R;
R.insert(y, x);
R[x];
R.erase(x, 1);
```

- 测试 `__int128`, `__float128`, `sizeof (long double)`
- 测试代码长度限制；测试 output limit；测试 stderr limit。
- 测试内存限制：MLE 还是报 RE？栈溢出呢？
- 测试浮点数性能：FFT 能跑多快？测试内存性能：线段树、树状数组、素数筛能跑多快？测试 CPU 性能：阶乘、快速幂能跑多快？记得开 O2。
- 测试 clar：如果问不同类型的愚蠢的问题，得到的回复是否不一样？
- 测试 clock() 是否能够正常工作；测试本地性能与提交性能。
- 测试本地是否有 fsan, gdb。

```
address, undefined, return, shift,
  ↪ integer-divide-by-zero,
bounds-strict, float-cast-overflow, builtin
```

- 测试 Python, Java 本地环境与提交环境。Python 快吗？$A \times B$ 能跑多快？输入输出呢？
- 测试 `time` 命令是否能显示内存占用。

```
/usr/bin/time -v ./a.out
```

1.3 Path 环境变量

编辑 `.bashrc`，加入以下内容：

```
export CXX="g++"
export CXXFLAGS="-Wall -Wextra -Wconversion -Wshadow
  ↪ -std=c++20 -DNTU"
```

2. Data Structure

2.1 RMQ（ST表）

```cpp
struct RMQ {
  | int n;
  | vector<vector<int>> dp;
  | int better(const initializer_list<int> &list) {
```

```
    | | return max(list);
    | }
    | RMQ(const vector<int> &init)
    | | : n(init.size()), dp(n, vector<int>(__lg(n) + 1)) {
    | | for (int i = 0; i < n; ++i) { dp[i][0] = init[i]; }
    | | for (int k = 1; k <= __lg(n); ++k) {
    | | | for (int i = 0; i < n; ++i) {
    | | | | if (i + (1 << k) - 1 >= n) continue;
    | | | | int j = i + (1 << (k - 1));
    | | | | dp[i][k] = better({dp[i][k - 1], dp[j][k -
    |   ↪ 1]});
    | | | }}};
    | int query(int l, int r) {  // [l, r)
    | | int len = r - l;
    | | int k = __lg(len);
    | | return better({dp[l][k], dp[r - (1 << k)][k]});
    | }};
```

2.2 并查集

```
struct DSU {  // [0, n)
  | int n;
  | std::vector<int> fa, size;
  | DSU(int n) : n(n) {
  | | fa.resize(n);
  | | std::iota(fa.begin(), fa.end(), 0);
  | | size.assign(n, 1);
  | }
  | void Union(const int lp, const int rp) {  // put v under
  |   ↪ u
  | | fa[rp] = lp;
  | | size[lp] += size[rp];
  | }
  | bool Compare(const int lp, const int rp) { return
  |   ↪ size[lp] > size[rp]; }
  | int find(int p) {
  | | while (fa[p] != p) {
  | | | p = fa[p] = fa[fa[p]];
  | | }
  | | return p;
  | }
  | bool same(int p, int q) { return find(p) == find(q); }
  | bool merge(int p, int q) {
  | | int u = find(p), v = find(q);
  | | if (u == v) return false;
  | | if (!Compare(u, v)) std::swap(u, v);
  | | Union(u, v);
  | | return true;
  | }
};
```

```
int n,m,a,b,c,fa[maxn];
void ini(){ for(int i=1;i<=n;i++) { fa[i]=i; }}
int find(int x){ if(fa[x] == x) return x; return
  ↪ fa[x]=find(fa[x]); }
void join(int x, int y){ fa[find(x)]=find(y); }
```

2.2.1 可撤销并查集

```
struct DSU{
  | int fa[MAXN],rk[MAXN];
  | vector<pair<int*,int>> stk;
  | void init(int n) { stk.clear(); for(int i=1;i<=n;i++)
  |   ↪ fa[i]=i,rk[i]=1; }
  | int find(int x) { if(x==fa[x]) return x; return
  |   ↪ find(fa[x]); }
  | bool join(int x,int y) {
  | | int rx=find(x),ry=find(y);
  | | if(rx==ry) return false; if(rk[rx]>rk[ry])
  |   ↪ swap(rx,ry);
  | | stk.emplace_back(fa+rx,rx); fa[rx]=ry;
  | | stk.emplace_back(rk+ry,rk[ry]); rk[ry]+=rk[rx];
  | | return true; }
  | void withdraw() {
  | | *stk.back().first=stk.back().second;
  | | stk.pop_back();
  | | *stk.back().first=stk.back().second;
  | | stk.pop_back();
  | }
}dsu;
```

2.3 树状数组

2.3.1 树状数组

```
using i64 = long long;
template <typename T>
struct Fenwick {
  | int n;
  | std::vector<T> a;
  | Fenwick(int n) : n(n), a(n, T()) {}
  | void add(int x, T v) {
  | | for (int i = x + 1; i <= n; i += i & -i) {
  | | | a[i - 1] += v;
  | | }
  | }
  | T sum(int x) {
  | | T ans;
  | | for (int i = x; i > 0; i -= i & -i) {
  | | | ans += a[i - 1];
  | | }
  | | return ans;
  | }
  | T rangeSum(int l, int r) { return sum(r) - sum(l); }
  | int kth(T k) {
  | | int x = 0;
  | | for (int i = 1 << std::__lg(n); i; i /= 2) {
  | | | if (x + i <= n && k >= a[x + i - 1]) {
  | | | | x += i;
  | | | | k -= a[x - 1];
  | | | }
  | | }
  | | return x;
  | }
};
struct Max {
  | int v;
  | Max(int x = -1E9) : v{x} {}
  | Max &operator+=(Max a) {
  | | v = std::max(v, a.v);
  | | return *this;
  | }
};
```

```
int lowbit(int x) { return x&(-x); }
void change(int x,int y) {
  | for(;x<=n;x+=lowbit(x)) b[x]+=y; }
int sum(int x){ int s=0;
  | for(;x>0;x-=lowbit(x)) s+=b[x];
  | return s; }
```

2.3.2 二维树状数组

```
int a[MAXN][MAXN];
int lowbit(int x){ return x&(-x); }
void change(int x,int y,int k){
  | for(int i=x;i<=n;i+=lowbit(i)){
  | | for(int j=y;j<=m;j+=lowbit(j)){
  | | | a[i][j]+=k; } } }
int sum(int x,int y){
  | int s=0;
  | for(int i=x;i>0;i-=lowbit(i)){
  | | for(int j=y;j>0;j-=lowbit(j)){
  | | | s+=a[i][j]; } } return s;}
```

2.4 线段树

2.4.1 区间加法

```
LL sum[MAXN<<2],tag[MAXN<<2];
void pushup(int x) { sum[x]=sum[x<<1]+sum[x<<1|1];}
void pushdown(int x,int l,int r) {
  | int m=(l+r)>>1;
  | sum[x<<1]+=tag[x]*(m-l+1);
  | tag[x<<1]+=tag[x];
  | sum[x<<1|1]+=tag[x]*(r-m);
  | tag[x<<1|1]+=tag[x];
  | tag[x]=0;
}
void build(int x,int l,int r,LL *a) {
  | tag[x]=0;
  | if(l==r) sum[x]=a[l];
  | else {
  | | int m=(l+r)>>1;
  | | build(x<<1,l,m,a);
```

```
    build(x<<1|1,m+1,r,a);
    pushup(x);
  }
}
void modify(int x,int l,int r,int ql,int qr,LL delta) {
  if(ql<=l && r<=qr) {
    sum[x]+=delta*(r-l+1);
    tag[x]+=delta;
    return;
  }
  if(tag[x]) pushdown(x,l,r);
  int m=(l+r)>>1;
  if(ql<=m) modify(x<<1,l,m,ql,qr,delta);
  if(m<qr) modify(x<<1|1,m+1,r,ql,qr,delta);
  // sum[x]=tag[x]*(r-l+1)+sum[x<<1]+sum[x<<1|1];
  pushup(x);
}
LL query(int x,int l,int r,int ql,int qr){
  if(ql<=l && r<=qr) return sum[x];
  if(tag[x]) pushdown(x,l,r);
  int m=(l+r)>>1;
  LL res=0;
  if(ql<=m) res+=query(x<<1,l,m,ql,qr);
  if(m<qr) res+=query(x<<1|1,m+1,r,ql,qr);
  // res+=tag[x]*(min(qr,r)-max(ql,l)+1);
  return res;
}
```

2.4.2 区间乘法

```
LL sum[MAXN<<2],ts[MAXN<<2],tm[MAXN<<2];
void pushup(int x) {sum[x]=(sum[x<<1]+sum[x<<1|1])%MOD;}
void pushdown(int x,int l,int r) {
  int m=(l+r)>>1;
  sum[x<<1]=(sum[x<<1]*tm[x]%MOD + ts[x]*(m-l+1)%MOD)%MOD;
  sum[x<<1|1]=(sum[x<<1|1]*tm[x]%MOD +
    ↪ ts[x]*(r-m)%MOD)%MOD;
  tm[x<<1]=tm[x<<1]*tm[x]%MOD;
  tm[x<<1|1]=tm[x<<1|1]*tm[x]%MOD;
  ts[x<<1]=(ts[x<<1]*tm[x]%MOD+ts[x])%MOD;
  ts[x<<1|1]=(ts[x<<1|1]*tm[x]%MOD+ts[x])%MOD;
  tm[x]=1;
  ts[x]=0;
}
void build(int x,int l,int r,LL* a) {
  ts[x]=0,tm[x]=1;
  if(l==r) sum[x]=a[l];
  else {
    int m=(l+r)>>1;
    build(x<<1,l,m,a);
    build(x<<1|1,m+1,r,a);
    pushup(x);
  }
}
void plus(int x,int l,int r,int ql,int qr,LL d) {
  if(ql<=l && r<=qr) {
    sum[x]=(sum[x]+d*(r-l+1)%MOD)%MOD;
    ts[x]=(ts[x]+d)%MOD;
    return;
  }
  pushdown(x,l,r);
  int m=(l+r)>>1;
  if(ql<=m) plus(x<<1,l,m,ql,qr,d);
  if(m<qr) plus(x<<1|1,m+1,r,ql,qr,d);
  pushup(x);
}
void multi(int x,int l,int r,int ql,int qr,LL d) {
  if(ql<=l && r<=qr) {
    sum[x]=(sum[x]*d)%MOD;
    ts[x]=(ts[x]*d)%MOD;
    tm[x]=(tm[x]*d)%MOD;
    return;
  }
  pushdown(x,l,r);
  int m=(l+r)>>1;
  if(ql<=m) multi(x<<1,l,m,ql,qr,d);
  if(m<qr) multi(x<<1|1,m+1,r,ql,qr,d);
  pushup(x);
}
LL query(int x,int l,int r,int ql,int qr) {
  if(ql<=l && r<=qr) {
    return sum[x];
  }
  pushdown(x,l,r);
  int m=(l+r)>>1;
  LL res=0;
  if(ql<=m) res=(res+query(x<<1,l,m,ql,qr))%MOD;
  if(m<qr) res=(res+query(x<<1|1,m+1,r,ql,qr))%MOD;
  return res;
}
```

2.4.3 区间加和乘

```
#define int long long
const int N = 1e5 + 10, M = 1e5 + 10;
int n, m, mod;
int A[N], jud;
struct type {
  int l, r, sum, a, m;
} tree[N * 4];
void pushup(int rt) {
  (tree[rt].sum = tree[rt * 2].sum + tree[rt * 2 + 1].sum)
    ↪ %= mod;
}
void eva(type& rt, int a, int m) {
  (rt.sum *= m) %= mod, (rt.sum += (rt.r - rt.l + 1) * a)
    ↪ %= mod;
  (rt.m *= m) %= mod, (rt.a *= m) %= mod, (rt.a += a) %=
    ↪ mod;
}
void pushdown(int rt) {
  eva(tree[rt * 2], tree[rt].a, tree[rt].m);
  eva(tree[rt * 2 + 1], tree[rt].a, tree[rt].m);
  tree[rt].a = 0, tree[rt].m = 1;
}
void build(int l, int r, int rt) {
  tree[rt] = {l, r, 0, 0, 1};
  if (l == r) {
    tree[rt] = {l, r, A[l], 0, 1};
    return;
  }
  int mid = (l + r) / 2;
  build(l, mid, rt * 2);
  build(mid + 1, r, rt * 2 + 1);
  pushup(rt);
}
void update(int l, int r, int x, int rt) {
  if (tree[rt].l >= l && tree[rt].r <= r) {
    if (jud == 1) {
      eva(tree[rt], 0, x);
      return;
    } else {
      eva(tree[rt], x, 1);
      return;
    }
  }
  pushdown(rt);
  int mid = (tree[rt].l + tree[rt].r) / 2;
  if (l <= mid) update(l, r, x, rt * 2);
  if (r >= mid + 1) update(l, r, x, rt * 2 + 1);
  pushup(rt);
}
int getsum(int l, int r, int rt) {
  if (tree[rt].l >= l && tree[rt].r <= r) return
    ↪ tree[rt].sum;
  pushdown(rt);
  int sum = 0, mid = (tree[rt].l + tree[rt].r) / 2;
  if (l <= mid) (sum += getsum(l, r, rt * 2)) %= mod;
  if (r >= mid + 1) (sum += getsum(l, r, rt * 2 + 1)) %=
    ↪ mod;
  return sum;
}
void oper() {
  cin >> n >> mod;
  for (int i = 1; i <= n; i++) cin >> A[i];
  build(1, n, 1);
  cin >> m;
  for (int i = 1; i <= m; i++) {
    char ch;
    cin >> ch;
    if (ch == '1') {
      int l, r, x;
      cin >> l >> r >> x;
      jud = 1;
      update(l, r, x, 1);
```

```
    } else if (ch == '2') {
      int l, r, x;
      cin >> l >> r >> x;
      jud = 2;
      update(l, r, x, 1);
    } else {
      int l, r;
      cin >> l >> r;
      cout << getsum(l, r, 1) << endl;
    }
  }
}
```

2.5 非递归线段树

2.5.1 区间加，区间求和

```
void update(int l, int r, int d) {
  int len = 1, cntl = 0, cntr = 0;
  for (l+=M-1, r+=M+1; l^r^1; l>>=1, r>>=1, len<<=1) {
    tree[l] += cntl * d, tree[r] += cntr * d;
    if (~l & 1) tree[l ^ 1] += d * len, mark[l ^ 1] += d, cntl += len;
    if (r & 1) tree[r ^ 1] += d * len, mark[r ^ 1] += d, cntr += len; }
  for (; l; l >>= 1, r >>= 1)
    tree[l] += cntl * d, tree[r] += cntr * d; }
int query(int l, int r) {
  int ans = 0, len = 1, cntl = 0, cntr = 0;
  for (l+=M-1, r+=M+1; l^r^1; l>>=1, r>>=1, len<<=1) {
    ans += cntl * mark[l] + cntr * mark[r];
    if (~l & 1) ans += tree[l ^ 1], cntl += len;
    if (r & 1) ans += tree[r ^ 1], cntr += len; }
  for (; l; l >>= 1, r >>= 1)
    ans += cntl * mark[l] + cntr * mark[r];
  return ans; }
```

2.5.2 区间加，区间求最大值

```
void update(int l, int r, int d) {
  for (l += M-1, r += M+1; l^r^1; l >>= 1, r >>= 1) {
    if (l < M) {
      tree[l] = max(tree[l*2], tree[l*2+1]) + mark[l];
      tree[r] = max(tree[r*2], tree[r*2+1]) + mark[r]; }
    if (~l & 1) { tree[l ^ 1] += d; mark[l ^ 1] += d; }
    if (r & 1) { tree[r ^ 1] += d; mark[r ^ 1] += d; } }
  for (; l; l >>= 1, r >>= 1)
    if (l < M) tree[l] = max(tree[l*2], tree[l*2+1]) + mark[l],
      tree[r] = max(tree[r*2], tree[r*2+1]) + mark[r]; }
int query(int l, int r) {
  int maxl = -INF, maxr = -INF;
  for (l += M-1, r += M+1; l^r^1; l >>= 1, r >>= 1) {
    maxl += mark[l]; maxr += mark[r];
    if (~l & 1) maxl = max(maxl, tree[l ^ 1]);
    if (r & 1) maxr = max(maxr, tree[r ^ 1]); }
  while (l) { maxl += mark[l]; maxr += mark[r];
    l >>= 1; r >>= 1; }
  return max(maxl, maxr); }
```

2.6 划分树

```
const int MAXN = 100010;
int tree[20][MAXN];  //表示每层每个位置的值
int sorted[MAXN];    //已经排序好的数
int toleft[20][MAXN];  //toleft[p][i] 表示第 i 层从 1 到 i 有数分入左边
void build(int l, int r, int dep) {
  if (l == r) return;
  int mid = (l + r) >> 1;
  int same = mid - l + 1;  //表示等于中间值而且被分入左边的个数
  for (int i = l; i <= r; i++)  //注意是 l, 不是 one
    if (tree[dep][i] < sorted[mid]) same--;
  int lpos = l;
  int rpos = mid + 1;
  for (int i = l; i <= r; i++) {
    if (tree[dep][i] < sorted[mid])
      tree[dep + 1][lpos++] = tree[dep][i];
    else if (tree[dep][i] == sorted[mid] && same > 0) {
      tree[dep + 1][lpos++] = tree[dep][i];
      same--;
    } else
      tree[dep + 1][rpos++] = tree[dep][i];
    toleft[dep][i] = toleft[dep][l - 1] + lpos - l;
  }
  build(l, mid, dep + 1);
  build(mid + 1, r, dep + 1);
}
//查询区间第 k 大的数,[L,R] 是大区间，[l,r] 是要查询的小区间
int query(int L, int R, int l, int r, int dep, int k) {
  if (l == r) return tree[dep][l];
  int mid = (L + R) >> 1;

  int cnt = toleft[dep][r] - toleft[dep][l - 1];
  if (cnt >= k) {
    int newl = L + toleft[dep][l - 1] - toleft[dep][L - 1];
    int newr = newl + cnt - 1;
    return query(L, mid, newl, newr, dep + 1, k);
  } else {
    int newr = r + toleft[dep][R] - toleft[dep][r];
    int newl = newr - (r - l - cnt);
    return query(mid + 1, R, newl, newr, dep + 1, k - cnt);
  }
}
int main() {
  int n, m;
  while (scanf("%d%d", &n, &m) == 2) {
    memset(tree, 0, sizeof(tree));
    for (int i = 1; i <= n; i++) {
      scanf("%d", &tree[0][i]);
      sorted[i] = tree[0][i];
    }
    sort(sorted + 1, sorted + n + 1);
    build(1, n, 0);
    int s, t, k;
    while (m--) {
      scanf("%d%d%d", &s, &t, &k);
      printf("%d\n", query(1, n, s, t, 0, k));
    }
  }
  return 0;
}
```

2.7 主席树

```
// 主席树 支持查询 [l,r] 区间第k大，以及区间内不重复数字个数
// M = maxn * 30;
int n, q, m, tot;       // n为数组大小，m为离散化后数组大小
int A[maxn], T[maxn];  // A为原数组，T为离散化数组
int tree[M], lson[M], rson[M], Cnt[M];  // Cnt[i] 表示节点i的子树包含数字的总数
void Init_hash() {
  for (int i = 1; i <= n; i++) T[i] = A[i];
  sort(T + 1, T + n + 1);
  m = unique(T + 1, T + n + 1) - T - 1;
}
inline int Hash(int x) {
  return lower_bound(T + 1, T + m + 1, x) - T;
}
int build(int l, int r) {
  int root = tot++;
  Cnt[root] = 0;
  if (l != r) {
    int mid = (l + r) >> 1;
    lson[root] = build(l, mid);
    rson[root] = build(mid + 1, r);
  }
  return root;
}
int update(int root, int pos, int val) {
  int newroot = tot++, tmp = newroot;
  Cnt[newroot] = Cnt[root] + val;
  int l = 1, r = m;
  while (l < r) {
    int mid = (l + r) >> 1;
    if (pos <= mid) {
      lson[newroot] = tot++;
      rson[newroot] = rson[root];
      newroot = lson[newroot];
      root = lson[root];
      r = mid;
    } else {
```

```
    rson[newroot] = tot++;
    lson[newroot] = lson[root];
    newroot = rson[newroot];
    root = rson[root];
    l = mid + 1;
  }
  Cnt[newroot] = Cnt[root] + val;
 }
 return tmp;
}
void init() {  // 查询l~r第k大
 Init_hash();
 tree[0] = build(1, m);
 for (int i = 1; i <= n; i++) {
  int pos = Hash(A[i]);
  tree[i] = update(tree[i - 1], pos, 1);
 }
}
int query(int lrt, int rrt,
    int k) {  // 查询l~r第k大：T[query(tree[l - 1],
      ↪ tree[r], k)]
 int l = 1, r = m;
 while (l < r) {
  int mid = (l + r) >> 1;
  if (Cnt[lson[rrt]] - Cnt[lson[lrt]] >= k) {
   r = mid;
   lrt = lson[lrt];
   rrt = lson[rrt];
  } else {
   l = mid + 1;
   k -= Cnt[lson[rrt]] - Cnt[lson[lrt]];
   lrt = rson[lrt];
   rrt = rson[rrt];
  }
 }
 return l;
}
void init() {  // 查询l~r内不重复数字个数
 tree[0] = build(1, n);
 map<int, int> mp;
 for (int i = 1; i <= n; i++) {
  if (mp.find(A[i]) == mp.end())
   tree[i] = update(tree[i - 1], i, 1);
  else {
   int tmp = update(tree[i - 1], mp[A[i]], -1);
   tree[i] = update(tmp, i, 1);
  }
  mp[A[i]] = i;
 }
}
int query(int root, int pos) {  // 查询l~r内不重复数字个数:
  ↪ query(tree[r], l)
 int ret = 0;
 int l = 1, r = n;
 while (pos > l) {
  int mid = (l + r) >> 1;
  if (pos <= mid) {
   ret += Cnt[rson[root]];
   root = lson[root];
   r = mid;
  } else {
   root = rson[root];
   l = mid + 1;
  }
 }
 return ret + Cnt[root];
}
```

2.7.1 在线区间第k大

```
// 主席树求 [l,r] 第k大，可单点修改 使用树状数组套主席树在线操
  ↪ 作，树状数组维护改变量
// M = maxn * 40;
int n, q, m, tot;
int A[maxn], T[maxn];
int tree[maxn], lson[M], rson[M], Cnt[M];
int Ntree[maxn],
 use[maxn];  // Ntree[i] 表示动态第i棵树的树根，use[i] 表示
    ↪ 第i个树根是谁在使用
struct Query {
 int kind;
 int l, r, k;
} query[10005];
```

```
void Init_hash(int k) {
 sort(T, T + k);
 m = unique(T, T + k) - T;
}
int Hash(int x) {
 return lower_bound(T, T + m, x) - T;
}
int build(int l, int r) {
 int root = tot++;
 Cnt[root] = 0;
 if (l != r) {
  int mid = (l + r) >> 1;
  lson[root] = build(l, mid);
  rson[root] = build(mid + 1, r);
 }
 return root;
}
int update(int root, int pos, int val) {
 int newroot = tot++, tmp = newroot;
 int l = 0, r = m - 1;
 Cnt[newroot] = Cnt[root] + val;
 while (l < r) {
  int mid = (l + r) >> 1;
  if (pos <= mid) {
   lson[newroot] = tot++;
   rson[newroot] = rson[root];
   newroot = lson[newroot];
   root = lson[root];
   r = mid;
  } else {
   rson[newroot] = tot++;
   lson[newroot] = lson[root];
   newroot = rson[newroot];
   root = rson[root];
   l = mid + 1;
  }
  Cnt[newroot] = Cnt[root] + val;
 }
 return tmp;
}
inline int lowbit(int x) {
 return x & (-x);
}
int sum(int x) {
 int ret = 0;
 while (x > 0) {
  ret += Cnt[lson[use[x]]];
  x -= lowbit(x);
 }
 return ret;
}
void Modify(int x, int pos, int val) {
 while (x <= n) {
  Ntree[x] = update(Ntree[x], pos, val);
  x += lowbit(x);
 }
}
int Query(int left, int right, int k) {
 int lrt = tree[left - 1];
 int rrt = tree[right];
 int l = 0, r = m - 1;
 for (int i = left - 1; i; i -= lowbit(i)) use[i] =
   ↪ Ntree[i];
 for (int i = right; i; i -= lowbit(i)) use[i] =
   ↪ Ntree[i];
 while (l < r) {
  int mid = (l + r) >> 1;
  // sum(right) - sum(left - 1) 为改变量，Cnt[lson[rrt]]
    ↪ - Cnt[lson[lrt]] 为基础差值
  int tmp = sum(right) - sum(left - 1) + Cnt[lson[rrt]]
    ↪ - Cnt[lson[lrt]];
  if (tmp >= k) {
   r = mid;
   for (int i = left - 1; i; i -= lowbit(i)) use[i] =
     ↪ lson[use[i]];
   for (int i = right; i; i -= lowbit(i)) use[i] =
     ↪ lson[use[i]];
   lrt = lson[lrt];
   rrt = lson[rrt];
  } else {
   l = mid + 1;
   k -= tmp;
```

```
    for (int i = left - 1; i; i -= lowbit(i)) use[i] = rson[use[i]];
    for (int i = right; i; i -= lowbit(i)) use[i] = rson[use[i]];
    lrt = rson[lrt];
    rrt = rson[rrt];
  }
 }
 return l;
}
int main() {
 int Tcase;
 char op[10];
 scanf("%d", &Tcase);
 while (Tcase--) {
  scanf("%d%d", &n, &q);
  tot = 0;
  m = 0;
  for (int i = 1; i <= n; i++) {
   scanf("%d", &A[i]);
   T[m++] = A[i];
  }
  for (int i = 0; i < q; i++) {
   scanf("%s", op);
   if (op[0] == 'Q') {
    query[i].kind = 0;
    scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
   } else {
    query[i].kind = 1;
    scanf("%d%d", &query[i].l, &query[i].r);
    T[m++] = query[i].r;
   }
  }
  Init_hash(m);
  tree[0] = build(0, m - 1);
  for (int i = 1; i <= n; i++)
   tree[i] = update(tree[i - 1], Hash(A[i]), 1);
  for (int i = 1; i <= n; i++) Ntree[i] = tree[0];
  for (int i = 0; i < q; i++) {
   if (query[i].kind == 0)
    printf("%d\n", T[Query(query[i].l, query[i].r, query[i].k)]);
   else {
    Modify(query[i].l, Hash(A[query[i].l]), -1);
    Modify(query[i].l, Hash(query[i].r), 1);
    A[query[i].l] = query[i].r;
   }
  }
 }
 return 0;
}
```

2.8 Splay

2.8.1 Splay Tree

```
int n, m;
int rt, w[N];
queue<int> q;
struct type {
 int s[2], v, p, rev, same;
 int sz, sum, ms, ls, rs;
 int &l = s[0], &r = s[1];
 void init(int vv, int pp) {
  l = r = 0, v = vv, p = pp;
  rev = same = 0;
  sz = 1, sum = ms = v;
  ls = rs = max(v, 0LL);
 }
} tree[N];
void pushup(int x) {
 auto &u = tree[x], &l = tree[u.l], &r = tree[u.r];
 u.sz = l.sz + r.sz + 1;
 u.sum = l.sum + r.sum + u.v;
 u.ls = max(l.ls, l.sum + u.v + r.ls);
 u.rs = max(r.rs, r.sum + u.v + l.rs);
 u.ms = max({l.ms, r.ms, l.rs + u.v + r.ls});
}
void pushdown(int x) {
 auto &u = tree[x], &l = tree[u.l], &r = tree[u.r];
 if (u.same) {
  u.same = u.rev = 0;
  if (u.l) l.same = 1, l.v = u.v, l.sum = l.v * l.sz;
  if (u.r) r.same = 1, r.v = u.v, r.sum = r.v * r.sz;
  if (u.v > 0) {
   if (u.l) l.ms = l.ls = l.rs = l.sum;
   if (u.r) r.ms = r.ls = r.rs = r.sum;
  } else {
   if (u.l) l.ms = l.v, l.ls = l.rs = 0;
   if (u.r) r.ms = r.v, r.ls = r.rs = 0;
  }
 } else if (u.rev) {
  u.rev = 0, l.rev ^= 1, r.rev ^= 1;
  swap(l.ls, l.rs), swap(r.ls, r.rs);
  swap(l.l, l.r), swap(r.l, r.r);
 }
}
void rotate(int x) {
 int y = tree[x].p, z = tree[y].p, k = tree[y].r == x;
 int& b = tree[x].s[k ^ 1];
 tree[b].p = y; tree[y].p = x; tree[x].p = z;
 tree[y].s[k] = b;
 b = y;
 tree[z].s[tree[z].r == y] = x;
 pushup(y); pushup(x);
}
void splay(int x, int k) {
 while (tree[x].p != k) {
  int y = tree[x].p, z = tree[y].p;
  if (z != k) {
   if ((tree[y].r == x) ^ (tree[z].r == y))
    rotate(x);
   else
    rotate(y);
  }
  rotate(x);
 }
 if (!k) rt = x;
}
int get(int k) {
 int x = rt;
 while (x) {
  pushdown(x);
  int sz = tree[tree[x].l].sz;
  if (sz >= k)
   x = tree[x].l;
  else if (sz + 1 == k)
   return x;
  else
   k -= sz + 1, x = tree[x].r;
 }
}
int build(int l, int r, int p) {
 int mid = (l + r) / 2;
 int x = q.front();
 q.pop();
 tree[x].init(w[mid], p);
 if (l < mid) tree[x].l = build(l, mid - 1, x);
 if (r > mid) tree[x].r = build(mid + 1, r, x);
 pushup(x);
 return x;
}
void dfs(int x) {
 if (tree[x].l) dfs(tree[x].l);
 if (tree[x].r) dfs(tree[x].r);
 q.push(x);
}
void oper() {
 cin >> n >> m;
 for (int i = 1; i < N; i++) q.push(i);
 tree[0].ms = w[0] = w[n + 1] = -inf;
 for (int i = 1; i <= n; i++) cin >> w[i];
 rt = build(0, n + 1, 0);
 for (int i = 1; i <= m; i++) {
  string s;
  cin >> s;
  if (s == "INSERT") {
   int pos, tot;
   cin >> pos >> tot;
   for (int i = 0; i < tot; i++) cin >> w[i];
   int l = get(pos + 1), r = get(pos + 2);
   splay(l, 0), splay(r, l);
   int x = build(0, tot - 1, r);
   tree[r].l = x;
   pushup(r);
```

```
    pushup(l);
  } else if (s == "DELETE") {
    int pos, tot;
    cin >> pos >> tot;
    int l = get(pos), r = get(pos + tot + 1);
    splay(l, 0), splay(r, l);
    dfs(tree[r].l);
    tree[r].l = 0;
    pushup(r);
    pushup(l);
  } else if (s == "MAKE-SAME") {
    int pos, tot, c;
    cin >> pos >> tot >> c;
    int l = get(pos), r = get(pos + tot + 1);
    splay(l, 0), splay(r, l);
    auto& son = tree[tree[r].l];
    son.same = 1, son.v = c, son.sum = c * son.sz;
    if (c > 0)
      son.ms = son.ls = son.rs = son.sum;
    else
      son.ms = c, son.ls = son.rs = 0;
    pushup(r);
    pushup(l);
  } else if (s == "REVERSE") {
    int pos, tot;
    cin >> pos >> tot;
    int l = get(pos), r = get(pos + tot + 1);
    splay(l, 0), splay(r, l);
    auto& son = tree[tree[r].l];
    son.rev ^= 1;
    swap(son.ls, son.rs);
    swap(son.l, son.r);
    pushup(r);
    pushup(l);
  } else if (s == "GET-SUM") {
    int pos, tot;
    cin >> pos >> tot;
    int l = get(pos), r = get(pos + tot + 1);
    splay(l, 0), splay(r, l);
    cout << tree[tree[r].l].sum << endl;
  } else
    cout << tree[rt].ms << endl;
 }
}
```

2.8.2 Splay Tree Flip

```
#include <cstdio>
#include <algorithm>
#define MAXN 100005
const int inf=0x3f3f3f3f;
using namespace std;
int root,len,a[MAXN];
struct node{
 int v,fa,ch[2],size,cnt,tag;
}sp[MAXN];
int getch(int x) {return sp[sp[x].fa].ch[1]==x;}
void pushup(int x) {
 sp[x].size=
 sp[x].cnt+sp[sp[x].ch[0]].size+sp[sp[x].ch[1]].size;}
void pushdown(int x){
 if(sp[x].tag){
  sp[sp[x].ch[0]].tag^=1;
  sp[sp[x].ch[1]].tag^=1;
  swap(sp[x].ch[0],sp[x].ch[1]);
  sp[x].tag=0;
 }
}
void rotate(int x){
 int f=sp[x].fa, ff=sp[f].fa;
 int k=getch(x);
 sp[ff].ch[getch(f)]=x; sp[x].fa=ff;
 sp[sp[x].ch[k^1]].fa=f; sp[f].ch[k]=sp[x].ch[k^1];
 sp[x].ch[k^1]=f; sp[f].fa=x;
 pushup(f); pushup(x);
}
void splay(int x,int goal=0){
 for(int f;(f=sp[x].fa)!=goal;rotate(x)){
  if(sp[f].fa!=goal)
   rotate(getch(x)==getch(f)?f:x);
 }
 if(!goal) root=x;
}
int find(int x){
 int cur=root;
 while(1){
  pushdown(cur);
  if(x<=sp[sp[cur].ch[0]].size)
   cur=sp[cur].ch[0];
  else if(x>sp[sp[cur].ch[0]].size+sp[cur].cnt){
   x-=sp[sp[cur].ch[0]].size+sp[cur].cnt;
   cur=sp[cur].ch[1];
  }
  else return cur;
 }
}
int built(int f,int l,int r){
 if(l>r) return 0;
 int mid=(l+r)>>1, cur=++len;
 sp[cur].fa=f;
 sp[cur].cnt=1;
 sp[cur].v=a[mid];
 sp[cur].tag=0;
 sp[cur].ch[0]=built(cur,l,mid-1);
 sp[cur].ch[1]=built(cur,mid+1,r);
 pushup(cur);
 return cur;
}
void flip(int l,int r){
 int last=find(l-1),next=find(r+1);
 splay(last);splay(next,last);
 sp[sp[sp[root].ch[1]].ch[0]].tag^=1;
}
void dfs(int cur){
 pushdown(cur);
 if(sp[cur].ch[0]) dfs(sp[cur].ch[0]);
 if(sp[cur].v!=-inf && sp[cur].v!=inf) printf("%d ",
   ↪ sp[cur].v);
 if(sp[cur].ch[1]) dfs(sp[cur].ch[1]);
}
int n,m;
int main(){
 scanf("%d %d", &n, &m);
 for(int i=1;i<=n;i++) a[i+1]=i;
 a[1]=-inf;a[n+2]=inf;
 len=0;
 root=built(0,1,n+2);
 sp[0].size=0;
 for(int i=1;i<=m;i++){
  int l,r;
  scanf("%d %d", &l, &r);
  flip(l+1,r+1);
 }
 dfs(root);
 return 0;
}
```

2.8.3 Splay Tree Dye & Flip

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define MAXN 500005
const int inf=0x3f3f3f3f;
using namespace std;
struct node{
 int v,fa,ch[2],cnt;//basic
 int size,sum,lm,rm,mm;//pushup
 int flip,color;//pushdown
}sp[MAXN];
int a[MAXN],len,root,recy[MAXN],rlen;
int getch(int x){return sp[sp[x].fa].ch[1]==x;}
void pushup(int x){
 int lc=sp[x].ch[0],rc=sp[x].ch[1];
 sp[x].size=sp[lc].size+sp[rc].size+sp[x].cnt;
 sp[x].sum=sp[lc].sum+sp[rc].sum+sp[x].v;
 sp[x].lm=max(sp[lc].lm, sp[lc].sum+sp[x].v+sp[rc].lm);
 sp[x].rm=max(sp[rc].rm, sp[rc].sum+sp[x].v+sp[lc].rm);
 sp[x].mm=max(max(sp[lc].mm,sp[rc].mm),
  sp[lc].rm+sp[x].v+sp[rc].lm);
}
void pushdown(int x){
 int lc=sp[x].ch[0],rc=sp[x].ch[1];
 if(sp[x].color!=inf){
  if(lc){
```

```
    sp[lc].v=sp[lc].color=sp[x].color;
    sp[lc].sum=sp[lc].size*sp[x].color;
  }
  if(rc){
    sp[rc].v=sp[rc].color=sp[x].color;
    sp[rc].sum=sp[rc].size*sp[x].color;
  }
  if(sp[x].color>0){
    if(lc) sp[lc].lm=sp[lc].rm=sp[lc].mm=sp[lc].sum;
    if(rc) sp[rc].lm=sp[rc].rm=sp[rc].mm=sp[rc].sum;
  }
  else{
    if(lc) {sp[lc].lm=sp[lc].rm=0;
      ↪ sp[lc].mm=sp[lc].v;}
    if(rc) {sp[rc].lm=sp[rc].rm=0;
      ↪ sp[rc].mm=sp[rc].v;}
  }
  sp[x].color=inf;
  sp[x].flip=0;
 }
 else if(sp[x].flip){
  if(lc){
    sp[lc].flip^=1;
    swap(sp[lc].ch[0],sp[lc].ch[1]);
    swap(sp[lc].lm,sp[lc].rm);
  }
  if(rc){
    sp[rc].flip^=1;
    swap(sp[rc].ch[0],sp[rc].ch[1]);
    swap(sp[rc].lm,sp[rc].rm);
  }
  sp[x].flip=0;
 }
}
void rotate(int x){
 int f=sp[x].fa, ff=sp[f].fa;
 int k=getch(x);
 sp[ff].ch[getch(f)]=x; sp[x].fa=ff;
 sp[sp[x].ch[k^1]].fa=f; sp[f].ch[k]=sp[x].ch[k^1];
 sp[x].ch[k^1]=f; sp[f].fa=x;
 pushup(f); pushup(x);
}
void splay(int x,int goal=0){
 for(int f;(f=sp[x].fa)!=goal;rotate(x)){
  if(sp[f].fa!=goal)
   rotate(getch(x)==getch(f)?f:x);
 }
 if(!goal) root=x;
}
int find(int x){
 int cur=root;
 while(1){
  pushdown(cur);
  if(x<=sp[sp[cur].ch[0]].size)
   cur=sp[cur].ch[0];
  else if(x>sp[sp[cur].ch[0]].size+sp[cur].cnt){
   x-=sp[sp[cur].ch[0]].size+sp[cur].cnt;
   cur=sp[cur].ch[1];
  }
  else return cur;
 }
}
int built(int f,int l,int r){
 if(l>r) return 0;
 int mid=(l+r)>>1, cur=rlen?recy[rlen--]:++len;
 sp[cur].v=a[mid];
 sp[cur].fa=f;
 sp[cur].cnt=1;
 sp[cur].flip=0;
 sp[cur].color=inf;
 sp[cur].ch[0]=built(cur,l,mid-1);
 sp[cur].ch[1]=built(cur,mid+1,r);
 pushup(cur);
 return cur;
}
void insert(int pos,int tot){
 int l=find(pos),r=find(pos+1);
 splay(l);splay(r,l);
 sp[r].ch[0]=built(r,1,tot);
 pushup(r); pushup(l);
}
void recycle(int x){
 if(!x) return;
 recycle(sp[x].ch[0]);
 recycle(sp[x].ch[1]);
 sp[sp[x].fa].ch[getch(x)]=0;
 recy[++rlen]=x;
}
void erase(int pos,int tot){
 int l=find(pos-1),r=find(pos+tot);
 splay(l);splay(r,l);
 recycle(sp[r].ch[0]);
 pushup(r); pushup(l);
}
void dye(int pos,int tot,int c){
 int l=find(pos-1),r=find(pos+tot);
 splay(l);splay(r,l);
 int x=sp[r].ch[0];
 sp[x].color=c;
 sp[x].v=c;
 sp[x].sum=sp[x].size*c;
 if(c>0)
  sp[x].lm=sp[x].rm=sp[x].mm=sp[x].sum;
 else{
  sp[x].lm=sp[x].rm=0;
  sp[x].mm=sp[x].v;
 }
 pushup(r); pushup(l);
}
void reverse(int pos,int tot){
 int l=find(pos-1),r=find(pos+tot);
 splay(l);splay(r,l);
 int x=sp[r].ch[0];
 sp[x].flip^=1;
 swap(sp[x].ch[0],sp[x].ch[1]);
 swap(sp[x].lm,sp[x].rm);
 pushup(r); pushup(l);
}
int getsum(int pos,int tot){
 int l=find(pos-1),r=find(pos+tot);
 splay(l);splay(r,l);
 return sp[sp[r].ch[0]].sum;
}
int n,m;
int main(){
 scanf("%d %d", &n, &m);
 for(int i=1;i<=n;i++){
  scanf("%d", &a[i+1]);
 }
 memset(sp,0,sizeof(sp[0]));
 sp[0].mm=a[1]=a[n+2]=-inf;
 rlen=0;
 len=0;
 root=built(0,1,n+2);
 for(int i=1;i<=m;i++){
  char opt[10];
  scanf("%s", opt);
  if(opt[0]=='I'){//Insert
   int pos,tot;
   scanf("%d %d", &pos, &tot);
   for(int i=1;i<=tot;i++)
    scanf("%d", &a[i]);
   insert(pos+1,tot);
  }
  else if(opt[0]=='D'){//Delete
   int pos,tot;
   scanf("%d %d", &pos, &tot);
   erase(pos+1,tot);
  }
  else if(opt[2]=='K'){//Make-Same
   int pos,tot,c;
   scanf("%d %d %d", &pos, &tot, &c);
   dye(pos+1,tot,c);
  }
  else if(opt[0]=='R'){//Reverse
   int pos,tot;
   scanf("%d %d", &pos, &tot);
   reverse(pos+1,tot);
  }
  else if(opt[0]=='G'){//Get-Sum
   int pos,tot;
   scanf("%d %d", &pos, &tot);
   printf("%d\n", getsum(pos+1,tot));
  }
 }
```

```
  | | else if(opt[0]=='M'){//Max-Sum
  | | | printf("%d\n", sp[root].mm);
  | | }
  | }
  | return 0;
}
```

2.9 点分治

```
vector<pair<int, int> > G[maxn];
int sz[maxn], son[maxn], q[maxn];
int pr[maxn], depth[maxn], rt[maxn][19], d[maxn][19];
int cnt_all[maxn],sum_all[maxn],cnt[maxn][],sum[maxn][];
bool vis[maxn], col[maxn];
int getcenter(int o, int s) {
  | int head = 0, tail = 0; q[tail++] = o;
  | while (head != tail) {
  | | int x = q[head++]; sz[x] = 1; son[x] = 0;
  | | for (auto [y, _] : G[x]) if (!vis[y] && y != pr[x]) {
  | | | pr[y] = x; q[tail++] = y; } }
  | for (int i = tail - 1; i; i--) {
  | | int x = q[i]; sz[pr[x]] += sz[x];
  | | if (sz[x] > sz[son[pr[x]]]) son[pr[x]] = x; }
  | int x = q[0];
  | while (son[x] && sz[son[x]] * 2 >= s) x = son[x];
  | return x; }
void getdis(int o, int k) {
  | int head = 0, tail = 0; q[tail++] = o;
  | while (head != tail) {
  | | int x = q[head++]; sz[x] = 1; rt[x][k] = o;
  | | for (auto [y, w] : G[x]) if (!vis[y] && y != pr[x]) {
  | | | pr[y]=x; d[y][k] = d[x][k] + w; q[tail++]=y; } }
  | for (int i = tail - 1; i; i--)sz[pr[q[i]]] += sz[q[i]];}
void build(int o, int k, int s, int fa) {
  | int x = getcenter(o, s);
  | vis[x] = true; depth[x] = k; pr[x] = fa;
  | for (auto [y, w] : G[x]) if (!vis[y]) {
  | | d[y][k] = w; pr[y] = x; getdis(y, k); }
  | for (auto [y, w] : G[x]) if (!vis[y])
  | | build(y, k + 1, sz[y], x); }
void modify(int x) {
  | int t = col[x] ? -1 : 1; cnt_all[x] += t;
  | for (int u = pr[x], k = depth[x] - 1; u; u = pr[u],k--){
  | | sum_all[u] += t * d[x][k]; cnt_all[u] += t;
  | | sum[rt[x][k]][k] += t*d[x][k]; cnt[rt[x][k]][k] += t;
  | } col[x] ^= true; }
int query(int x) { int ans = sum_all[x];
  | for (int u = pr[x], k = depth[x] - 1; u; u = pr[u], k--)
  | | ans += sum_all[u] - sum[rt[x][k]][k]
  | | | + d[x][k] * (cnt_all[u] - cnt[rt[x][k]][k]);
  | return ans; }
```

容斥

```
int n, m;
int h[N], e[M], w[M], ne[M], idx;
bool ste[N];
int p[N], q[N];
void add(int a, int b, int c) {
  | e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
int get_size(int pre, int from) {
  | if (ste[pre]) return 0;
  | int res = 1;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i];
  | | if (to == from) continue;
  | | res += get_size(to, pre);
  | }
  | return res;
}
int get_wc(int pre, int from, int tot, int& wc) {
  | if (ste[pre]) return 0;
  | int sum = 1, ms = 0;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i];
  | | if (to == from) continue;
  | | int t = get_wc(to, pre, tot, wc);
  | | ms = max(ms, t);
  | | sum += t;
  | }
  | ms = max(ms, tot - sum);
  | if (ms <= tot / 2) wc = pre;
  | return sum;
}
void get_dis(int pre, int from, int dis, int& qt) {
  | if (ste[pre]) return;
  | q[qt++] = dis;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i];
  | | if (to == from) continue;
  | | get_dis(to, pre, dis + w[i], qt);
  | }
}
int get(int A[], int k) {
  | sort(A, A + k);
  | int res = 0;
  | for (int i = k - 1, j = -1; i >= 0; i--) {
  | | while (j + 1 < i && A[j + 1] + A[i] <= m) j++;
  | | j = min(j, i - 1);
  | | res += j + 1;
  | }
  | return res;
}
int calc(int pre) {
  | if (ste[pre]) return 0;
  | int res = 0;
  | get_wc(pre, -1, get_size(pre, -1), pre);
  | ste[pre] = 1;
  | int pt = 0;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i], qt = 0;
  | | get_dis(to, -1, w[i], qt);
  | | res -= get(q, qt);
  | | for (int k = 0; k < qt; k++) {
  | | | if (q[k] <= m) res++;
  | | | p[pt++] = q[k];
  | | }
  | }
  | res += get(p, pt);
  | for (int i = h[pre]; ~i; i = ne[i]) res += calc(e[i]);
  | return res;
}
void oper() {
  | while (cin >> n >> m, n || m) {
  | | for (int i = 0; i <= n; i++) h[i] = -1, ste[i] = 0;
  |   ↪ idx = 0;
  | | for (int i = 1; i < n; i++) {
  | | | int a, b, c; cin >> a >> b >> c;
  | | | add(a, b, c), add(b, a, c);
  | | }
  | | cout << calc(0) << endl;
  | }
}
```

哈希表

```
int n, m;
int h[N], e[M], w[M], ne[M], idx;
bool ste[N];
int f[S], ans;
pii p[N], q[N];
void add(int a, int b, int c) {
  | e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
int get_size(int pre, int from) {
  | if (ste[pre]) return 0;
  | int res = 1;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i];
  | | if (to == from) continue;
  | | res += get_size(to, pre);
  | }
  | return res;
}
int get_wc(int pre, int from, int tot, int& wc) {
  | if (ste[pre]) return 0;
  | int sum = 1, ms = 0;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  | | int to = e[i];
  | | if (to == from) continue;
  | | int t = get_wc(to, pre, tot, wc);
  | | ms = max(ms, t);
  | | sum += t;
```

```
  | }
  | ms = max(ms, tot - sum);
  | if (ms <= tot / 2) wc = pre;
  | return sum;
}
void get_dis(int pre, int from, int dis, int cnt, int& qt)
  ↪ {
  | if (ste[pre] || dis > m) return;
  | q[qt++] = { dis, cnt };
  | for (int i = h[pre]; ~i; i = ne[i]) {
  |  | int to = e[i];
  |  | if (to == from) continue;
  |  | get_dis(to, pre, dis + w[i], cnt + 1, qt);
  | }
}
void calc(int pre) {
  | if (ste[pre]) return;
  | get_wc(pre, -1, get_size(pre, -1), pre);
  | ste[pre] = 1;
  | int pt = 0;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  |  | int to = e[i], qt = 0;
  |  | get_dis(to, -1, w[i], 1, qt);
  |  | for (int k = 0; k < qt; k++) {
  |  |  | auto& t = q[k];
  |  |  | if (t.x == m) ans = min(ans, t.y);
  |  |  | ans = min(ans, f[m - t.x] + t.y);
  |  |  | p[pt++] = t;
  |  | }
  |  | for (int k = 0; k < qt; k++) {
  |  |  | auto& t = q[k];
  |  |  | f[t.x] = min(f[t.x], t.y);
  |  | }
  | }
  | for (int i = 0; i < pt; i++) f[p[i].x] = inf;
  | for (int i = h[pre]; ~i; i = ne[i]) calc(e[i]);
}
void oper() {
  | cin >> n >> m;
  | memset(f, 0x3f, sizeof f);
  | for (int i = 0; i <= n; i++) h[i] = -1, ste[i] = 0; idx
    ↪ = 0; ans = inf;
  | for (int i = 1; i < n; i++) {
  |  | int a, b, c; cin >> a >> b >> c;
  |  | add(a, b, c), add(b, a, c);
  | }
  | calc(0);
  | cout << (ans == inf ? -1 : ans) << endl;
}
```

2.10 可持久化字典树

```
int n, m;
int s[N], ne[M][2];
int rt[N], idx, maxid[M];
void insert(int id, int from, int pre) {
  | maxid[pre] = id;
  | for (int i = 23; i >= 0; i--) {
  |  | int to = s[id] >> i & 1;
  |  | if (from) ne[pre][to ^ 1] = ne[from][to ^ 1];
  |  | ne[pre][to] = ++idx;
  |  | from = ne[from][to], pre = ne[pre][to];
  |  | maxid[pre] = id;
  | }
}
int getans(int l, int r, int x) {
  | int pre = rt[r], ans = 0;
  | for (int i = 23; i >= 0; i--) {
  |  | int to = x >> i & 1;
  |  | if (ne[pre][to ^ 1] && maxid[ne[pre][to ^ 1]] >= l)
    ↪ pre = ne[pre][to ^ 1], ans += 1 << i;
  |  | else pre = ne[pre][to];
  | }
  | return ans;
}
void oper() {
  | cin >> n >> m;
  | rt[0] = ++idx; insert(0, 0, rt[0]);
  | for (int i = 1; i <= n; i++) {
  |  | int a; cin >> a;
  |  | s[i] = s[i - 1] ^ a;
  |  | rt[i] = ++idx; insert(i, rt[i - 1], rt[i]);
  | }
  | for (int i = 1; i <= m; i++) {
  |  | char ch; cin >> ch;
  |  | if (ch == 'A') {
  |  |  | int x; cin >> x;
  |  |  | n++;
  |  |  | s[n] = s[n - 1] ^ x;
  |  |  | rt[n] = ++idx; insert(n, rt[n - 1], rt[n]);
  |  | }
  |  | else {
  |  |  | int l, r, x; cin >> l >> r >> x;
  |  |  | cout << getans(l - 1, r - 1, s[n] ^ x) << endl;
  |  | }
  | }
}
```

2.11 可持久化线段树

```
int n, q;
int A[N], rt[N], idx;
struct type {
  | int l, r, cnt;
}tree[N * 4 + N * 17];
int build(int l, int r) {
  | int pre = ++idx; tree[pre] = { 0, 0, 0 };
  | if (l == r) return pre;
  | int mid = (l + r) / 2;
  | tree[pre].l = build(l, mid), tree[pre].r = build(mid +
    ↪ 1, r);
  | return pre;
}
int insert(int from, int l, int r, int x) {
  | int pre = ++idx;
  | tree[pre] = tree[from];
  | if (l == r) {
  |  | tree[pre].cnt++;
  |  | return pre;
  | }
  | int mid = (l + r) / 2;
  | if (x <= mid) tree[pre].l = insert(tree[from].l, l, mid,
    ↪ x);
  | else tree[pre].r = insert(tree[from].r, mid + 1, r, x);
  | tree[pre].cnt = tree[tree[pre].l].cnt +
    ↪ tree[tree[pre].r].cnt;
  | return pre;
}
int getans(int from, int pre, int l, int r, int x) {
  | if (l == r) return l;
  | int mid = (l + r) / 2, cnt = tree[tree[pre].l].cnt -
    ↪ tree[tree[from].l].cnt;
  | if (x <= cnt) return getans(tree[from].l, tree[pre].l,
    ↪ l, mid, x);
  | else return getans(tree[from].r, tree[pre].r, mid + 1,
    ↪ r, x - cnt);
}
void oper() {
  | cin >> n >> q;
  | set<int> s;
  | map<int, int> mp; int cnt = 0;
  | vector<int> v; v.push_back(0);
  | for (int i = 1; i <= n; i++) cin >> A[i],
    ↪ s.insert(A[i]);
  | for (auto& it : s) mp[it] = ++cnt, v.push_back(it);
  | rt[0] = build(1, cnt);
  | for (int i = 1; i <= n; i++) rt[i] = insert(rt[i - 1],
    ↪ 1, cnt, mp[A[i]]);
  | for (int i = 1; i <= q; i++) {
  |  | int l, r, x; cin >> l >> r >> x;
  |  | cout << v[getans(rt[l - 1], rt[r], 1, cnt, x)] <<
      ↪ endl;
  | }
}
```

2.12 支配树

给定一张 n 点 m 边的有向图，问以 1 为起点，dfs和bfs得到的最短路是否一定相同。

最短路径树+支配树板子。

```
int n, m;
int d[N], ans[N], dep[N], dp[N][21];
vector<int> v[N], son[N];
namespace dtree {
```

```
    vector<int> E[N], RE[N], rdom[N];
    int S[N], RS[N], cs;
    int F[N], val[N], sdom[N], rp[N], dom[N];
    void clear(int n) {
        cs = 0;
        for (int i = 0; i <= n; i++) {
            F[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
            E[i].clear(), RE[i].clear(), rdom[i].clear();
        }
    }
    void add(int a, int b) { E[a].push_back(b); }
    int Find(int x, int c = 0) {
        if (F[x] == x) return c ? -1 : x;
        int p = Find(F[x], 1);
        if (p == -1) return c ? F[x] : val[x];
        if (sdom[val[x]] > sdom[val[F[x]]]) val[x] = val[F[x]];
        F[x] = p;
        return c ? p : val[x];
    }
    void dfs(int x) {
        RS[S[x] = ++cs] = x;
        F[cs] = sdom[cs] = val[cs] = cs;
        for (auto& to : E[x]) {
            if (S[to] == 0) dfs(to), rp[S[to]] = S[x];
            RE[S[to]].push_back(S[x]);
        }
    }
    int solve(int s, int* up) {
        dfs(s);
        for (int i = cs; i; i--) {
            for (int to : RE[i]) sdom[i] = min(sdom[i], sdom[Find(to)]);
            if (i > 1) rdom[sdom[i]].push_back(i);
            for (auto& to : rdom[i]) {
                int p = Find(to);
                if (sdom[p] == i) dom[to] = i;
                else dom[to] = p;
            }
            if (i >= 1) F[i] = rp[i];
        }
        for (int i = 2; i <= cs; i++) if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
        for (int i = 2; i <= cs; i++) up[RS[i]] = RS[dom[i]];
        return cs;
    }
}
void dfs(int pre, int depth) {
    dep[pre] = depth;
    for (auto& to : son[pre]) dfs(to, depth + 1);
}
int LCA(int a, int b) {
    if (dep[a] < dep[b]) swap(a, b);
    for (int i = 20; i >= 0; i--) if (dep[dp[a][i]] >= dep[b]) a = dp[a][i];
    if (a == b) return a;
    for (int i = 20; i >= 0; i--) if (dp[a][i] != dp[b][i]) a = dp[a][i], b = dp[b][i];
    return dp[a][0];
}
void oper() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) v[i].clear(), son[i].clear(), d[i] = 1e18; d[1] = 0;
    dtree::clear(n);
    for (int i = 1; i <= m; i++) {
        int a, b; cin >> a >> b;
        v[a].push_back(b);
        dtree::add(a, b);
    }
    dtree::solve(1, ans);
    for (int i = 2; i <= n; i++) son[ans[i]].push_back(i);
    dfs(1, 1);
    for (int i = 1; i <= n; i++) dp[i][0] = ans[i];
    for (int i = 1; i <= 20; i++) for (int pre = 1; pre <= n; pre++) dp[pre][i] = dp[dp[pre][i - 1]][i - 1];
    queue<int> q; q.push(1);
    while (q.size()) {
        int pre = q.front(); q.pop();
        for (auto& to : v[pre]) {
            if (d[to] > d[pre] + 1) {
                d[to] = d[pre] + 1;
                q.push(to);
            }
        }
    }
    for (int pre = 1; pre <= n; pre++) {
        for (auto& to : v[pre]) {
            if (d[to] != d[pre] + 1 && LCA(pre, to) != to) {
                cout << "No" << endl;
                return;
            }
        }
    }
    cout << "Yes" << endl;
}
```

3. Tree & Graph

3.1 树的重心 $O(n)$

```
// 以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
// 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
// 把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
// 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。
struct edge { int to,next; }e[MAXN<<1];
int tot,head[MAXN];
void add(int x,int y) {
    tot++;
    e[tot].to=y; e[tot].next=head[x]; head[x]=tot;
}
int n, size[MAXN],weight[MAXN];
vector<int> centroid;
void get_centroid(int x,int f) {
    size[x]=1; weight[x]=0;
    for(int p=head[x];p;p=e[p].next) {
        int u=e[p].to;
        if(u==f) continue;
        get_centroid(u,x);
        size[x]+=size[u];
        weight[x]=max(weight[x],size[u]);
    }
    weight[x]=max(weight[x],n-size[x]);
    if(weight[x]*2<=n) centroid.push_back(x);
}
void solve() {
    scanf("%d", &n); tot=0;
    memset(head+1,0,n*sizeof(head[0]));
    for(int i=1;i<n;i++) {
        int f,g; scanf("%d %d", &f, &g);
        add(f,g); add(g,f);
    }
    centroid.clear(); get_centroid(1,1);
    sort(centroid.begin(), centroid.end());
    printf("%d %d\n", centroid[0], weight[centroid[0]]);
}
```

3.2 LCA

3.2.1 Tarjan

```
struct edge{ int to,next;}e[MAXM<<1],eq[MAXM<<1];
int n,m,s,tot,totq;
int head[MAXN],headq[MAXN],fa[MAXN],ans[MAXM];
void add(int x, int y){
    tot++;
    e[tot].to=y; e[tot].next=head[x]; head[x]=tot;
}
void addq(int x, int y){
    tot++;
    eq[tot].to=y; eq[tot].next=headq[x]; headq[x]=tot;
}
int find(int x){
    if(fa[x]==x) return x;
    return fa[x]=find(fa[x]);
}
void tarjan(int x){
    fa[x]=x;
    for(int p=head[x];p;p=e[p].next){
        int u=e[p].to;
        if(!fa[u]){
```

```
    |   |   | tarjan(u);
    |   |   | fa[u]=x;
    |   | }
    | }
    | for(int p=headq[x];p;p=eq[p].next){
    |   | int u=eq[p].to;
    |   | if(fa[u]){
    |   |   | ans[(p+1)>>1]=find(u);
    |   | }
    | }
}
int main(){
    | scanf("%d %d %d", &n, &m, &s);
    | for(int i=1;i<=n;i++){
    |   | head[i]=0; headq[i]=0;
    | }
    | tot=0;
    | for(int i=1;i<n;i++){
    |   | int f,g;
    |   | scanf("%d %d",&f,&g);
    |   | add(f,g); add(g,f);
    | }
    | tot=0;
    | for(int i=1;i<=m;i++){
    |   | int f,g;
    |   | scanf("%d %d",&f,&g);
    |   | addq(f,g); addq(g,f);
    | }
    | for(int i=1;i<=n;i++) fa[i]=0;
    | tarjan(s);
    | for(int i=1;i<=m;i++)
    |   | printf("%d\n", ans[i]);
    | return 0;
}
```

3.2.2 重链剖分

```
/*
  | LCA重链剖分
  | luogu3379
*/
struct edge {
  | int to,next; |
}e[MAXN<<1];
int tot,head[MAXN];
void add(int x,int y) {
  | tot++;
  | e[tot].to=y;
  | e[tot].next=head[x];
  | head[x]=tot;
}
int fa[MAXN],dep[MAXN],siz[MAXN],son[MAXN];
int top[MAXN],dfn[MAXN],rnk[MAXN];
void dfs1(int x,int f) {
  | fa[x]=f;
  | dep[x]=dep[f]+1;
  | siz[x]=1;
  | son[x]=0;
  | for(int p=head[x];p;p=e[p].next) {
  |   | int u=e[p].to;
  |   | if(u==f) continue;
  |   | dfs1(u,x);
  |   | siz[x]+=siz[u];
  |   | if(!son[x] || siz[u]>siz[son[x]])
  |   |   | son[x]=u;
  | }
}
int cnt;
void dfs2(int x,int f) {
  | top[x]=f;
  | dfn[x]=++cnt;
  | rnk[cnt]=x;
  | if(!son[x]) return;
  | dfs2(son[x],f);
  | for(int p=head[x];p;p=e[p].next) {
  |   | int u=e[p].to;
  |   | if(u==fa[x] || u==son[x]) continue;
  |   | dfs2(u,u);
  | }
}
int lca(int x,int y) {
  | while(top[x]!=top[y]) {
  |   | if(dep[top[x]]>dep[top[y]])
  |   |   | x=fa[top[x]];
  |   | else y=fa[top[y]];
  | }
  | if(dep[x]>dep[y]) return y;
  | else return x;
}
void solve() {
  | int n,m,s;
  | scanf("%d %d %d", &n, &m, &s);
  | tot=0;
  | memset(head+1,0,n*sizeof(head[0]));
  | for(int i=1;i<n;i++) {
  |   | int f,g;
  |   | scanf("%d %d", &f, &g);
  |   | add(f,g);
  |   | add(g,f);
  | }
  | dep[0]=0;
  | dfs1(s,0);
  | cnt=0;
  | dfs2(s,s);
  | for(int i=1;i<=m;i++) {
  |   | int f,g;
  |   | scanf("%d %d", &f, &g);
  |   | printf("%d\n", lca(f,g));
  | }
}
```

3.2.3 Doubling

```
int n, m, rt;
int h[N], e[M], ne[M], idx;
int dep[N], dp[N][16];
void add(int a, int b) {
  | e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void bfs() {
  | memset(dep, 0x3f, sizeof dep);
  | queue<int> q; q.push(rt);
  | dep[rt] = 1; dep[0] = 0;
  | while (q.size()) {
  |   | int ver = q.front(); q.pop();
  |   | for (int i = h[ver]; ~i; i = ne[i]) {
  |   |   | int to = e[i];
  |   |   | if (dep[to] > dep[ver] + 1) {
  |   |   |   | dep[to] = dep[ver] + 1;
  |   |   |   | q.push(to); dp[to][0] = ver;
  |   |   |   | for (int i = 1; i <= 15; i++)
  |   |   |   |   | dp[to][i] = dp[dp[to][i - 1]][i - 1];
  |   |   | }
  |   | }
  | }
}
int LCA(int a, int b) {
  | if (dep[a] < dep[b]) swap(a, b);
  | for (int i = 15; i >= 0; i--) {
  |   | if (dep[dp[a][i]] >= dep[b]) a = dp[a][i];
  | }
  | if (a == b) return b;
  | for (int i = 15; i >= 0; i--) {
  |   | if (dp[a][i] != dp[b][i]) {
  |   |   | a = dp[a][i];
  |   |   | b = dp[b][i];
  |   | }
  | }
  | return dp[a][0];
}
```

3.3 拓扑排序

3.3.1 $O(n+m)$

```
bool topsort() {
  | int hh = 0, tt = -1;
  | // d[i] 存储点i的入度
  | for (int i = 1; i <= n; i++)
  |   | if (!d[i]) q[++tt] = i;
  | while (hh <= tt) {
  |   | int t = q[hh++];
  |   | for (int i = h[t]; i != -1; i = ne[i]) {
  |   |   | int j = e[i];
  |   |   | if (--d[j] == 0) q[++tt] = j;
  |   | }
```

```
    }
    // 如果所有点都入队了，说明存在拓扑序列；否则不存在拓扑序列。
    return tt == n - 1;
}
```

3.4 Dijkstra

3.4.1 $O(n^2+m)$

```
int g[N][N];  // 存储每条边
int dist[N];  // 存储1号点到每个点的最短距离
bool st[N];   // 存储每个点的最短路是否已经确定
// 求1号点到n号点的最短路，如果不存在则返回-1
int dijkstra() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    for (int i = 0; i < n - 1; i++) {
        int t = -1;  // 在还未确定最短路的点中，寻找距离最小的点
        for (int j = 1; j <= n; j++)
            if (!st[j] && (t == -1 || dist[t] > dist[j])) t = j;
        // 用t更新其他点的距离
        for (int j = 1; j <= n; j++) dist[j] = min(dist[j], dist[t] + g[t][j]);
        st[t] = true;
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

3.4.2 $O(mlogn)$

```
typedef pair<int, int> PII;
int n;                                // 点的数量
int h[N], w[N], e[N], ne[N], idx; // 邻接表存储所有边
int dist[N];                          // 存储所有点到1号点的距离
bool st[N];  // 存储每个点的最短距离是否已确定
// 求1号点到n号点的最短距离，如果不存在，则返回-1
int dijkstra() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1});  // first存储距离，second存储节点编号
    while (heap.size()) {
        auto t = heap.top();
        heap.pop();
        int ver = t.second, distance = t.first;
        if (st[ver]) continue;
        st[ver] = true;
        for (int i = h[ver]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > distance + w[i]) {
                dist[j] = distance + w[i];
                heap.push({dist[j], j});
            }
        }
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

3.5 Bellman-Ford

3.5.1 $O(nm)$

```
int n, m;       // n表示点数，m表示边数
int dist[N];    // dist[x] 存储1到x的最短路距离
struct Edge {  // 边，a表示出点，b表示入点，w表示边的权重
    int a, b, w;
} edges[M];
// 求1到n的最短路距离，如果无法从1走到n，则返回-1。
int bellman_ford() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    // 如果第n次迭代仍然会松弛三角不等式，就说明存在一条长度是n+1的最短路径，由抽屉原理，路径中至少存在两个相同的点，说明图中存在负权回路。
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int a = edges[j].a, b = edges[j].b, w = edges[j].w;
            if (dist[b] > dist[a] + w) dist[b] = dist[a] + w;
        }
    }
    if (dist[n] > 0x3f3f3f3f / 2) return -1;
    return dist[n];
}
```

3.5.2 $O(m)$ to $O(nm)$

```
int n;                                // 总点数
int h[N], w[N], e[N], ne[N], idx;  // 邻接表存储所有边
int dist[N];                          // 存储每个点到1号点的最短距离
bool st[N];                           // 存储每个点是否在队列中
// 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1
int spfa() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    queue<int> q;
    q.push(1);
    st[1] = true;
    while (q.size()) {
        auto t = q.front();
        q.pop();
        st[t] = false;
        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                if (!st[j]) {  // 如果队列中已存在j，则不需要将j重复插入
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}
```

3.5.3 判断是否存在负环

```
int n, m, h[N], e[M], w[M], ne[M], idx;
int cnt[N], p[N];
double d[N];
bool ste[N];
void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
}
bool check(double x) {
    memset(d, 0x3f, sizeof d);
    memset(cnt, 0, sizeof cnt);
    memset(ste, 0, sizeof ste);
    queue<int> q;
    for (int i = 1; i <= n; i++) q.push(i), ste[i] = 1;
    while (q.size()) {
        int ver = q.front();
        q.pop();
        ste[ver] = 0;
        for (int i = h[ver]; ~i; i = ne[i]) {
            int to = e[i];
            if (d[to] > d[ver] + x * w[i] - p[ver]) {
                d[to] = d[ver] + x * w[i] - p[ver];
                cnt[to] = cnt[ver] + 1;
                if (cnt[to] >= n) return 1;
                if (!ste[to]) q.push(to), ste[to] = 1;
            }
        }
    }
    return 0;
}
void oper() {
    cin >> n >> m;
    memset(h, -1, sizeof h);
    idx = 0;
    for (int i = 1; i <= n; i++) cin >> p[i];
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    double l = 0, r = 1000, mid;
    while (r - l > eps) {
```

```
    mid = (l + r) / 2;
    if (check(mid))
      l = mid;
    else
      r = mid;
  }
  cout << fixed << setprecision(2) << l << endl;
}
```

3.6 floyd $O(n^3)$

```
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
```

3.7 Hungarian $O(VE)$

```
int n, k, mat[N], vis[N]; vector<int> E[N];
bool dfs(int tim, int x) {
  for (auto y : E[x]) {
    if (vis[y] == tim) continue;
    vis[y] = tim;
    if (!mat[y] || dfs(tim, mat[y])) {
      mat[y] = x; return 1; } }
  return 0; }
int solve() {
  fill(vis + 1, vis + k + 1, 0);
  int ans = 0;
  for (int i = 1; i <= n; ++i)
    if (dfs(i, i)) ans++;
  return ans; }
```

3.8 Shuffle 一般图最大匹配 $O(VE)$

```
mt19937 rng(233);
int n, m, mat[N], vis[N]; vector<int> E[N];
bool dfs(int tim, int x) {
  shuffle(E[x].begin(), E[x].end(), rng);
  vis[x] = tim;
  for (auto y : E[x]) {
    int z = mat[y]; if (vis[z] == tim) continue;
    mat[x] = y, mat[y] = x, mat[z] = 0;
    if (!z || dfs(tim, z)) return true;
    mat[x] = 0, mat[y] = z, mat[z] = y; }
  return false; }
int main() {
  for (int T = 0; T < 10; ++T) {
    fill(vis + 1, vis + n + 1, 0);
    for (int i = 1; i <= n; ++i)
      if (!mat[i]) dfs(i, i); } }
```

3.9 带花树 一般图最大匹配

```
#define int long long
const int N = 1e3 + 10;
int n, m, ans;
int tot, T[N], vis[N], ma[N], V[N], F[N];
vector<int> v[N];
queue<int> q;
int Find(int x) {
  if (F[x] == x) return x;
  return F[x] = Find(F[x]);
}
int LCA(int x, int y) {
  tot++;
  while (1) {
    if (x) {
      x = Find(x);
      if (T[x] == tot) return x;
      T[x] = tot, x = V[ma[x]];
    }
    swap(x, y);
  }
}
void flower(int x, int y, int p) {
  while (Find(x) != p) {
    V[x] = y, y = ma[x], vis[y] = 1, q.push(y);
    if (Find(x) == x) F[x] = p;
    if (Find(y) == y) F[y] = p;
    x = V[y];
  }
}
void bfs(int st) {
  for (int i = 1; i <= n; i++) V[i] = vis[i] = 0, F[i] = i;
  while (q.size()) q.pop();
  vis[st] = 1, q.push(st);
  while (q.size()) {
    int x = q.front();
    q.pop();
    for (auto& y : v[x])
      if (Find(y) != Find(x) && vis[y] != 2) {
        if (vis[y] == 1) {
          int l = LCA(x, y);
          flower(x, y, l);
          flower(y, x, l);
          continue;
        }
        vis[y] = 2, V[y] = x;
        if (!ma[y]) {
          int px = y;
          while (px) {
            int py = V[px], pz = ma[py];
            ma[px] = py, ma[py] = px, px = pz;
          }
          ans++;
          return;
        }
        vis[ma[y]] = 1, q.push(ma[y]);
      }
  }
}
void oper() {
  cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int a, b;
    cin >> a >> b;
    v[a].push_back(b);
    v[b].push_back(a);
  }
  for (int i = 1; i <= n; i++)
    if (!ma[i]) bfs(i);
  cout << ans << endl;
  for (int i = 1; i <= n; i++) cout << ma[i] << " \n"[i == n];
}
```

3.10 KM 最大权匹配 $O(V^3)$

```
struct KM {
int n, nl, nr;
LL a[N][N];
LL hl[N], hr[N], slk[N];
int fl[N], fr[N], vl[N], vr[N], pre[N], q[N], ql, qr;
int check(int i) {
  if (vl[i] = 1, fl[i] != -1)
    return vr[q[qr++] = fl[i]] = 1;
  while (i != -1) swap(i, fr[fl[i] = pre[i]]);
  return 0; }
void bfs(int s) {
  fill(slk, slk + n, INF);
  fill(vl, vl + n, 0); fill(vr, vr + n, 0);
  q[ql = 0] = s; vr[s] = qr = 1;
  for (LL d;;) {
    for (; ql < qr; ++ql)
    for (int i = 0, j = q[ql]; i < n; ++i)
    if (d=hl[i]+hr[j]-a[i][j], !vl[i] && slk[i] >= d) {
      if (pre[i] = j, d) slk[i] = d;
      else if (!check(i)) return; }
    d = INF;
    for (int i = 0; i < n; ++i)
    if (!vl[i] && d > slk[i]) d = slk[i];
    for (int i = 0; i < n; ++i) {
      if (vl[i]) hl[i] += d; else slk[i] -= d;
      if (vr[i]) hr[i] -= d; }
    for (int i = 0; i < n; ++i)
      if (!vl[i] && !slk[i] && !check(i)) return; } }
void solve() {
  n = max(nl, nr);
  fill(pre, pre + n, -1); fill(hr, hr + n, 0);
  fill(fl, fl + n, -1); fill(fr, fr + n, -1);
  for (int i = 0; i < n; ++i)
    hl[i] = *max_element(a[i], a[i] + n);
```

```
  | for (int i = 0; i < n; ++i)
  |  | bfs(i); }
LL calc() {
  | LL ans = 0;
  | for (int i = 0; i < nl; ++i)
  |  | if (~fl[i]) ans += a[i][fl[i]];
  | return ans; }
void output() {
  | for (int i = 0; i < nl; ++i)
  | printf("%d ", (~fl[i] && a[i][fl[i]] ? fl[i] + 1 : 0));
} } km;
```

3.11 欧拉回路

```
/* comment : directed */
int e, cur[N]/*, deg[N]*/;
vector<int>E[N];
int id[M]; bool vis[M];
stack<int>stk;
void dfs(int u) {
  | for (cur[u]; cur[u] < E[u].size(); cur[u]++) {
  |  | int i = cur[u];
  |  | if (vis[abs(E[u][i])]) continue;
  |  | int v = id[abs(E[u][i])] ^ u;
  |  | vis[abs(E[u][i])] = 1; dfs(v);
  |  | stk.push(E[u][i]); }
}// dfs for all when disconnect
void add(int u, int v) {
  | id[++e] = u ^ v; // s = u
  | E[u].push_back(e); E[v].push_back(-e);
/*  | E[u].push_back(e); deg[v]++; */
} bool valid() {
  | for (int i = 1; i <= n; i++)
  |  | if (E[i].size() & 1) return 0;
/* |  | if (E[i].size() != deg[i]) return 0;*/
  | return 1;}
```

3.12 kosaraju强连通分量

```
int main() {
  | int n, m; cin >> n >> m;
  | vector<int> c(n); for (int &i : c) { cin >> i; }
  | vector<vector<pair<int, int>>> adj(n), radj(n);
  | for (int i = 0; i < m; ++i) {
  |  | int u, v, w; cin >> u >> v >> w; --u, --v;
  |  | adj[u].push_back({v, w}); radj[v].push_back({u, w});
  | }
  | vector<bool> vis(n); vector<int> p;
  | function<void(int)> dfs1 = [&] (int u) {
  |  | vis[u] = true;
  |  | for (auto V : adj[u]) {
  |  |  | int v = V.first; if (!vis[v]) { dfs1(v); }
  |  | } p.push_back(u);
  | };
  | for (int i = 0; i < n; ++i) { if (!vis[i]) { dfs1(i); }
    ↪ }
  | vector<int> sccidx(n, -1), sccc(n);
  | function<void(int, int)> dfs2 = [&] (int u, int idx) {
  |  | sccidx[u] = idx; sccc[idx] += c[u];
  |  | for (auto V : radj[u]) {
  |  |  | int v = V.first, w = V.second;
  |  |  | if (sccidx[v] == -1) { dfs2(v, idx); sccc[idx] +=
    ↪ w;
  |  |  | } else if (sccidx[v] == idx) { sccc[idx] += w; }
  |  | }
  | };
  | int sccn = 0; reverse(p.begin(), p.end());
  | for (int u : p) {
  |  | if (sccidx[u] == -1) { dfs2(u, sccn); sccn += 1; }
  | }
  | vector<vector<pair<int, int>>> sccadj(sccn);
  | vector<int> sccd(sccn);
  | for (int u = 0; u < n; ++u) {
  |  | for (auto V : adj[u]) {
  |  |  | int v = V.first, w = V.second;
  |  |  | int sccu = sccidx[u], sccv = sccidx[v];
  |  |  | if (sccu != sccv) { sccadj[sccu].push_back({sccv,
    ↪ w}); sccd[sccv] += 1;}
  |  | }
  | }
  | vector<int> dp(sccn); queue<int> topo;
  | for (int i = 0; i < sccn; ++i) {
  |  | if (sccd[i] == 0) { dp[i] = sccc[i]; topo.push(i);}
  | }
  | while (!topo.empty()) {
  |  | int u = topo.front(); topo.pop();
  |  | for (auto V : sccadj[u]) {
  |  |  | int v = V.first, w = V.second;
  |  |  | dp[v] = max(dp[v], dp[u] + w + sccc[v]);
  |  |  | if (--sccd[v] == 0) { topo.push(v);}
  |  | }
  | }
  | cout << *max_element(dp.begin(), dp.end()) << endl;
}
```

3.13 Tarjan 强连通分量

```
int n, m;
int h[N], e[M], ne[M], idx;
int tt[N], low[N], timestamp;
stack<int> s;
bool ste[N];
int id[N], scc_cnt, sz[N];
int dout[N];
void add(int a, int b) {
  | e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void tarjan(int pre) {
  | tt[pre] = low[pre] = ++timestamp;
  | s.push(pre), ste[pre] = 1;
  | for (int i = h[pre]; ~i; i = ne[i]) {
  |  | int to = e[i];
  |  | if (!tt[to]) {
  |  |  | tarjan(to);
  |  |  | low[pre] = min(low[pre], low[to]);
  |  | }
  |  | else if (ste[to]) low[pre] = min(low[pre], tt[to]);
  | }
  | if (tt[pre] == low[pre]) {
  |  | int ver;
  |  | scc_cnt++;
  |  | do {
  |  |  | ver = s.top(); s.pop();
  |  |  | ste[ver] = 0;
  |  |  | id[ver] = scc_cnt;
  |  |  | sz[scc_cnt]++;
  |  | } while (ver != pre);
  | }
}
void oper() {
  | cin >> n >> m;
  | memset(h, -1, sizeof h); idx = 0;
  | for (int i = 1; i <= m; i++) {
  |  | int a, b; cin >> a >> b;
  |  | add(a, b);
  | }
  | for (int i = 1; i <= n; i++) {
  |  | if (!tt[i]) tarjan(i);
  | }
  | for (int ver = 1; ver <= n; ver++) {
  |  | for (int i = h[ver]; ~i; i = ne[i]) {
  |  |  | int to = e[i];
  |  |  | int a = id[ver], b = id[to];
  |  |  | if (a != b) dout[a]++;
  |  | }
  | }
  | int zero = 0, ans = 0;
  | for (int i = 1; i <= scc_cnt; i++) {
  |  | if (!dout[i]) ans += sz[i], zero++;
  |  | if (zero >= 2) {
  |  |  | cout << "0" << endl;
  |  |  | return;
  |  | }
  | }
  | cout << ans << endl;
}
```

3.14 Tarjan 点双，边双

```
/** 边双 **/
int n, m, head[N], nxt[M << 1], to[M << 1], ed;
int dfn[N], low[N], bcc_id[N], bcc_cnt, stp;
bool bri[M << 1], vis[N];
vector<int> bcc[N];
void tar(int now, int fa) {
```

```
  | dfn[now] = low[now] = ++stp;
  | for (int i = head[now]; ~i; i = nxt[i]) {
  |   | if (!dfn[to[i]]) {
  |   |   | tar(to[i], now);
  |   |   | low[now] = min(low[now], low[to[i]]);
  |   |   | if (low[to[i]] > dfn[now])
  |   |   |   | bri[i] = bri[i ^ 1] = 1; }
  |   | else if (dfn[to[i]] < dfn[now] && to[i] != fa)
  |   |   | low[now] = min(low[now], dfn[to[i]]); } }
void DFS(int now) {
  | vis[now] = 1;
  | bcc_id[now] = bcc_cnt;
  | bcc[bcc_cnt].push_back(now);
  | for (int i = head[now]; ~i; i = nxt[i]) {
  |   | if (bri[i]) continue;
  |   | if (!vis[to[i]]) DFS(to[i]); } }
void EBCC() {// clear dfn low bri bcc_id vis
  | bcc_cnt = stp = 0;
  | for (int i = 1; i <= n; ++i) if (!dfn[i]) tar(i, 0);
  | for (int i = 1; i <= n; ++i)
  |   | if (!vis[i]) ++bcc_cnt, DFS(i); }
/** 点双 **/
vector<int> G[N],bcc[N];
int dfn[N], low[N], bcc_id[N], bcc_cnt, stp;
bool iscut[N]; pii stk[N]; int top;
void tar(int now, int fa) {
  | int child = 0;
  | dfn[now] = low[now] = ++stp;
  | for (int to: G[now]) {
  |   | if (!dfn[to]) {
  |   |   | stk[++top] = mkpair(now, to); ++child;
  |   |   | tar(to, now);
  |   |   | low[now] = min(low[now], low[to]);
  |   |   | if (low[to] >= dfn[now]) {
  |   |   |   | iscut[now] = 1;
  |   |   |   | bcc[++bcc_cnt].clear();
  |   |   |   | while (1) {
  |   |   |   |   | pii tmp = stk[top--];
  |   |   |   |   | if (bcc_id[tmp.first] != bcc_cnt) {
  |   |   |   |   |   | bcc[bcc_cnt].push_back(tmp.first);
  |   |   |   |   |   | bcc_id[tmp.first] = bcc_cnt; }
  |   |   |   |   | if (bcc_id[tmp.second] != bcc_cnt) {
  |   |   |   |   |   | bcc[bcc_cnt].push_back(tmp.second);
  |   |   |   |   |   | bcc_id[tmp.second] = bcc_cnt; }
  |   |   |   |   | if (tmp.first == now && tmp.second == to)
  |   |   |   |   |   | break; } } }
  |   | else if (dfn[to] < dfn[now] && to != fa) {
  |   |   | stk[++top] = mkpair(now, to);
  |   |   | low[now] = min(low[now], dfn[to]); } }
  | if (!fa && child == 1) iscut[now] = 0; }
void PBCC() { // clear dfn low iscut bcc_id
  | stp = bcc_cnt = top = 0;
  | for (int i = 1; i <= n; ++i) if (!dfn[i]) tar(i, 0); }
```

3.15 弦图

弦图的定义 连接环中不相邻的两个点的边称为弦. 一个无向图称为弦图, 当图中任意长度都大于 3 的环都至少有一个弦.

单纯点 一个点称为单纯点当 $\{v\} \cup A(v)$ 的导出子图为一个团. 任何一个弦图都至少有一个单纯点, 不是完全图的弦图至少有两个不相邻的单纯点.

完美消除序列 一个序列 $v_1, v_2, ..., v_n$ 满足 v_i 在 $v_i, \cdots, v_n$ 的诱导子图中为一个单纯点. 一个无向图是弦图当且仅当它有一个完美消除序列.

最大势算法 从 n 到 1 的顺序依次给点标号. 设 label_i 表示第 i 个点与多少个已标号的点相邻, 每次选择 label 最大的未标号的点进行标号. 用桶维护优先队列可以做到 $O(n+m)$.

弦图的判定 判定最大势算法输出是否合法即可. 如果依次判断是否构成团, 时间复杂度为 $O(nm)$. 考虑优化, 设 $v_{i+1}, \cdots, v_n$ 中所有与 v_i 相邻的点依次为 $N(v_i) = \{v_{j1}, \cdots, v_{jk}\}$. 只需判断 v_{j1} 是否与 $v_{j2}, \cdots, v_{jk}$ 相邻即可. 时间复杂度 $O(n+m)$.

弦图的染色 完美消除序列从后往前染色, 染上出度的 mex.

最大独立集 完美消除序列从前往后能选就选.

团数 最大团的点数. 一般图团数 $\leq$ 色数, 弦图团数 = 色数.

极大团 弦图的极大团一定为 $\{x\} \cup N(x)$.

最小团覆盖 用最少的团覆盖所有的点. 设最大独立集为 $\{p_1, \ldots, p_t\}$, 则 $\{p_1 \cup N(p_1), \ldots, p_t \cup N(p_t)\}$ 为最小团覆盖.

弦图 k 染色计数 $\prod_{v \in V} k - N(v) + 1$.

区间图 每个顶点代表一个区间, 有边当且仅当区间有交. 区间图是弦图, 一个完美消除序列是右端点排序.

```
vector <int> L[N];
int seq[N], lab[N], col[N], id[N], vis[N];
void mcs() {
  | for (int i = 0; i < n; i++) L[i].clear();
  | fill(lab + 1, lab + n + 1, 0);
  | fill(id + 1, id + n + 1, 0);
  | for (int i = 1; i <= n; i++) L[0].push_back(i);
  | int top = 0;
  | for (int k = n; k; k--) {
  |   | int x = -1;
  |   | for ( ; ; ) {
  |   |   | if (L[top].empty()) top --;
  |   |   | else {
  |   |   |   | x = L[top].back(), L[top].pop_back();
  |   |   |   | if (lab[x] == top) break;
  |   |   | }
  |   | }
  |   | seq[k] = x; id[x] = k;
  |   | for (auto v : E[x]) {
  |   |   | if (!id[v]) {
  |   |   |   | L[++lab[v]].push_back(v);
  |   |   |   | top = max(top, lab[v]);
  |   |   | } } } }
bool check() {
  | fill(vis + 1, vis + n + 1, 0);
  | for (int i = n; i; i--) {
  |   | int x = seq[i];
  |   | vector <int> to;
  |   | for (auto v : E[x])
  |   |   | if (id[v] > i) to.push_back(v);
  |   | if (to.empty()) continue;
  |   | int w = to.front();
  |   | for (auto v : to) if (id[v] < id[w]) w = v;
  |   | for (auto v : E[w]) vis[v] = i;
  |   | for (auto v : to)
  |   |   | if (v != w && vis[v] != i) return false;
  | } return true; }
void color() {
  | fill(vis + 1, vis + n + 1, 0);
  | for (int i = n; i; i--) {
  |   | int x = seq[i];
  |   | for (auto v : E[x]) vis[col[v]] = x;
  |   | for (int c = 1; !col[x]; c++)
  |   |   | if (vis[c] != x) col[x] = c;
  | } }
```

3.16 Minimum Mean Cycle 最小平均值环 $O(n^2)$

```
// 点标号为 1, 2, ···, n, 0为虚拟源点向其他点连权值为0的单向边.
// f[i][v] : 从 0 到 v 恰好经过 i 条路的最短路
ll f[N][N] = {Inf}; int u[M], v[M], w[M]; f[0][0] = 0;
for(int i = 1; i <= n + 1; i ++)
  | for(int j = 0; j < m; j ++)
  |   | f[i][v[j]] = min(f[i][v[j]], f[i - 1][u[j]] + w[j]);
double ans = Inf;
for(int i = 1; i <= n; i ++) {
  | double t = -Inf;
  | for(int j = 1; j <= n; j ++)
  |   | t = max(t, (f[n][i] - f[j][i]) / (double)(n - j));
  | ans = min(t, ans); }
```

3.17 Dinic 最大流

复杂度证明思路 假设 dist 为残量网络上的距离. Dinic 一轮增广会找到一个极大的长度为 $\text{dist}(s,t)$ 的增广路集合 blocking flow, 增广后 $\text{dist}(s,t)$ 将会增大. 因此只有 $O(V)$ 轮; 如果一轮增广是 $O(VE)$ 的, 总复杂度是 $O(V^2E)$. 没有当前弧优化的 Dinic 复杂度应是指数级别的.

单位流量网络 在 0-1 流量图上 Dinic 有更好的性质.

- 复杂度为 $O(\min\{V^{2/3}, E^{1/2}\}E)$.
- $\text{dist}(s,t) = d$, 残量网络上至多还存在 E/d 的流.
- 每个点只有一个入/出度时复杂度 $O(V^{1/2}E)$, 例如 Hopcroft-Karp.

```
int n, m, st, ed;
int h[N], e[M], w[M], ne[M], idx;
int d[N], cu[N];
void add(int a, int b, int c) {
  | e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
  | e[idx] = a, w[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}
bool bfs() {
  | memset(d, -1, sizeof d); d[st] = 0;
  | queue<int> q; q.push(st);
```

```
  | cu[st] = h[st];
  | while (q.size()) {
  |   | int ver = q.front(); q.pop();
  |   | for (int i = h[ver]; ~i; i = ne[i]) {
  |   |   | int to = e[i];
  |   |   | if (d[to] == -1 && w[i]) {
  |   |   |   | d[to] = d[ver] + 1;
  |   |   |   | cu[to] = h[to];
  |   |   |   | if (to == ed) return 1;
  |   |   |   | q.push(to);
  |   |   | }
  |   | }
  | }
  | return 0;
}
int Find(int pre, int limit) {
  | if (pre == ed) return limit;
  | int flow = 0;
  | for (int i = cu[pre]; ~i && flow < limit; i = ne[i]) {
  |   | cu[pre] = i;
  |   | int to = e[i];
  |   | if (d[to] == d[pre] + 1 && w[i]) {
  |   |   | int tmp = Find(to, min(w[i], limit - flow));
  |   |   | if (!tmp) d[to] = -1;
  |   |   | w[i] -= tmp, w[i ^ 1] += tmp, flow += tmp;
  |   | }
  | }
  | return flow;
}
int dinic() {
  | int maxf = 0, flow;
  | while (bfs()) while (flow = Find(st, inf)) maxf += flow;
  | return maxf;
}
void oper() {
  | cin >> n >> m >> st >> ed;
  | memset(h, -1, sizeof h); idx = 0;
  | for (int i = 1; i <= m; i++) {
  |   | int a, b, c; cin >> a >> b >> c;
  |   | add(a, b, c);
  | }
  | cout << dinic() << endl;
}
```

3.18 原始对偶费用流

```
bool bfs() {
  for (int i = S; i <= T; i++) cur[i] = head[i];
  for (int i = S; i <= T; i++) dep[i] = INF_int; // CHECK S-T
  dep[S] = 0; queue<int> q; q.push(S);
  while (!q.empty()) {
    int x = q.front(); q.pop();
    for (int i = head[x]; i; i = e[i].nxt) {
      int d = e[i].v;
      if (e[i].f > 0 && h[d] == h[x] + e[i].w
      && dep[d] > dep[x] + 1) {
        dep[d] = dep[x] + 1, q.push(d);
  } } } return dep[T] < INF_int; }
int dfs(int x, int lim) {
  if (x == T || !lim) return lim;
  int f = 0, flow = 0;
  for (int &i = cur[x]; i; i = e[i].nxt) {
    int d = e[i].v;
    if ((dep[d] == dep[x] + 1) && h[e[i].v] == e[i].w + h[x]
    && (f = dfs(d, min(lim, e[i].f)))) {
      e[i].f -= f; e[i ^ 1].f += f;
      flow += f; lim -= f;
      if (lim == 0) break;
  } } return flow; }
typedef pair <LL, int> pii;
pii solve() { // return {cost, flow}
  LL res = 0; int flow = 0;
  for (int i = 0; i <= T; ++i) h[i] = 0;
  int first = true;
  while (true) {
    priority_queue<pii, vector<pii>, greater<pii>> q;
    for (int i = S; i <= T; i++) dis[i] = INF_LL; // CHECK S-T
    dis[S] = 0;
    if (first) {
      // TODO: SSSP, may Bellman-Ford or DP
      first = false;
    } else { q.push(pii(0, S));
      while (!q.empty()) {
        pii now = q.top(); q.pop(); int x = now.second;
        if (dis[x] < now.first) continue;
        for (int o = head[x]; o; o = e[o].nxt) {
          LL w = dis[x] + e[o].w + h[x] - h[e[o].v];
          if (e[o].f > 0 && dis[e[o].v] > w) {
            dis[e[o].v] = w; q.push(pii(w, e[o].v));
    } } } }
    if (dis[T] >= INF_LL) break;
    // 所有点必须可达，可以加 T-0->i: min(dis[i], dis[T])
    for (int i = 0; i <= T; ++i) h[i] += dis[i];
    int fl = 0; while (bfs()) fl += dfs(S, INF_int);
    res += fl * h[T]; flow += fl;
  }
return make_pair(res, flow); }
```

3.19 K短路

```
int F[N],FF[N];namespace Left_Tree{//可持久化左偏树
  | struct P{int l,r,h,v,x,y;}Tr[N*40];int RT[N],num;
  | //l和r是左右儿子,h是高度,v是数值,x和y是在图中的两点
  | int New(P o){Tr[++num]=o;return num;}
  | void start()
      ↪{num=0;Tr[0].l=Tr[0].r=Tr[0].h=0;Tr[0].v=inf;}
  | int mg(int x,int y){
  |   | if(!x)return y;
  |   | if(Tr[x].v>Tr[y].v)swap(x,y);
  |   | int o=New(Tr[x]);Tr[o].r=mg(Tr[o].r,y);
  |   | if(Tr[Tr[o].l].h<Tr[Tr[o].r].h)swap(Tr[o].l,Tr[o].r);
  |   | Tr[o].h=Tr[Tr[o].r].h+1;return o;}
  | void add(int&k,int v,int x,int y){
  |   | int o=New(Tr[0]);
  |   | Tr[o].v=v;Tr[o].x=x;Tr[o].y=y;
  |   | k=mg(k,o); | }}
using namespace Left_Tree;
struct SPFA{//SPFA, 这里要记录路径
  | void in(){tot=0;CL(fir);}void ins(x,y,z){}
  | void work(int S,int n){//F[] 求最短路从哪个点来，FF[] 记最短路从哪条边来
}}A;
struct Kshort{
int tot,n,m,S,T,k,fir[N],va[M],la[M],ne[M];bool v[N];
struct P{
  | int x,y,z;P(){}P(int x,int y,int z):x(x),y(y),z(z){}
  | bool operator<(P a)const{return a.z<z;}};
priority_queue<P>Q;void in(){tot=0;CL(fir);}
void ins(x,y,z){}
void init(){//将图读入
  | int i,x,y,z;in();A.in();start();rd(n,m)
  | fr(i,1,m)rd(x,y,z),A.ins(y,x,z),ins(x,y,z);
  | rd(S,T,k);if(S==T)k++;//注意起点终点相同的情况
  | A.work(T,n);}//A是反向边
void dfs(int x){
  | if(v[x])return;v[x]=1;if(F[x])RT[x]=RT[F[x]];
  | for(int i=fir[x],y;i;i=ne[i])if(y=la[i],A.d[y]!=inf&&FF[x]!=i)
  |   | add(RT[x],A.d[y]-A.d[x]+va[i],x,y);
  | for(int i=A.fir[x];i;i=A.ne[i])if(F[A.la[i]]==x)dfs(A.la[i]);}
int work(){//返回答案，没有返回-1
  | int i,x;dfs(T);
  | if(!--k)return A.d[S]==inf?-1:A.d[S];
  | P u,w;if(RT[S])Q.push(P(S,RT[S],A.d[S]+Tr[RT[S]].v));
  | for(;k--;){
  |   | if(Q.empty())return -1;u=Q.top();Q.pop();
  |   | if(x=mg(Tr[u.y].l,Tr[u.y].r))
  |   |   | Q.push(P(u.x,x,Tr[x].v-Tr[u.y].v+u.z));
  |   | if(RT[x=Tr[u.y].y])Q.push(P(x,RT[x],u.z+Tr[RT[x]].v));}
  | return u.z;}}G;
```

3.20 树链剖分

已知一棵包含 N 个结点的树（连通且无环），每个节点上包含一个数值，需要支持以下操作：

‘1 x y z‘，表示将树从 x 到 y 结点最短路径上所有节点的值都加上 z。

‘2 x y‘，表示求树从 x 到 y 结点最短路径上所有节点的值之和。

‘3 x z‘，表示将以 x 为根节点的子树内所有节点值都加上 z。

‘4 x‘ 表示求以 x 为根节点的子树内所有节点值之和

```cpp
vector<int> adj[N];
int n,m,r,p,cnt;
int son[N],depth[N],fa[N],size[N];
int id[N],top[N],w[N];
ll c1[N],c2[N];
//以下为树状数组
inline int lowbit(int x){
  return x&(-x);
}
inline void add(int l,int r,int x){
  x %= p;
  int ad1 = (ll)(l-1)*x%p;
  int ad2 = (ll)r*x%p;
  for(int t=l;t<=n;t+=lowbit(t)){
    c1[t] = (c1[t]+x)%p;
    c2[t] = (c2[t]+ad1)%p;
  }
  for(int t=r+1;t<=n;t+=lowbit(t)){
    c1[t] = (c1[t]-x)%p;
    c1[t] = (c1[t]+p)%p;
    c2[t] = (c2[t]-ad2)%p;
    c2[t] = (c2[t]+p)%p;
  }
}
inline int qwq(int i){ //qwq
  int res = 0;
  for(int t=i;t>0;t-=lowbit(t)){
    res = (res+(ll)i*c1[t]%p)%p;
    res = (res-c2[t])%p;
    res = (res+p)%p;
  }
  return res;
}

inline int query(int l,int r){
  int res = (qwq(r)-qwq(l-1))%p;
  return (res+p)%p;
}
//以上树状数组
inline void read(int &x){
  x = 0;
  char c = getchar();
  while(c<'0'||c>'9') c = getchar();
  while(c>='0'&&c<='9'){
    x = (x<<3)+(x<<1)+(c^48);
    c = getchar();
  }
}
void print(int x){
  if(x>9) print(x/10);
  putchar(x%10+'0');
}
void dfs1(int u,int f){ //u为当前节点，f为父节点
  fa[u] = f;
  size[u] = 1; //子树大小要算上子树的根节点，也就是u
  depth[u] = depth[f]+1; //比父亲深度大1
  int v,t = -1,l = adj[u].size(); //此处使用vector存图
  for(int i=0;i<l;++i){ //遍历连接u的点v
    v = adj[u][i];
    if(v==f) continue;
    dfs1(v,u);
    size[u] += size[v];
    if(size[v]>t){
      //如果这个子树大小比已找到的还大，那就更新已找到的，同时更新u的重儿子为v
      t = size[v];
      son[u] = v;
    }
  }
}
void dfs2(int u,int f){ //这里f就不是指父亲了，是u所在重链的顶端节点
  top[u] = f;
  id[u] = ++cnt;
  if(w[u]!=0)
    add(id[u],id[u],w[u]); //树状数组维护区间和，w[u] 为u的初始权值
  if(son[u]==0) return; //重儿子编号为0意味着没有儿子，返回
  dfs2(son[u],f); //先从重儿子开始dfs，这样可以使一条重链上的节点id连续，便于区间操作
  int v,l = adj[u].size();
  for(int i=0;i<l;++i){
    v = adj[u][i];
    if(v==son[u]||v==fa[u]) continue;
    dfs2(v,v); //由于是轻儿子，所以其所在重链顶端节点自然是自己
  }
}
int queryPath(int u,int v){
  int res = 0;
  while(top[u]!=top[v]){
    if(depth[top[u]]<depth[top[v]])
      swap(u,v);
    res = (res+query(id[top[u]],id[u]))%p;
    u = fa[top[u]];
  }
  if(depth[u]>depth[v]) swap(u,v);
  res = (res+query(id[u],id[v]))%p;
  return res;
}
void addPath(int u,int v,int k){
  k %= p;
  while(top[u]!=top[v]){
    if(depth[top[u]]<depth[top[v]])
      swap(u,v); //深度大的点先跳，保证能跳到一条重链上
    add(id[top[u]],id[u],k);
    u = fa[top[u]];
  }
  if(depth[u]>depth[v]) swap(u,v);
  add(id[u],id[v],k);//在一条重链上，直接加
}
int querySon(int u){
  return query(id[u],id[u]+size[u]-1);
}
void addSon(int u,int k){
  k %= p;
  add(id[u],id[u]+size[u]-1,k);
}
inline int lca(int u,int v){
  while(top[u]!=top[v]){
    if(depth[top[u]]<depth[top[v]])
      swap(u,v);
    u = fa[top[u]];
  }
  if(depth[u]<depth[v]) return u;
  return v;
}
int main(){
  int u,v;
  read(n),read(m),read(r),read(p);
  for(int i=1;i<=n;++i)
    read(w[i]);
  for(int i=1;i<n;++i){
    read(u),read(v);
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  dfs1(r,0);
  dfs2(r,r);
  int ans,op,x,y,z;
  while(m--){
    read(op),read(x);
    if(op==1){
      read(y),read(z);
      addPath(x,y,z);
      continue;
    }
    if(op==2){
      read(y);
      ans = queryPath(x,y);
      print(ans);
      putchar('\n');
      continue;
    }
    if(op==3){
      read(z);
      addSon(x,z);
      continue;
    }
    ans = querySon(x);
    print(ans);
    putchar('\n');
  }
```

```
  | return 0;
}
```

3.21 网络流总结

最小割集, 最小割必须边以及可行边

最小割集 从 S 出发, 在残余网络中BFS所有权值非 0 的边 (包括反向边), 得到点集 $\{S\}$, 另一集为 $\{V\}-\{S\}$.

最小割集必须点 残余网络中S直接连向的点必在S的割集中, 直接连向T的点必在T的割集中; 若这些点的并集为全集, 则最小割方案唯一.

最小割可行边 在残余网络中求强联通分量, 将强联通分量缩点后, 剩余的边即为最小割可行边, 同时这些边也必然满流.

最小割必须边 在残余网络中求强联通分量, 若S出发可到u, T出发可到v, 等价于 $\mathrm{scc}_S=\mathrm{scc}_u$ 且 $\mathrm{scc}_T=\mathrm{scc}_v$, 则该边为必须边.

常见问题

最大权闭合子图 适用问题: 每个点有点权, 限制条件形如: 选择A则必须选择B, 选择B则必须选择C, D. 建图方式: B向A连边, CD向B连边. 求解: S向正权点连边, 负权点向T连边, 其余边容量 ∞, 求最小割, 答案为S所在最小割集.

二元关系 适用问题: 有 n 个元素, 每个元素可选A或者B, 各有代价; 有 m 个限制条件, 若元素 i 与 j 的种类不同则产生额外的代价, 求最小代价. 求解: S向i连边 A_i, i向T连边 B_i, 一组限制 (i,j) 代价为 z, 则i与j之间连双向容量为 z 的边, 求最小割.

混合图欧拉回路 把无向边随便定向, 计算每个点的入度和出度, 如果有某个点出入度之差 $\deg_i=\mathrm{in}_i-\mathrm{out}_i$ 为奇数, 肯定不存在欧拉回路. 对于 $\deg_i>0$ 的点, 连接边 $(i,T,\deg_i/2)$; 对于 $\deg_i<0$ 的点, 连接边 $(S,i,-\deg_i/2)$. 最后检查是否满流即可.

二物流 水源 S_1, 水汇 T_1, 油源 S_2, 油汇 T_2, 每根管道流量共用. 求流量和最大. 建超级源 SS_1 汇 TT_1, 连边 $SS_1\to S_1$,$SS_1\to S_2$,$T_1\to TT_1$,$T_2\to TT_1$, 设最大流为 x_1. 建超级源 SS_2 汇 TT_2, 连边 $SS_2\to S_1$, $SS_2\to T_2$,$T_1\to TT_2$, $S_2\to TT_2$, 设最大流为 x_2. 则最大流中水流量 $\frac{x_1+x_2}{2}$, 油流量 $\frac{x_1-x_2}{2}$.

一些网络流建图

无源汇有上下界可行流 每条边 (u,v) 有一个上界容量 $C_{u,v}$ 和下界容量 $B_{u,v}$, 我们让下界变为 0, 上界变为 $C_{u,v}-B_{u,v}$, 但这样做流量不守恒. 建立超级源点 SS 和超级汇点 TT, 用 du_i 来记录每个节点的流量情况, $du_i=\sum B_{j,i}-\sum B_{i,j}$, 添加一些附加弧. 当 $du_i>0$ 时, 连边 (SS,i,du_i); 当 $du_i<0$ 时, 连边 $(i,TT,-du_i)$. 最后对 (SS,TT) 求一次最大流即可, 当所有附加边全部满流时 (即 maxflow $==du_i>0$) 时有可行解.

有源汇有上下界最大可行流 建立超级源点 SS 和超级汇点 TT, 首先判断是否存在可行流, 用无源汇有上下界可行流的方法判断. 增设一条从 T 到 S 没有下界容量为无穷的边, 那么原图就变成了一个无源汇有上下界可行流问题. 同样地建图后, 对 (SS,TT) 进行一次最大流, 判断是否有可行解. 如果有可行解, 删除超级源点 SS 和超级汇点 TT, 并删去 T 到 S 的这条边, 再对 (S,T) 进行一次最大流, 此时得到的 maxflow 即为有源汇有上下界最大可行流.

有源汇有上下界最小可行流 建立超级源点 SS 和超级汇点 TT, 和无源汇有上下界可行流一样新增一些边, 然后从SS到TT跑最大流. 接着加上边 (T,S,∞), 再从 SS 到 TT 跑一遍最大流. 如果所有新增边都是满的, 则存在可行流, 此时 T 到 S 这条边的流量即为最小可行流.

有上下界费用流 如果求无源汇有上下界最小费用可行流或有源汇有上下界最小费用最大可行流, 用1.6.3.1/1.6.3.2 的构图方法, 给边加上费用即可. 求有源汇有上下界最小费用最小可行流, 要先用1.6.3.3的方法建图, 先求出一个保证必要边满流情况下的最小费用. 如果费用全部非负, 那么这时的费用就是答案. 如果费用有负数, 那么流多了可能更好, 继续做从 S 到 T 的流量任意的最小费用流, 加上原来的费用就是答案.

费用流消负环 新建超级源SS汇TT, 对于所有流量非空的负权边e, 先流满(`ans+=e.f*e.c, e.rev.f+=e.f, e.f=0`), 再连边SS→e.to, e.from→TT, 流量均为e.f(>0), 费用均为0. 再连边T→S流量 ∞ 费用0. 此时没有负环了. 做一遍SS到TT的最小费用最大流, 将费用累加ans, 拆掉T→S的那条边 (此边的流量为残量网络中S→T的流量). 此时负环已消, 再继续跑最小费用最大流.

整数线性规划转费用流

首先将约束关系转化为所有变量下界为 0, 上界没有要求, 并满足一些等式, 每个变量在均在等式左边且出现恰好两次, 系数为 +1 和 −1, 优化目标为 $\max\sum v_ix_i$ 的形式. 将等式看做点, 等式i右边的值 b_i 若为正, 则 S 向 i 连边 $(b_i,0)$, 否则i向T连边 $(-b_i,0)$. 将变量看做边, 记变量 x_i 的上界为 m_i(无上界则 $m_i=inf$), 将 x_i 系数为 +1 的那个等式 u 向系数为 −1 的等式 v 连边 (m_i,v_i).

3.22 图论结论

3.22.1 最小乘积问题原理

每个元素有两个权值 $\{x_i\}$ 和 $\{y_i\}$, 要求在某个限制下 (例如生成树, 二分图匹配) 使得 $\Sigma x\Sigma y$ 最小. 对于任意一种符合限制的选取方法, 记 $X=\Sigma x_i, Y=\Sigma y_i$, 可看做平面内一点 (X,Y). 答案必在下凸壳上, 找出该下凸壳所有点, 即可枚举获得最优答案. 可以递归求出此下凸壳所有点, 分别找出距 x,y 轴最近的两点 A,B, 分别对应于 $\Sigma y_i,\Sigma x_i$ 最小. 找出距离线段最远的点 C, 则 C 也在下凸壳上, C 点满足 $AB\times AC$ 最小, 也即

$$(X_B-X_A)Y_C+(Y_A-Y_B)X_C-(X_B-X_A)Y_A-(Y_B-Y_A)X_A$$

最小, 后两项均为常数, 因此将所以权值改成 $(X_B-X_A)y_i+(Y_B-Y_A)x_i$, 求同样问题 (例如最小生成树, 最小权匹配) 即可. 求出 C 点以后, 递归 AC, BC.

3.22.2 最小环

无向图最小环: 每次floyd到 k 时, 判断 1 到 $k-1$ 的每一个 i,j:

$$\mathrm{ans}=\min\{\mathrm{ans},d(i,j)+G(i,k)+G(k,j)\}.$$

有向图最小环: 做完floyd后, $d(i,i)$ 即为经过 i 的最小环.

3.22.3 度序列的可图性

判断一个度序列是否可转化为简单图, 除了一种贪心构造的方法外, 下列方法更快速. EG定理: 将度序列从大到小排序得到 $\{d_i\}$, 此序列可转化为简单图当且仅当 Σd_i 为偶数, 且对于任意的 $1\le k\le n-1$ 满足 $\sum_{i=1}^{k}d_i\le k(k-1)+\sum_{i=k+1}^{n}\min(k,d_i)$.

3.22.4 切比雪夫距离与曼哈顿距离转化

曼哈顿转切比雪夫: $(x+y,x-y)$, 适用于一些每次只能向四联通的格子走一格的问题. 切比雪夫转曼哈顿: $(\frac{x+y}{2},\frac{x-y}{2})$, 适用于统计距离.

3.22.5 树链的交

```
bool cmp(int a,int b){return dep[a]<dep[b];}
path merge(path u, path v){
  | int d[4], c[2];
  | if (!u.x||!v.x) return path(0, 0);
  | d[0]=lca(u.x,v.x); d[1]=lca(u.x,v.y);
  | d[2]=lca(u.y,v.x); d[3]=lca(u.y,v.y);
  | c[0]=lca(u.x,u.y); c[1]=lca(v.x,v.y);
  | sort(d,d+4,cmp); sort(c,c+1,cmp);
  | if (dep[c[0]] <= dep[d[0]] && dep[c[1]] <= dep[d[2]])
  |   | return path(d[2],d[3]);
  | else return path(0, 0); }
```

3.22.6 带修改MST

维护少量修改的最小生成树, 可以缩点缩边使暴力复杂度变低. (银川 21: 求有 16 个‘某两条边中至少选一条’的限制条件的最小生成树)

找出必须边 将修改边标 $-\infty$, 在MST上的其余边为必须边, 以此缩点.

找出无用边 将修改边标 ∞, 不在MST上的其余边为无用边, 删除之.

假设修改边数为 k, 操作后图中最多剩下 $k+1$ 个点和 $2k$ 条边.

3.22.7 差分约束

$x_r-x_l\le c$ `:add(l, r, c)` $x_r-x_l\ge c$ `:add(r, l, -c)`

3.22.8 李超线段树

添加若干条线段或直线 $(a_i,b_i)\to(a_j,b_j)$, 每次求 $[l,r]$ 上最上面的那条线段的值. 思想是让线段树中一个节点只对应一条直线, 如果在这个区间加入一条直线, 如果一段比原来的优, 一段比原来的劣, 那么判断一下两条线的交点, 判断哪条直线可以完全覆盖一段一半的区间, 把它保留, 另一条直线下传到另一半区间. 时间复杂度 $O(n\log n)$.

3.22.9 Segment Tree Beats

区间 min, max, 区间求和. 以区间取 min 为例, 额外维护最大值 m, 严格次大值 s 以及最大值个数 t. 现在假设我们要让区间 $[L,R]$ 对 x 取 min, 先在线段树中定位若干个节点, 对于每个节点分三种情况讨论: 1, 当 $m\le x$ 时, 显然这一次修改不会对这个节点产生影响, 直接退出; 2, 当 $se<x<ma$ 时, 显然这一次修改只会影响到所有最大值, 所以把 num 加上 $t*(x-ma)$, 把 ma 更新为 x, 打上标记退出; 3, 当 $se\ge x$ 时, 无法直接更新着一个节点的信息, 对当前节点的左儿子和右儿子递归处理. 单次操作均摊复杂度 $O(\log^2 n)$.

3.22.10 二分图

最小点覆盖=最大匹配数. 独立集与覆盖集互补. 最小点覆盖构造方法: 对二分图流图求割集, 跨过的边指示最小点覆盖. **Hall定理** $G=(X,Y,E)$, $|M|=|X|\Leftrightarrow\forall S\subseteq X,|S|\le|A(S)|$.

3.22.11 稳定婚姻问题

男士按自己喜欢程度从高到底依次向每位女士求婚, 女士遇到更喜欢的男士时就接受他, 并抛弃以前的配偶. 被抛弃的男士继续按照列表向剩下的女士依次求婚, 直到所有人都有配偶. 算法一定能得到一个匹配, 而且这个匹配一定是稳定的. 时间复杂度 $O(n^2)$.

3.22.12 三元环

对于无向边 (u,v), 如果 $\deg_u<\deg_v$, 那么连有向边 (u,v)(以点标号为第二关键字). 枚举 x 暴力即可. 时间复杂度 $O(m\sqrt{m})$.

3.22.13 图同构

令 $F_t(i)=(F_{t-1}(i)*A+\sum_{i\to j}F_{t-1}(j)*B+\sum_{j\to i}F_{t-1}(j)*C+D*(i-a))\bmod P$, 枚举点 a, 迭代 K 次后求得的就是 a 点所对应的 $hash$ 值, 其中 K,A,B,C,D,P 为 $hash$ 参数, 可自选.

3.22.14 竞赛图 Landau's Theorem

n 个点竞赛图点按出度按升序排序, 前 i 个点的出度之和不小于 $\frac{i(i-1)}{2}$, 度数总和等于 $\frac{n(n-1)}{2}$. 否则可以用优先队列构造出方案.

3.22.15 Ramsey Theorem R(3,3)=6, R(4,4)=18

6 个人中存在 3 人相互认识或者相互不认识.

3.22.16 树的计数 Prufer序列

树和其prufer编码一一对应, 一颗 n 个点的树, 其prufer编码长度为 $n-2$, 且度数为 d_i 的点在prufer 编码中出现 d_i-1 次.

由树得到序列: 总共需要 $n-2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为Prufer序列的第 i 个元素 p_i, 并将

此叶子节点从树中删除, 直到最后得到一个长度为 $n-2$ 的Prufer 序列和一个只有两个节点的树.
由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在Prufer序列中出现的次数, 得到每个点的度; 执行 $n-2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v=p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.
相关结论: n 个点完全图, 每个点度数依次为 d_1,d_2,...,d_n, 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!...(d_n-1)!}$.
左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1}\times n_2^{n_1-1}$.
m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2}\prod c_i$

3.22.17 有根树的计数

首先, 令 $S_{n,j}=\sum_{1\le j\le n/j}$; 于是 $n+1$ 个结点的有根树的总数为 $a_{n+1}=\frac{\sum_{j=1}^{n} j a_j S_{n-j}}{n}$. 注: $a_1=1,a_2=1,a_3=2,a_4=4,a_5=9,a_6=20,a_9=286,a_{11}=1842$.

3.22.18 无根树的计数

n 是奇数时, 有 $a_n-\sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.
n 时偶数时, 有 $a_n-\sum_i^{n/2} a_i a_{n-i}+\frac{1}{2}a_{n/2}(a_{n/2}+1)$ 种不同的无根树.

3.22.19 生成树计数 Kirchhoff's Matrix-Tree Thoerem

Kirchhoff Matrix $T=Deg-A$, Deg 是度数对角阵, A 是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.
邻接矩阵 $A[u][v]$ 表示 $u\to v$ 边个数, 重边按照边数计算, 自环不计入度数.
无向图生成树计数: $c=|K$ 的任意1个 $n-1$ 阶主子式 $|$
有向图外向树计数: $c=|$ 去掉根所在的那阶得到的主子式 $|$

3.22.20 有向图欧拉回路计数 BEST Thoerem

$$ec(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

其中 deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.
欧拉连通图中任意两点外向树个数相同: $t_v(G)=t_w(G)$.

3.22.21 Tutte Matrix

Tutte matrix A of a graph $G=(V,E)$:

$$A_{ij}=\begin{cases}x_{ij} & \text{if } (i,j)\in E \text{ and } i<j\\ -x_{ij} & \text{if } (i,j)\in E \text{ and } i>j\\ 0 & \text{otherwise}\end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i<j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

3.22.22 Edmonds Matrix

Edmonds matrix A of a balanced ($|U|=|V|$) bipartite graph $G=(U,V,E)$:

$$A_{ij}=\begin{cases}x_{ij} & (u_i,v_j)\in E\\ 0 & (u_i,v_j)\notin E\end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $det(A_{ij})$ 不恒为 0. 完美匹配的个数等于多项式中单项式的个数.

3.22.23 有向图无环定向, 色多项式

图的色多项式 $P_G(q)$ 对图 G 的 q-染色计数.
Triangle K_3 : $x(x-1)(x-2)$
Complete graph K_n : $x(x-1)(x-2)\cdots(x-(n-1))$
Tree with n vertices : $x(x-1)^{n-1}$
Cycle C_n : $(x-1)^n+(-1)^n(x-1)$
acyclic orientations of an n-vertex graph G is $(-1)^n P_G(-1)$.

3.22.24 拟阵交问题

最大带权拟阵交问题: 全集 U 中每个元素都有权值 w_i. 设同一个全集 U 上有两个满足拟阵性质的集族 $\mathcal{F}_1,\mathcal{F}_2$. 对于 $k=1..|U|$, 分别求出一个集合 S, 满足 $S\in\mathcal{F}_1\cap\mathcal{F}_2$ 且 $|S|$ 恰好为 k 的前提下, S 中元素权值和最小.
设集合大小为 k 时已经求出了答案 S. 现在希望求出集合大小为 $k+1$ 的答案. U 中所有元素分为两个集合: 当前答案集合 S, 和剩余集合 $T=U\backslash S$. 考虑 T 中的某个元素 x_i. 记 $A=\{x_i|S\cup\{x_i\}\in\mathcal{F}_1\}$, $B=\{x_i|S\cup\{x_i\}\in\mathcal{F}_2\}$. 如果 T 中某个元素 $x_i\notin A$, 说明 x_i 加进 S 中形成了某个" 环", 从而不满足 $\mathcal{F}_1$ 的限制. 考虑这个" 环" 上每个元素 y_j, 满足 $S\backslash\{y_j\}\cup\{x_i\}\in\mathcal{F}_1$, 将 x_i 向每个 y_j 连边. 如果 T 中某个元素 $x_i\notin B$, 同理找出 S 中每一个元素 y_j 使得 $S\backslash\{y_j\}\cup\{x_i\}\in\mathcal{F}_2$, 将 y_j 向 x_i 连边.
现在求出从 A 到 B 的多源多汇最短路, 权值在点上, 若点属于 T 则权值为正, 否则属于 S, 权值为负. 最短路上每个 T 中的点放进 S, S 中的点放进 T, 则完成了一次增广. 由于每次增广路的起点和终点都在 T 中, 所以每次增广都会使得 $|S|$ 增加1.
最大拟阵交问题可以去掉权值直接求增广路.

3.22.25 双极定向

```cpp
//双极定向: 给定无向图和两个极点s,t, 要求将每条边定向后成为DAG, 使得s可达所有点, 所有点均可达t
//topo为定向后DAG的拓扑序, 边 (u,v) 定向为u->v当且仅当拓扑序中u在v的前面.
int n, dfn[N], low[N], stamp, p[N], preorder[N], topo[N];
bool fucked = 0, sign[N]; vector<int> G[N];
void dfs(int x, int fa, int s, int t){
  dfn[x] = low[x] = ++stamp;
  preorder[stamp] = x, p[x] = fa;
  if (x == s) dfs(t, x, s, t);
  for (int y : G[x]){
    if (x == s && y == t) continue;
    if (!dfn[y]){
      if (x == s) fucked = true;
      dfs(y, x, s, t);
      low[x] = min(low[x], low[y]); }
    else if (dfn[y] < dfn[x] && y != fa)
      low[x] = min(low[x], dfn[y]); } }
bool bipolar_orientation(int s, int t){
  G[s].push_back(t), G[t].push_back(s);
  stamp = fucked = 0, dfs(s, s, s, t);
  for (int i = 1; i <= n; i++)
    if (i != s && (!dfn[i] || low[i] >= dfn[i]))
      fucked = true;
  if (fucked) return false;
  sign[s] = 0;//memset sign[] is not necessary
  int pre[n + 5], suf[n + 5]; // list
  suf[0] = s; pre[s] = 0, suf[s] = t;
  pre[t] = s, suf[t] = n + 1; pre[n + 1] = t;
  for (int i = 3; i <= n; i++){
    int v = preorder[i];
    if (!sign[preorder[low[v]]]){ // insert before p[v]
      int P = pre[p[v]];
      pre[v] = P, suf[v] = p[v];
      suf[P] = pre[p[v]] = v; }
    else{ // insert after p[v]
      int S = suf[p[v]];
      pre[v] = p[v], suf[x] = S;
      suf[p[v]] = pre[S] = v; }
    sign[p[x]] = !sign[preorder[low[x]]]; }
  for (int x = s, cnt = 0; x != n + 1; x = suf[x])
    topo[++cnt] = x;
  return true; }
```

3.22.26 图中的环

没有奇环的图是二分图, 没有偶环的图是仙人掌. 判定没有奇环仅用深度奇偶性判即可; 判定没有偶环的图需要记录覆盖次数判定是否存在奇环有交.

4. Math 数学

4.1 高斯消元

```cpp
int n;
double A[N][N];
int eq, va; // 方程数, 变量数
double x[N]; // 解集
int sgn(double x) {
  return (x > eps) - (x < -eps);
}
int gauss() {
  eq = n, va = n;
  int i, j, k, max_r, col;
  double tmp;
  col = 0;
  for (k = 0; k < eq && col < va; k++, col++) {
    max_r = k;
    for (i = k + 1; i < eq; i++) if (sgn(fabs(A[i][col]) - fabs(A[max_r][col])) > 0) max_r = i;
    if (max_r != k) {
      for (j = k; j < va + 1; j++) swap(A[k][j], A[max_r][j]);
    }
    if (!sgn(A[k][col])) {
      k--;
      continue;
    }
    for (i = k + 1; i < eq; i++) {
      if (sgn(A[i][col])) {
        tmp = A[i][col] / A[k][col];
        for (j = col; j < va + 1; j++) A[i][j] = A[i][j] - A[k][j] * tmp;
      }
    }
```

```
  |  | }
  | }
  | for (i = k; i < eq; i++) if (sgn(A[i][col])) return -1;
  | if (k < va) return va - k;
  | for (i = va - 1; i >= 0; i--) {
  |  | tmp = A[i][va];
  |  | for (j = i + 1; j < va; j++) if (sgn(A[i][j])) tmp -=
  |      ↪A[i][j] * x[j];
  |  | x[i] = tmp / A[i][i];
  | }
  | return 0;
}
void oper() {
  | cin >> n;
  | for (int i = 0; i < n; i++) for (int j = 0; j < n + 1;
  |     ↪j++) cin >> A[i][j];
  | int ok = gauss();
  | if (ok == -1) cout << "No solution" << endl;
  | else if (ok >= 1) cout << "Infinite group solutions" <<
  |     ↪endl;
  | else for (int i = 0; i < n; i++) cout << fixed <<
  |     ↪setprecision(2) << x[i] << endl;
}
```

4.2 Lucas

```
//C(n, m) mod p(n 很大 p 较小 (不知道能不能为非素数)
LL Lucas(LL n, LL m, const int &pr) {
  | if (m == 0) return 1;
  | return C(n % pr, m % pr, pr) * Lucas(n / pr, m / pr, pr)
  |     ↪% pr;
}
```

4.3 计算C(n,0) 到C(n,p) 的值

```
//by Yuhao Du
int p;
std::vector<int> gao(int n) {
  | std::vector<int> ret(p+1,0);
  | if (n==0) {
  |  | ret[0]=1;
  | } else if (n%2==0) {
  |  | std::vector<int> c = gao(n/2);
  |  | for(int i = 0; i <= p+1; i++) {
  |  |  | for(int j = 0; j <= p+1; j++) {
  |  |  |  | if (i+j<=p) ret[i+j]+=c[i]*c[j];
  |  |  | }
  |  | }
  | } else {
  |  | std::vector<int> c = gao(n-1);
  |  | for(int i = 0; i <= p+1; i++) {
  |  |  | for(int j = 0; j <= 2; j++) {
  |  |  |  | if (i+j<=p) ret[i+j]+=c[i];
  |  |  | }
  |  | }
  | }
  | return ret;
}
```

4.4 线性求逆元

```
int inv[MAXN],f[MAXN],fi[MAXN];
void pre_inv(int n,int p) {
  | inv[1]=1;
  | f[0]=fi[0]=f[1]=fi[1]=1;
  | for(int i=2;i<=n;i++) {
  |  | inv[i]=1LL*(p-p/i)*inv[p%i]%p;
  |  | f[i]=1LL*f[i-1]*i%p;
  |  | fi[i]=1LL*fi[i-1]*inv[i]%p;
  | }
}
```

```
int fact[N], inv[N], infact[N];

void preoper() {
  | fact[0] = 1, infact[0] = 1;
  | for (int i = 1; i < N; i++) {
  |  | fact[i] = fact[i - 1] * i % mod;
  |  | inv[i] = i == 1 ? 1 : (mod - mod / i) * inv[mod % i]
  |      ↪% mod;
  |  | infact[i] = infact[i - 1] * inv[i] % mod;
  | }
}

int C(int n, int k) {
  | return fact[n] * infact[k] % mod * infact[n - k] % mod;
}
```

4.5 数论分块

```
LL ans=0;
for(LL i=1;i<=n;i++){
  | LL t=n/i,j=n/t;
  | ans+=(j-i+1)*t;
  | i=j;
}
printf("%lld\n", ans);
```

4.6 Sieve 筛法

```
vector<int> minp(n), primes;
void sieve(int n) {
  | minp.assign(n + 1, 0); primes.clear();
  | for (int i = 2; i <= n; i++) {
  |  | if (minp[i] == 0) {
  |  |  | minp[i] = i; primes.push_back(i);
  |  | }
  |  | for (auto p : primes) {
  |  |  | if (i * p > n) { break; }
  |  |  | minp[i * p] = p;
  |  |  | if (p == minp[i]) { break; }
  |  | }
  | }
}
```

4.7 杜教筛

$$S_\varphi(n) = \frac{n(n+1)}{2} - \sum_{d=2}^{n} S_\varphi\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$$

$$S_\mu(n) = 1 - \sum_{d=2}^{n} S_\mu\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$$

4.8 原根

定义 使得 $a^x \bmod m = 1$ 的最小的x, 记作 $\delta_m(a)$. 若 $a \equiv g^s \bmod m$, 其中 g 为 m 的一个原根. 则虽然 s 随 g 的不同取值有所不同, 但是必然满足 $\delta_m(a) = \gcd(s, \varphi(m))$.

性质 $\delta_m(a^k) = \frac{\delta_m(a)}{\gcd(\delta_m(a),k)}$

k 次剩余 给定方程 $x^k \equiv a \bmod m$, 求所有解. 若 k 与 $\varphi(m)$ 互质, 则可以直接求出 k 对 $\varphi(m)$ 的逆元. 否则, 将 k 拆成两部分, $k = uv$, 其中 $u \perp \varphi(m)$, $v | \varphi(m)$, 先求 $x^v \equiv a \bmod m$, 则 $ans = x^{u^{-1}}$. 下面讨论 $k | \varphi(m)$ 的问题. 任取一原根 g, 对两侧取离散对数, 设 $x = g^s$, $a = g^t$, 其中 t 可以用BSGS求出, 则问题转化为求出所有的 s 满足 $ks \equiv t \bmod \varphi(m)$, exgcd 即可求解, 显然有解的条件是 $k | \delta_m(a)$.

4.9 Long Long $O(1)$ 乘

```
// kactl,  M ≤ 7.2 · 10^18
ULL modmul(ULL a, ULL b, LL M) {
  | LL ret = a * b - M * ULL(1.L / M * a * b);
  | return ret + M * (ret < 0) - M * (ret >= (LL)M); }
// orz@CF, M in 63 bit
ULL modmul(ULL a, ULL b, LL M) {
  | ULL c = (long double)a * b / M;
  | LL ret = LL(x * y - c * M) % M; // must be signed
  | return ret < 0 ? ret + M : ret;}
// use int128 instead if c > 63 bit
```

4.10 exgcd

假设我们已经找到了一组解 (p_0, q_0) 满足 $ap_0 + bq_0 = \gcd(a, b)$, 那么其他的解都满足

$$p = p_0 + \frac{b}{\gcd(p,q)} \times t \qquad q = q_0 - \frac{a}{\gcd(p,q)} \times t$$

其中t为任意整数.
如果 a 和 b 互质, 那么 p_0 是 a 在模 b 下的逆元。

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
  | if (b == 0) return x = 1, y = 0, a;
```

```
  | LL t = exgcd(b, a % b, y, x);
  | y -= a / b * x; return t;}
LL inv(LL x, LL m) {
  | LL a, b; exgcd(x, m, a, b); return (a % m + m) % m; }
```

```
LL exgcd(LL a, LL b, LL& x, LL& y) {
  | if (!b) {
  |   | x = 1, y = 0;
  |   | return a;
  | }
  | int g = exgcd(b, a % b, y, x);
  | y -= a / b * x;
  | return g;
}
void oper() {
  | cin >> a >> m >> b;
  | LL gcd = exgcd(a, b, x, y);
  | x = (x * m / gcd);
  | LL z = b / gcd;
  | x = (x % z + z) % z;
  | if (m % gcd != 0) cout << "impossible" << endl;
  | else cout << x << endl;
}
```

4.11 CRT 中国剩余定理

```
bool crt_merge(LL a1, LL m1, LL a2, LL m2, LL &A, LL &M) {
LL c = a2 - a1, d = __gcd(m1, m2); //合并两个模方程
if(c % d) return 0; // gcd(m1, m2) | (a2 - a1) 时才有解
c = (c % m2 + m2) % m2; c /= d; m1 /= d; m2 /= d;
c = c * inv(m1 % m2, m2) % m2; //0逆元可任意值
M = m1*m2*d; A = (c *m1 %M *d %M +a1) % M; return 1;}//有解
```

4.12 Miller Rabin, Pollard Rho

- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356

```
mt19937 rng(123);
#define rand() LL(rng() & LLONG_MAX)
const int BASE[] = {2, 7, 61};//int(7,3e9)
//{2,325,9375,28178,450775,9780504,1795265022}LL(37)
struct miller_rabin {
bool check (const LL &M, const LL &base) {
  | LL a = M - 1;
  | while (~a & 1) a >>= 1;
  | LL w = power (base, a, M); // power should use mul
  | for (; a != M - 1 && w != 1 && w != M - 1; a <<= 1)
  |   | w = mul (w, w, M);
  | return w == M - 1 || (a & 1) == 1; }
bool solve (const LL &a) {//O((3 or 7) · log n · mul)
  | if (a < 4) return a > 1;
  | if (~a & 1) return false;
  | for (int i = 0; i < sizeof(BASE)/4 && BASE[i] < a; ++i)
  |   | if (!check (a, BASE[i])) return false;
  | return true; } };
miller_rabin is_prime;
LL get_factor (LL a, LL seed) {//O(n^1/4 · log n · mul)
  | LL x = rand () % (a - 1) + 1, y = x;
  | for (int head = 1, tail = 2; ; ) {
  |   | x = mul (x, x, a); x = (x + seed) % a;
  |   | if (x == y) return a;
  |   | LL ans = gcd (abs (x - y), a);
  |   | if (ans > 1 && ans < a) return ans;
  |   | if (++head == tail) { y = x; tail <<= 1; } } }
void factor (LL a, vector<LL> &d) {
  | if (a <= 1) return;
  | if (is_prime.solve (a)) d.push_back (a);
  | else {
  |   | LL f = a;
  |   | for (; f >= a; f = get_factor (a, rand() % (a - 1) +
  |       ↪ 1));
  |   | factor (a / f, d);
  |   | factor (f, d); } }
```

4.13 扩展卢卡斯

```
int l,a[33],p[33],P[33];
U fac(int k,LL n){// 求 n! mod pk^tk, 返回值 U{ 不包含 pk 的
  ↪ 值 ,pk 出现的次数 }
  | if (!n)return U{1,0};LL x=n/p[k],y=n/P[k],ans=1;int i;
  | if(y){// 求出循环节的答案
  |   | for(i=2;i<P[k];i++)if(i%p[k])ans=ans*i%P[k];
  |   | ans=Pw(ans,y,P[k]);
  | }for(i=y*P[k];i<=n;i++) if(i%p[k])ans=ans*i%M;// 求零散部
      ↪ 分
  | U z=fac(k,x);return U{ans*z.x%M,x+z.z};
}LL get(int k,LL n,LL m){// 求 C(n,m) mod pk^tk
  | U a=fac(k,n),b=fac(k,m),c=fac(k,n-m);// 分三部分求解
  | return Pw(p[k],a.z-b.z-c.z,P[k])*a.x%P[k]*
      ↪ inv(b.x,P[k])%P[k]*inv(c.x,P[k])%P[k];
}LL CRT(){// CRT 合并答案
  | LL d,w,y,x,ans=0;
  | fr(i,1,l)w=M/P[i],exgcd(w,P[i],x,y),
      ↪ ans=(ans+w*x%M*a[i])%M;
  | return (ans+M)%M;
}LL C(LL n,LL m){// 求 C(n,m)
  | fr(i,1,l)a[i]=get(i,n,m);
  | return CRT();
}LL exLucas(LL n,LL m,int M){
  | int jj=M,i; // 求 C(n,m)mod M,M=prod(pi^ki), 时间
      ↪ O(pi^kilg^2n)
  | for(i=2;i*i<=jj;i++)if(jj%i==0) for(p[+
      ↪ +l]=i,P[l]=1;jj%i==0;P[l]*=p[l])jj/=i;
  | if(jj>1)l++,p[l]=P[l]=jj;
  | return C(n,m);}
```

4.14 Cantor 康托展开

```
/*  康托 (Cantor) 展开：求全排列是第几个。  */
#include <bits/stdc++.h>
#define LL long long
using namespace std;
const int MAXN=2000005; const int MOD=998244353;
struct BIT { int n,b[MAXN];
  | int lowbit(int x){ return x&(-x); }
  | void change(int x,int y){ for(;x<=n;x+=lowbit(x))
      ↪ b[x]+=y; }
  | int sum(int x){ int s=0;
  |   | for(;x>0;x-=lowbit(x)) s+=b[x];
  |   |   | return s; }
}bit;
int a[MAXN],f[MAXN];
void solve() {
  | int n; scanf("%d", &n);
  | for(int i=1;i<=n;i++) scanf("%d", &a[i]);
  | f[0]=1; for(int i=1;i<=n;i++) f[i]=1LL*f[i-1]*i%MOD;
  | bit.n=n; int ans=1;
  | for(int i=1;i<n;i++) { bit.change(a[i],1);
      ↪ ans=(ans+1LL*f[n-i]*(a[i]-bit.sum(a[i]))%MOD)%MOD; }
  | printf("%d\n", ans);
}
int main() {
  | int T=1,cas=1;(void)(cas); // scanf("%d", &T);
  | while(T--) { // printf("Case #%d: ", cas++);
  |   | solve(); }
  | return 0;
}
```

4.15 Polya

```
/* Polya定理 本质不同个数=sigma(颜色个数^ 等价置换的划分个数)/
  ↪ (等价置换总个数)
例题：n条个点围成一圈，m种颜色。求染色旋转本质不同个数。
ans = 1/n Σ_{i=1}^{n} (m^gcd(n,i))
  = 1/n Σ_{d|n} (m^d φ(n/d)) ----- 莫比乌斯反演
*/
#include <bits/stdc++.h>
#define LL long long
using namespace std; const int MOD=1e9+7, MAXN=2000005;
int binpow(int x,int y,int m) {
  | int r=1%m; while(y) { if(y&1) r=1LL*r*x%m; x=1LL*x*x%m;
      ↪ y>>=1;
  | } return r; }
int phi(int x) {
  | int r=x;
  | for(int i=2;i*i<=x;i++) {
  |   | if(x%i==0) { r=r/i*(i-1); while(x%i==0) x/=i; }
  | } if(x>1) r=r/x*(x-1);
```

```
  | return r; }
void solve() {
  | int n,m; //n个点，m种颜色。
  | scanf("%d", &n); m=n; int ans=0;
  | for(int i=1;i*i<=n;i++) {
  |   | if(n%i==0) {
  |   |   | ans=(ans+1LL*binpow(m,i,MOD)*phi(n/i)%MOD)%MOD;
  |   |   | if(i*i!=n) {
  |   |   |   | ans=(ans+1LL*binpow(m,n/i,MOD)*phi(i)%MOD)%MOD;
  |   |   | } } }
  | ans=1LL*ans*binpow(n,MOD-2,MOD)%MOD;
  | printf("%d\n", ans); }
int main() {
  | int T=1,cas=1;(void)(cas);
  | scanf("%d", &T);
  | while(T--) { // printf("Case #%d: ", cas++);
  |   | solve(); } return 0;
}
```

4.16 BSGS 离散对数

```
LL inv(LL a,LL n){LL x,y;exgcd(a,n,x,y);return(x+n)%n;}
LL bsgs(LL a,LL b,LL n){// 在 (a,n)=1 时求最小的 x 使得 a^x
  ↪ mod n=b
  | LL m=sqrt(n+0.5),e=1,i;map<LL,LL>mp;mp[1]=0;
  | for(i=1;i<m;i++)if(!mp.count(e=e*a%n))mp[e]=i;
  | e=e*a%n;e=inv(e,n);// e=a^m, 求出其逆元后放到等式右边
  | for(i=0;i<m;b=b*e%n,i++)if(mp.count(b))return i*m+mp[b];
  | return -1;// 无解
}LL exbsgs(LL a,LL b,LL n){// 求最小的 x 使 a^x mod n=b
  | LL V,k=0,d,e=1;
  | for(;(d=gcd(a,n))!=1;){
  |   | if(b%d)return b==1?0:-1;// 如果 (a,n)=1, 要么 x=0&b=1,
  |     ↪ 要么无解
  |   | k++;n=n/d;b=b/d;e=e*a/d%n;
  |   | if(e==b)return k; }// 特判
  | V=bsgs(a,b*inv(e,n)%n,n);return ~V?V+k:V;}// 有解返回 V+k
```

4.17 FFT

```
struct cpx {
  | double x, y;
  | cpx(double a = 0, double b = 0) {
  |   | x = a, y = b;
  | }
  | cpx operator + (const cpx& t) const {
  |   | return cpx(x + t.x, y + t.y);
  | }
  | cpx operator - (const cpx& t) const {
  |   | return cpx(x - t.x, y - t.y);
  | }
  | cpx operator * (const cpx& t) const {
  |   | return cpx(x * t.x - y * t.y, x * t.y + y * t.x);
  | }
};

int n, m;
int limit;
cpx A[N];

void FFT(cpx A[], int type) {
  | vector<cpx> w(limit); w[0].x = 1;
  | for (int i = 0, j = 0; i < limit; i++) {
  |   | if (i > j) swap(A[i], A[j]);
  |   | for (int l = limit / 2; (j ^= l) < l; l /= 2);
  | }
  | for (int i = 1; i < limit; i *= 2) {
  |   | cpx wn(cos(pi / i), type * sin(pi / i));
  |   | for (int j = (i - 2) / 2; ~j; j--) w[j * 2 + 1] =
  |     ↪ (w[j * 2] = w[j]) * wn;
  |   | for (int j = 0; j < limit; j += i * 2) {
  |   |   | for (int k = j; k < i + j; k++) {
  |   |   |   | cpx x = A[k], y = A[k + i] * w[k - j];
  |   |   |   | A[k] = x + y, A[k + i] = x - y;
  |   |   | }
  |   | }
  | }
  | if (type != 1) for (int i = 0; i < limit; i++) A[i].x /=
  |   ↪ limit, A[i].y /= limit;
}

void oper() {
  | cin >> n >> m;
  | for (int i = 0; i <= n; i++) cin >> A[i].x;
  | for (int i = 0; i <= m; i++) cin >> A[i].y;
  | limit = 1; while (limit <= n + m) limit *= 2;
  | FFT(A, 1);
  | for (int i = 0; i < limit; i++) A[i] = A[i] * A[i];
  | FFT(A, -1);
  | for (int i = 0; i <= n + m; i++) cout << (int)(A[i].y /
  |   ↪ 2 + 0.5) << " ";
}
```

4.18 NTT

```
const int N = 4e5 + 10, mod = 4179340454199820289, G = 3;
int n, m;
int A[N], B[N], R[N], limit, L;
int qpow(int a, int b) {
  | int res = 1;
  | while (b) {
  |   | if (b & 1) res = (__int128)res * a % mod;
  |   | a = (__int128)a * a % mod;
  |   | b /= 2;
  | }
  | return res;
}
void NTT(int A[], int type) {
  | for (int i = 0; i < limit; i++) if (i < R[i]) swap(A[i],
  |   ↪ A[R[i]]);
  | for (int mid = 1; mid < limit; mid *= 2) {
  |   | int wn = qpow(G, (mod - 1) / (mid * 2));
  |   | if (type == -1) wn = qpow(wn, mod - 2);
  |   | for (int len = mid * 2, pos = 0; pos < limit; pos +=
  |   |   ↪ len) {
  |   |   | int w = 1;
  |   |   | for (int k = 0; k < mid; k++, w = (__int128)w * wn
  |   |   |   ↪ % mod) {
  |   |   |   | int x = A[pos + k], y = (__int128)w * A[pos +
  |   |   |   |   ↪ mid + k] % mod;
  |   |   |   | A[pos + k] = ((__int128)x + y) % mod;
  |   |   |   | A[pos + mid + k] = ((__int128)x - y + mod) %
  |   |   |   |   ↪ mod;
  |   |   | }
  |   | }
  | }
  | if (type == -1) {
  |   | int inv = qpow(limit, mod - 2);
  |   | for (int i = 0; i < limit; i++) A[i] = (__int128)A[i]
  |   |   ↪ * inv % mod;
  | }
}
void oper() {
  | cin >> n >> m;
  | for (int i = 0; i <= n; i++) cin >> A[i];
  | for (int i = 0; i <= m; i++) cin >> B[i];
  | for (limit = 1, L = 0; limit <= n + m; limit *= 2, L++);
  | for (int i = 0; i < limit; i++) R[i] = (R[i / 2] / 2) |
  |   ↪ ((i & 1) << (L - 1));
  | NTT(A, 1), NTT(B, 1);
  | for (int i = 0; i < limit; i++) A[i] = (__int128)A[i] *
  |   ↪ B[i] % mod;
  | NTT(A, -1);
  | for (int i = 0; i <= n + m; i++) cout << A[i] << " ";
}
```

5. DP

5.1 背包

```
// 复杂度: O(NV), O(NV), O(VΣ log_Ci)
/*01背包 */
void ZeroOnepark(int V/*背包容量*/,int val/*物品价值*/, int
  ↪ vol/*物品体积*/) {
  | for (int j = V; j >= vol; j--) {
  |   | dp[j] = max(dp[j], dp[j - vol] + val);
  | }
}
/*完全背包 */
void Completepark(int V/*背包容量*/,int val/*物品价值*/, int
  ↪ vol/*物品体积*/) {
  | for (int j = vol; j <= V; j++) {
  |   | dp[j] = max(dp[j], dp[j - vol] + val);
  | }
```

```
}
/*多重背包 */
void Multiplepark(int val/*物品价值*/, int vol/*物品体积*/,
  ↪ int amount/*物品数量*/) {
  | if (vol * amount >= v) {
  |   | Completepark(val, vol);
  | } else {
  |   | int k = 1;
  |   | while (k < amount) {
  |   |   | ZeroOnepark(k * val, k * vol);
  |   |   | amount -= k; k <<= 1;
  |   | }
  |   | if (amount > 0) {
  |   |   | ZeroOnepark(amount * val, amount * vol);
  |   | }
  | }
}
int main(){
  | memset(dp, 0, sizeof(dp));
  | for (int i = 1; i <= n; i++) {
  |   | Multiplepark(value[i], volume[i], num[i]);
  | }
}

// 单调队列优化多重背包 O(NV)
int solve(int volume[],int value[],int n,int V){
  | pair<int,int> q[maxn];int head=1,tail=0;
  | for (int i = 1; i <= n; i++) {
  |   | for (int j = 0; j < volume[i]; j++) {
  |   |   | q[head = tail = 1] = make_pair(dp[j], 0);
  |   |   | for (int k = j + volume[i]; k <= V; k +=
  |   |   |   ↪ volume[i]) {
  |   |   |   | int a = k / volume[i], t = dp[k] - a *
  |   |   |   |   ↪ value[i];
  |   |   |   | while (head <= tail && q[tail].first <= t)
  |   |   |   |   ↪ tail--;
  |   |   |   | q[++tail] = make_pair(t, a);
  |   |   |   | while (head <= tail && q[head].second + num[i]
  |   |   |   |   ↪ < a) head++;
  |   |   |   | dp[k] = max(dp[k], q[head].first + a *
  |   |   |   |   ↪ value[i]);
  |   |   | }
  |   | }
  | }
}
```

5.2 树形依赖背包

```
/* 定义dp[u][i] 表示，以u为根节点的子树中保留i条树枝所获得的最
  ↪ 大权值
 * 则转移方程为dp[u][i]=max(dp[u][i],dp[left[u]]
  ↪ [i-j-1]+left[u].w+dp[right[u]][j-1]+right[u].w)
 * 表示u的右儿子保留j-1条边，u的左儿子保留剩下的i-j-1条边，此时
  ↪ 总共有i-2条边，还要加上u-left[u],u-right[u] 这两条边。
 * 另外一种转移状态dp[u][i]=max(dp[u][i],dp[u][i-j]+dp[v]
  ↪ [j-1]+w)
 * 跟上面类似，只不过将u与其中一个儿子节点的状态放在一起。此时
  ↪ 需要倒序枚举i来保证只选择一次（类似01背包）。
 * 没有访问过的子树不会保存在dp[u][i] 中，所以不会出现重复计算
  ↪ 的情况。
 */
void dfs(int u,int p=-1){
  | sz[u]=1;
  | for(auto t:g[u]){
  |   | int &v=t.first,&w=t.second;
  |   | if(v==p) continue;
  |   | dfs(v,u);sz[u]+=sz[v];
  |   | for(int i=min(q,sz[u]);i>=1;i--){
  |   |   | for(int j=1;j<=min(sz[v],i);j++){
  |   |   |   | dp[u][i]=max(dp[u][i],dp[u][i-j]+dp[v][j-1]+w);
  |   |   | }
  |   | }
  | }
}
```

5.3 最长上升子序列

```
int arr[maxn], n;
template<class Cmp>
int LIS (Cmp cmp) {
  | static int m, end[maxn];
  | m = 0;
  | for (int i=0; i<n; i++) {
  |   | int pos = lower_bound(end, end+m, arr[i], cmp)-end;
  |   | end[pos] = arr[i], m += pos==m;
  | }
  | return m;
}
/*********
  | std::cout << LIS(std::less<int>()) << std::endl;
  |   ↪ //严格上升
  | std::cout << LIS(std::less_equal<int>()) << std::endl;
  |   ↪ //非严格上升
  | std::cout << LIS(std::greater<int>()) << std::endl;
  |   ↪ //严格下降
  | std::cout << LIS(std::greater_equal<int>()) <<
  |   ↪ std::endl;//非严格下降
**********/

// 求最长不下降子序列的长度 O(nlog(n))
int solve(int n,int a[]){
  | int dp[maxn]; memset(dp, INF, sizeof(dp));
  | for(int i=1;i<=n;i++){
  |   | *upper_bound(dp+1,dp+n+1,a[i])=a[i];
  | }
  | int ans=lower_bound(dp+1,dp+n+1,INF)-dp;
  | return ans;
}
// 求最长上升子序列 O(nlog(n))
vector<int> solve(int n,int p[]){
  | vector<int>f;f.push_back(0);
  | int a[n+5]; /* 记录每个数的前一个数是多少 */
  | for(int i=1;i<=n;i++){
  |   | auto it=lower_bound(f.begin(),f.end(),p[i]);
  |   | if(it==f.end()){
  |   |   | a[p[i]]=f.back();
  |   |   | f.push_back(p[i]);
  |   | }
  |   | else{
  |   |   | *it=p[i];it--;
  |   |   | a[p[i]]=*it;
  |   | }
  | }
  | vector<int>vis; /*最长上升子序列 */
  | for(int t=f.back();t;t=a[t]) vis.push_back(t);
  | reverse(vis.begin(),vis.end());
  | return vis;
}
int a[maxn];/*原始数组 */
memset(f,INF,sizeof(int)*(n+2));
for(int i=1;i<=n;i++){ //nlog(n) 求最长上升子序列
  | int j = lower_bound(f + 1, f + n + 1, a[i]) - f;
  | End[i] = j; //以i为终点的最长严格上升子序列长度为j
  | f[j] = a[i];
}
memset(g,0,sizeof(int)*(n+2));
for (int i = n; i >= 1; i--) {
  | int j=lower_bound(g+1,g+n+1,-a[i])-g;
  | Start1[i]=j; //以i为起点的最长严格上升子序列的长度为j
  | g[j]=-a[i];
}
memset(g,INF,sizeof(int)*(n+2));
reverse(a+1,a+n+1);
for(int i=1;i<=n;i++){ //nlog(n) 求最长上升子序列
  | int j = lower_bound(g + 1, g + n + 1, a[i]) - g;
  | Start[n-i+1] = j; //以i为起点的最长严格下降子序列长度为j
  | g[j] = a[i];
}
memset(f,0,sizeof(int)*(n+2));
for (int i = n; i >= 1; i--) {
  | int j = lower_bound(f + 1, f + n + 1, -a[i]) - f;
  | End1[n-i+1] = j; //以i为终点的最长严格下降子序列的长度为j
  | f[j] = -a[i];
}
```

5.4 最长公共子序列

```
int n,a[MAXN],len; map <int,int> mp,rmp;
int main(){ scanf("%d", &n);
  | for(int i=1,t;i<=n;i++){ scanf("%d", &t);
  |   | mp.insert(make_pair(t,i)); }
  | a[len=0]=0; map <int,int>::iterator ite;
  | for(int i=1,t,v;i<=n;i++){
  |   | scanf("%d",&t); ite=mp.find(t); v=ite->second;
```

```
    if(v>a[len])
      a[++len]=v;
    else{
      int *p=upper_bound(a+1,a+len+1,v,less<int>());
      *p=v;
  } } printf("%d\n",len);  return 0;}
```

5.5 区间dp

```
for (int x = 0; x < n; x++){//枚举长度
  for (int i = 1; i + x <= n; i++){//枚举起点
    dp[i][i] = 1;
    int j = x + i;//终点
    dp[i][j] = dp[i + 1][j] + 1;
    for (int k = i + 1; k <= j; k++) {
      if (a[i] == a[k])
        dp[i][j] = min(dp[i][j], dp[i][k - 1] + dp[k +
            ↪ 1][j]);
    }
  }
}
```

5.6 数位dp

给定一段区间，找出一个数满足，所有位中的最大值和最小值相差最小

```
const int N = 20;
int f[20][10][10][2][2], nums[N];
int len, a, b;
int dfs(int pos, int lim, int mx, int mn, int lead, int
  ↪ tar) {
  if (!pos) return mx - mn == tar;
  int& v = f[pos][mx][mn][lim][lead];
  if (~v) return v;
  int res = 0, upper = lim ? nums[pos] : 9;
  for (int i = 0; i <= upper; i++) {
    int nl = lead & !i;
    int nmx = mx, nmn = mn;
    if (!nl) nmx = max(mx, i), nmn = min(mn, i);
    res += dfs(pos - 1, lim & i == upper, nmx, nmn, nl,
        ↪ tar);
  }
  return v = res;
}
int dp(int n, int tar) {
  memset(f, -1, sizeof f);
  len = 0;
  while (n) nums[++len] = n % 10, n /= 10;
  return dfs(len, 1, 0, 9, 1, tar);
}
void solve() {
  int l, r;
  cin >> l >> r;
  for (int i = 0; i <= 9; i++) {
    int rval = dp(r, i), lval = dp(l - 1, i);
    if (rval > lval) {
      int ll = l, rr = r;
      while (ll < rr) {
        int mid = ll + rr >> 1;
        if (dp(mid, i) - lval > 0) {
          rr = mid;
        } else {
          ll = mid + 1;
        }
      }
      cout << ll << '\n';
      return;
    }
  }
}
```

6. String

6.1 String Hash

```
struct Hash {
  int n, base; string s;
  vector<uint64_t> info, pow;
  Hash(const string &s, int base = 13331):
    n(s.size()), base(base), s(s), info(n + 1, 0), pow(n
        ↪ + 1, 1) {
    for (int i = 1; i <= n; ++i) {
      pow[i] = pow[i - 1] * base;
    }
    for (int i = 1; i <= n; ++i) {
      info[i] = info[i - 1] * base + s[i - 1] + base;
    }
  }
  uint64_t subhash(int pos, int len = 2E9) {
    assert(pos >= 0 && pos < n && len > 0);
    len = min(len, n - pos);
    int l = pos + 1, r = pos + len;
    auto res = info[r] - info[l - 1] * pow[r - l + 1];
    return res;
  }
};
```

```
const int N = 1e5 + 10, P = 131;
int n, m;
char A[N];
ull h[N], p[N];
ull getstr(int l, int r) {
  return h[r] - h[l - 1] * p[r - l + 1];
}
void oper() {
  cin >> n >> m >> A + 1;
  p[0] = 1;
  for (int i = 1; i <= n; i++) {
    h[i] = h[i - 1] * P + A[i];
    p[i] = p[i - 1] * P;
  }
  int l, r, a, b;
  for (int i = 1; i <= m; i++) {
    cin >> l >> r >> a >> b;
    if (getstr(l, r) == getstr(a, b)) cout << "Yes" <<
        ↪ endl;
    else cout << "No" << endl;
  }
}
```

6.2 最小表示法

```
int min_pos(vector<int> a) {
  int n = a.size(), i = 0, j = 1, k = 0;
  while (i < n && j < n && k < n) {
    auto u = a[(i + k) % n]; auto v = a[(j + k) % n];
    int t = u > v ? 1 : (u < v ? -1 : 0);
    if (t == 0) k++; else {
      if (t > 0) i += k + 1; else j += k + 1;
      if (i == j) j++;
      k = 0; } } return min(i, j); }
```

6.3 Manacher

```
// n为串长，回文半径输出到p数组中，数组要开串长的两倍
void manacher(const char *t, int n) {
  static char s[maxn * 2];
  for (int i = n; i; i--) s[i * 2] = t[i];
  for (int i = 0; i <= n; i++) s[i * 2 + 1] = '#';
  s[0] = '$'; s[(n + 1) * 2] = '\0'; n = n * 2 + 1;
  int mx = 0, j = 0;
  for (int i = 1; i <= n; i++) {
    p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
    while (s[i - p[i]] == s[i + p[i]]) p[i]++;
    if (i + p[i] > mx) { mx = i + p[i]; j = i; } } }
```

```
int n;
char A[N], B[N];
int p[N];
void init() {
  int idx = 0;
  B[++idx] = '$', B[++idx] = '#';
  for (int i = 1; i <= n; i++) B[++idx] = A[i], B[++idx] =
      ↪ '#';
  B[++idx] = '^';
  n = idx;
}
void manacher() {
  int mr = -1, mid;
  for (int i = 1; i <= n; i++) {
    if (i < mr) p[i] = min(p[mid * 2 - i], mr - i);
    else p[i] = 1;
    while (B[i - p[i]] == B[i + p[i]]) p[i]++;
```

```
    | | if (i + p[i] > mr) {
    | | | mr = i + p[i];
    | | | mid = i;
    | | }
    | }
}
void oper() {
    | cin >> A + 1; n = strlen(A + 1);
    | init(); manacher();
    | int ans = 0;
    | for (int i = 1; i <= n; i++) ans = max(ans, p[i] - 1);
    | cout << ans << endl;
}
```

6.4 KMP, exKMP

```
void kmp(const int *s, int n) {
    | fail[0] = fail[1] = 0;
    | for (int i = 1; i < n; i++) { int j = fail[i];
    | | while (j && s[i + 1] != s[j + 1]) j = fail[j];
    | | if (s[i + 1] == s[j + 1]) fail[i + 1] = j + 1;
    | | else fail[i + 1] = 0; } }
void exkmp(const char *s, int *a, int n) { // 0-based
    | int l = 0, r = 0; a[0] = n;
    | for (int i = 1; i <= n; i++) {
    | | a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
    | | while (i+a[i] < n && s[a[i]] == s[i+a[i]]) a[i]++;
    | | if (i + a[i] - 1 > r) {l = i; r = i + a[i] - 1;}}}
```

```
const int N = 1e6 + 10, M = 1e5 + 10;
int n, m, ne[M];
char S[N], P[M];
void oper() {
    | cin >> m >> P + 1 >> n >> S + 1;
    | for (int i = 2, j = 0; i <= m; i++) {
    | | while (j && P[i] != P[j + 1]) j = ne[j];
    | | if (P[i] == P[j + 1]) j++;
    | | ne[i] = j;
    | }
    | for (int i = 1, j = 0; i <= n; i++) {
    | | while (j && S[i] != P[j + 1]) j = ne[j];
    | | if (S[i] == P[j + 1]) j++;
    | | if (j == m) {
    | | | cout << i - m << " ";
    | | | j = ne[j];
    | | }
    | }
}
```

6.5 Trie

```
int son[N][26], cnt[N], idx;
// 0号点既是根节点，又是空节点
// son[][] 存储树中每个节点的子节点
// cnt[] 存储以每个节点结尾的单词数量
// 插入一个字符串
void insert(char *str) {
  int p = 0;
  for (int i = 0; str[i]; i++) {
    int u = str[i] - 'a';
    if (!son[p][u])
      son[p][u] = ++idx;
    p = son[p][u];
  }
  cnt[p]++;
}
// 查询字符串出现的次数
int query(char *str) {
  int p = 0;
  for (int i = 0; str[i]; i++) {
    int u = str[i] - 'a';
    if (!son[p][u])
      return 0;
    p = son[p][u];
  }
  return cnt[p];
}
```

```
struct Info {
    | int cnt_end;
    | int prefix;
    | Info() : cnt_end(0), prefix(0) {}
};
void apply(Info &x) { ++x.cnt_end; }
int query(Info &x) { return x.prefix; }
template <int LETTER = 'a', int ALPHABET = 26, typename T =
  ↪ Info>
struct Trie {
    | std::vector<std::vector<int>> next;
    | std::vector<T> info;
    | Trie() : next(1, std::vector<int>(ALPHABET, -1)),
      ↪ info(1) {}
    | void insert(const std::string &s) {
    | | int n = s.size(), p = 0;
    | | for (int i = 0; i < n; ++i) {
    | | | int c = s[i] - LETTER;
    | | | if (next[p][c] == -1) {
    | | | | next[p][c] = next.size();
    | | | | next.emplace_back(std::vector<int>(ALPHABET,
                ↪ -1));
    | | | | info.emplace_back(T());
    | | | }
    | | | p = next[p][c];
    | | | info[p].prefix += 1;
    | | }
    | | apply(info[p]);
    | }
    | int query(const std::string s) {
    | | int n = s.size(), p = 0;
    | | for (int i = 0; i < n; ++i) {
    | | | int c = s[i] - LETTER;
    | | | if (next[p][c] == -1) {
    | | | | return 0;
    | | | }
    | | | p = next[p][c];
    | | }
    | | return ::query(info[p]);
    | }
};
```

6.6 AC 自动机

```
int ch[maxn][26], fail[maxn], q[maxn], sum[maxn], cnt = 0;
int insert(const char *c) { int x = 0; while (*c) {
    | if (!ch[x][*c - 'a']) ch[x][*c - 'a'] = ++cnt;
    | x = ch[x][*c++ - 'a']; } return x; }
void getfail() { int x, head = 0, tail = 0;
    | for (int c = 0; c < 26; c++) if (ch[0][c])
    | | | q[tail++] = ch[0][c];
    | while (head != tail) { x = q[head++];
    | | for (int c = 0; c < 26; c++) { if (ch[x][c]) {
    | | | | fail[ch[x][c]] = ch[fail[x]][c];
    | | | | q[tail++] = ch[x][c];
    | | | } else ch[x][c] = ch[fail[x]][c]; } } }
```

```
struct AhoCorasick {
    | static constexpr int ALPHABET = 26;
    | struct Node {
    | | int len;
    | | int link;
    | | std::array<int, ALPHABET> next;
    | | Node() : link{}, next{} {}
    | };
    | std::vector<Node> t;
    | AhoCorasick() {
    | | init();
    | }
    | void init() {
    | | t.assign(2, Node());
    | | t[0].next.fill(1);
    | | t[0].len = -1;
    | }
    | int newNode() {
    | | t.emplace_back();
    | | return t.size() - 1;
    | }
    | int add(const std::vector<int> &a) {
    | | int p = 1;
    | | for (auto x : a) {
    | | | if (t[p].next[x] == 0) {
    | | | | t[p].next[x] = newNode();
    | | | | t[t[p].next[x]].len = t[p].len + 1;
    | | | }
```

```
        p = t[p].next[x];
    }
    return p;
  }
  int add(const std::string &a, char offset = 'a') {
    std::vector<int> b(a.size());
    for (int i = 0; i < (int)a.size(); i++) {
      b[i] = a[i] - offset;
    }
    return add(b);
  }
  void work() {
    std::queue<int> q;
    q.push(1);
    while (!q.empty()) {
      int x = q.front();
      q.pop();
      for (int i = 0; i < ALPHABET; i++) {
        if (t[x].next[i] == 0) {
          t[x].next[i] = t[t[x].link].next[i];
        } else {
          t[t[x].next[i]].link = t[t[x].link].next[i];
          q.push(t[x].next[i]);
        }
      }
    }
  }
  int next(int p, int x) {
    return t[p].next[x];
  }
  int next(int p, char c, char offset = 'a') {
    return next(p, c - 'a');
  }
  int link(int p) {
    return t[p].link;
  }
  int len(int p) {
    return t[p].len;
  }
  int size() {
    return t.size();
  }
};
```

6.7 后缀数组

```
int n, m;
char A[N];
int sa[N], x[N], y[N], cnt[N], rk[N], height[N];
void get_sa() {
  for (int i = 1; i <= n; i++) cnt[x[i] = A[i]]++;
  for (int i = 2; i <= m; i++) cnt[i] += cnt[i - 1];
  for (int i = n; i >= 1; i--) sa[cnt[x[i]]--] = i;
  for (int k = 1; k <= n; k *= 2) {
    int num = 0;
    for (int i = n - k + 1; i <= n; i++) y[++num] = i;
    for (int i = 1; i <= n; i++)
      if (sa[i] > k) y[++num] = sa[i] - k;
    for (int i = 1; i <= m; i++) cnt[i] = 0;
    for (int i = 1; i <= n; i++) cnt[x[i]]++;
    for (int i = 2; i <= m; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i >= 1; i--) sa[cnt[x[y[i]]]--] =
        ↪ y[i], y[i] = x[i];
    x[sa[1]] = 1, num = 1;
    for (int i = 2; i <= n; i++) {
      if (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] ==
          ↪ y[sa[i - 1] + k])
        x[sa[i]] = num;
      else
        x[sa[i]] = ++num;
    }
    if (num == n) break;
    m = num;
  }
}
void get_height() {
  for (int i = 1; i <= n; i++) rk[sa[i]] = i;
  for (int i = 1, k = 0; i <= n; i++) {
    if (rk[i] == 1) continue;
    if (k) k--;
    int j = sa[rk[i] - 1];
    while (i + k <= n && j + k <= n && A[i + k] == A[j +
        ↪ k]) k++;
    height[rk[i]] = k;
  }
}
void oper() {
  cin >> A + 1;
  n = strlen(A + 1); m = 128;
  get_sa(); get_height();
  for (int i = 1; i <= n; i++) cout << sa[i] << " \n"[i ==
      ↪ n];
  for (int i = 1; i <= n; i++) cout << height[i] << "
      ↪ \n"[i == n];
}
```

6.8 后缀自动机

```
struct SuffixAutomaton {
  static constexpr int ALPHABET_SIZE = 26, N = 1e6;
  struct Node {
    int len;
    int link;
    int next[ALPHABET_SIZE];
    Node() : len(0), link(0), next{} {}
  } t[2 * N];
  int cntNodes;
  SuffixAutomaton() {
    cntNodes = 1;
    std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
    t[0].len = -1;
  }
  int extend(int p, int c) {
    if (t[p].next[c]) {
      int q = t[p].next[c];
      if (t[q].len == t[p].len + 1) return q;
      int r = ++cntNodes;
      t[r].len = t[p].len + 1;
      t[r].link = t[q].link;
      std::copy(t[q].next, t[q].next + ALPHABET_SIZE,
          ↪ t[r].next);
      t[q].link = r;
      while (t[p].next[c] == q) {
        t[p].next[c] = r;
        p = t[p].link;
      }
      return r;
    }
    int cur = ++cntNodes;
    t[cur].len = t[p].len + 1;
    while (!t[p].next[c]) {
      t[p].next[c] = cur;
      p = t[p].link;
    }
    t[cur].link = extend(p, c);
    return cur;
  }
};
```

```
int last, val[maxn], par[maxn], go[maxn][26], sam_cnt;
void extend(int c) { // 结点数要开成串长的两倍
  int p = last, np = ++sam_cnt; val[np] = val[p] + 1;
  while (p && !go[p][c]) { go[p][c] = np; p = par[p]; }
  if (!p) par[np] = 1; else { int q = go[p][c];
    if (val[q] == val[p] + 1) par[np] = q;
    else { int nq = ++sam_cnt; val[nq] = val[p] + 1;
      memcpy(go[nq], go[q], sizeof(go[q]));
      par[nq] = par[q]; par[np] = par[q] = nq;
      while (p && go[p][c] == q) { go[p][c] = nq;
        p = par[p]; } } } last = np; }
int c[maxn], q[maxn]; int main() { last = sam_cnt = 1;
  for (int i = 1; i <= sam_cnt; i++) c[val[i] + 1]++;
  for (int i = 1; i <= n; i++) c[i] += c[i - 1];
  for (int i = 1; i <= sam_cnt; i++) q[++c[val[i]]] = i;
  return 0; }
```

6.9 回文自动机

```
#define int long long
const int N = 5e5 + 10;
string s;
int n, lst, len[N];
int now, tot = 1, fail[N], cnt[N], T[N][26];
int getfail(int pre, int p) {
```

```
  while (p - len[pre] - 1 <= 0 || s[p - len[pre] - 1] !=
    ↪ s[p]) pre = fail[pre];
  return pre;
}
int insert(char c, int id) {
  int p = getfail(now, id);
  if (!T[p][c - 'a']) {
  | fail[++tot] = T[getfail(fail[p], id)][c - 'a'];
  | T[p][c - 'a'] = tot;
  | len[tot] = len[p] + 2;
  | cnt[tot] = cnt[fail[tot]] + 1;
  }
  return cnt[now = T[p][c - 'a']];
}
void oper() {
  cin >> s;
  s = "?" + s;
  n = s.size() - 1;
  fail[0] = 1, len[1] = -1;
  for (int i = 1; i <= n; i++) {
  | if (i > 1) s[i] = (s[i] - 'a' + lst) % 26 + 'a';
  | cout << (lst = insert(s[i], i)) << " ";
  }
}
```

6.10 SAMSA & 后缀树

```
bool vis[maxn * 2]; char s[maxn];
int id[maxn * 2], ch[maxn * 2][26], height[maxn], tim = 0;
void dfs(int x) {
  | if (id[x]) { height[tim++] = val[last];
  |   | sa[tim] = id[x]; last = x; }
    for (int c = 0; c < 26; c++)
  |   | if (ch[x][c]) dfs(ch[x][c]);
  | last = par[x]; }
int main() { last = ++cnt; scanf("%s", s + 1);
    int n = strlen(s + 1); for (int i = n; i; i--) {
  |   | expand(s[i] - 'a'); id[last] = i; }
  | vis[1] = true; for (int i = 1; i <= cnt; i++) if (id[i])
  |   |   | for (int x = i,pos = n; x && !vis[x]; x = par[x]){
  |   |   |   | vis[x] = true; pos -= val[x] - val[par[x]];
  |   |   |   | ch[par[x]][s[pos + 1] - 'a'] = x; }
  | dfs(1); for (int i = 1; i <= n; i++)
  |   | printf("%d%c", sa[i], i < n ? ' ' : '\n');
  | for (int i = 1; i < n; i++) printf("%d%c", height[i],
        i < n ? ' ' : '\n'); return 0; }
```

6.11 回文树

```
int val[maxn], par[maxn], go[maxn][26], last, cnt;
char s[maxn];
void extend(int n) { int p = last, c = s[n] - 'a';
  | while (s[n - val[p] - 1] != s[n]) p = par[p];
  | if (!go[p][c]) { int q = ++cnt, now = p;
  |   | val[q] = val[p] + 2;
  |   | do p = par[p];
  |   | while (s[n - val[p] - 1] != s[n]);
  |   | par[q] = go[p][c]; last = go[now][c] = q;
  | } else last = go[p][c]; }
int main() { par[0] = cnt = 1; val[1] = -1; }
```

6.12 字符串结论

6.12.1 双回文串

如果 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|, x_2, y_1, y_2, z_1$ 是回文串, 则 x_1 和 z_2 也是回文串.

6.12.2 Border 和周期

如果 r 是 S 的一个border, 则 $|S| - r$ 是 S 的一个周期.
如果 p 和 q 都是 S 的周期, 且满足 $p + q \le |S| + gcd(p, q)$, 则 $gcd(p, q)$ 也是一个周期.

6.12.3 字符串匹配与Border

若字符串 S, T 满足 $2|S| \ge |T|$, 则 S 在 T 中所有匹配位置成等差数列.
若 S 的匹配次数大于2, 则等差数列的周期恰好等于 S 的最小周期.

6.12.4 Border 的结构

字符串 S 的所有不小于 $|S|/2$ 的border长度组成一个等差数列.
字符串 S 的所有 border 按长度排序后可分成 $O(\log |S|)$ 段, 每段是一个等差数列.

6.12.5 回文串Border

回文串长度为 t 的后缀是一个回文后缀, 等价于 t 是该串的border. 因此回文后缀的长度也可以划分成 $O(\log |S|)$ 段.

6.12.6 子串最小后缀

设 $s[p..n]$ 是 $s[i..n]$, $(l \le i \le r)$ 中最小者, 则minsuf(l, r) 等于 $s[p..r]$ 的最短非空 border. minsuf(l, r) = min{$s[p..r]$, minsuf(r − 2^k + 1, r)}, $(2^k < rl + 1 \le 2^{k+1})$.

6.12.7 子串最大后缀

从左往右扫, 用set维护后缀的字典序递减的单调队列, 并在对应时刻添加”小于事件” 点以便在之后修改队列; 查询直接在set里lower_bound.

7. Other

7.1 几何公式

7.1.1 阿波罗尼茨圆

所有关于两点 A, B 满足 $PA/PB = k$ 且不等于 1 的点 P 的轨迹是一个圆.

硬币游戏

两两相切的圆 r1, r2, r3, 求与他们都相切的圆 r4 分母取负号, 答案再取绝对值, 为外切圆半径分母取正号为内切圆半径 $r_4^{\pm} = \frac{r_1r_2r_3}{r_1r_2+r_1r_3+r_2r_3\pm2\sqrt{r_1r_2r_3(r_1+r_2+r_3)}}$

7.1.2 圆幂 圆反演 根轴

圆幂: 半径为 R 的圆 O, 任意一点 P 到 O 的幂为 $h = OP^2 - R^2$
圆幂定理: 过 P 的直线交圆在 A 和 B 两点, 则 $PA \cdot PB = |h|$
根轴: 到两圆等幂点的轨迹是一条垂直于连心线的直线
反演: 已知一圆 C, 圆心为 O, 半径为 r, 如果 P 与 P' 在过圆心 O 的直线上, 且 $OP \cdot OP' = r^2$, 则称 P 与 P' 关于 O 互为反演. 一般 C 取单位圆.
反演的性质:
不过反演中心的直线反形是过反演中心的圆, 反之亦然.
不过反演中心的圆, 它的反形是一个不过反演中心的圆.
两条直线在交点 A 的夹角, 等于它们的反形在相应点 A' 的夹角, 但方向相反.
两个相交圆周在交点 A 的夹角等于它们的反形在相应点 A' 的夹角, 但方向相反.
直线和圆周在交点 A 的夹角等于它们的反演图形在相应点 A' 的夹角, 但方向相反.
正交圆反形也正交. 相切圆反形也相切, 当切点为反演中心时, 反形为两条平行线.

7.1.3 球面基础

球面距离: 连接球面两点的大圆劣弧 (所有曲线中最短)
球面角: 球面两个大圆弧所在半平面形成的二面角
球面凸多边形: 把一个球面多边形任意一边向两方无限延长成大圆, 其余边都在此大圆的同旁.
球面角盈 E: 球面凸n边形的内角和与 $(n-2)\pi$ 的差
离北极夹角 θ, 距离 h 的球冠: $S = 2\pi Rh = 2\pi R^2(1 - \cos\theta)$, $V = \frac{\pi h^2}{3}(3R - h)$
球面凸n边形面积: $S = ER^2$

7.1.4 Heron's Formula

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$
$$p = \frac{a+b+c}{2}$$

7.1.5 四面体内接球球心

假设 s_i 是第 i 个顶点相对面的面积, 则有

$$\begin{cases} x = \frac{s_1x_1 + s_2x_2 + s_3x_3 + s_4x_4}{s_1 + s_2 + s_3 + s_4} \\ y = \frac{s_1y_1 + s_2y_2 + s_3y_3 + s_4y_4}{s_1 + s_2 + s_3 + s_4} \\ z = \frac{s_1z_1 + s_2z_2 + s_3z_3 + s_4z_4}{s_1 + s_2 + s_3 + s_4} \end{cases}$$

体积可以使用 1/6 混合积求, 内接球半径为

$$r = \frac{3V}{s_1 + s_2 + s_3 + s_4}$$

7.1.6 三角形内心

$$\vec{I} = \frac{a\vec{A} + b\vec{B} + c\vec{C}}{a+b+c}$$

7.1.7 三角形外心

$$\vec{O} = \frac{\vec{A} + \vec{B} - \frac{\vec{BC}\cdot\vec{CA}}{\vec{AB}\times\vec{BC}}\vec{AB}^T}{2}$$

7.1.8 三角形垂心

$$\vec{H} = 3\vec{G} - 2\vec{O}$$

7.1.9 三角形偏心

$$\frac{-a\vec{A} + b\vec{B} + c\vec{C}}{-a+b+c}$$

内角的平分线和对边的两个外角平分线交点, 外切圆圆心. 剩余两点的同理.

7.1.10 三角形内接外接圆半径

$$r = \frac{2S}{a+b+c}, \quad R = \frac{abc}{4S}$$

7.1.11 Pick's Theorem 格点多边形面积

$S = I + \frac{B}{2} - 1$. I 内部点, B 边界点。

7.1.12 Euler's Formula 多面体与平面图的点、边、面

For convex polyhedron: $V - E + F = 2$.
For planar graph: $|F| = |E| - |V| + n + 1$, n : #(connected components).

7.2 三角公式

$$\sin(a \pm b) = \sin a\cos b \pm \cos a\sin b$$
$$\cos(a \pm b) = \cos a\cos b \mp \sin a\sin b$$
$$\tan(a \pm b) = \frac{\tan(a) \pm \tan(b)}{1 \mp \tan(a)\tan(b)}$$
$$\tan(a) \pm \tan(b) = \frac{\sin(a \pm b)}{\cos(a)\cos(b)}$$

$$\sin(a)+\sin(b)=2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\sin(a)-\sin(b)=2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\cos(a)+\cos(b)=2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\cos(a)-\cos(b)=-2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$$

$$\sin(na)=n\cos^{n-1}a\sin a-\binom{n}{3}\cos^{n-3}a\sin^3 a+\binom{n}{5}\cos^{n-5}a\sin^5 a-\dots$$

$$\cos(na)=\cos^n a-\binom{n}{2}\cos^{n-2}a\sin^2 a+\binom{n}{4}\cos^{n-4}a\sin^4 a-\dots$$

7.2.1 超球坐标系

$$\begin{aligned}
x_1 &= r\cos(\phi_1)\\
x_2 &= r\sin(\phi_1)\cos(\phi_2)\\
&\cdots\\
x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1})\\
x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1})\\
\phi_{n-1} &\in [0,2\pi]\\
\forall i=1..n-1\phi_i &\in [0,\pi]
\end{aligned}$$

7.2.2 三维旋转公式

绕着 $(0,0,0)-(ux,uy,uz)$ 旋转 θ, (ux,uy,uz) 是单位向量

$$R=\begin{matrix}
\cos\theta+u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+u_y\sin\theta\\
u_yu_x(1-\cos\theta)+u_z\sin\theta & \cos\theta+u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta)-u_x\sin\theta\\
u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & \cos\theta+u_z^2(1-\cos\theta)
\end{matrix}.$$

$$\begin{bmatrix}x'\\y'\\z'\end{bmatrix}=R\begin{bmatrix}x\\y\\z\end{bmatrix}$$

7.2.3 立体角公式

ϕ：二面角

$$\Omega=(\phi_{ab}+\phi_{bc}+\phi_{ac})\text{ rad}-\pi\text{ sr}$$

$$\tan\left(\frac{1}{2}\Omega/\text{rad}\right)=\frac{\left|\vec{a}\,\vec{b}\,\vec{c}\right|}{abc+\left(\vec{a}\cdot\vec{b}\right)c+\left(\vec{a}\cdot\vec{c}\right)b+\left(\vec{b}\cdot\vec{c}\right)a}$$

$$\theta_s=\frac{\theta_a+\theta_b+\theta_c}{2}$$

7.2.4 常用体积公式

- 棱锥 Pyramid $V=\frac{1}{3}Sh$.
- 球 Sphere $V=\frac{4}{3}\pi R^3$.
- 棱台 Frustum $V=\frac{1}{3}h(S_1+\sqrt{S_1S_2}+S_2)$.
- 椭球 Ellipsoid $V=\frac{4}{3}\pi abc$.

7.2.5 扇形与圆弧重心

扇形重心与圆心距离为 $\frac{4r\sin(\theta/2)}{3\theta}$, 圆弧重心与圆心距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$.

7.2.6 高维球体积

$$V_2=\pi R^2,\ S_2=2\pi R$$
$$V_3=\frac{4}{3}\pi R^3,\ S_3=4\pi R^2$$
$$V_4=\frac{1}{2}\pi^2R^4,\ S_4=2\pi^2R^3$$
$$\text{Generally, }V_n=\frac{2\pi}{n}V_{n-2},\ S_{n-1}=\frac{2\pi}{n-2}S_{n-3}$$
$$\text{Where, }S_0=2,\ V_1=2,\ S_1=2\pi,\ V_2=\pi$$

7.3 计算几何基础模板

```cpp
namespace Geometry {
const double pi = acos(-1);
const double eps = 1e-8;
// 点与向量
struct Point {
  double x, y;
  Point(double x = 0, double y = 0) : x(x), y(y) {}
  bool operator==(const Point a) const {
    return (fabs(x - a.x) <= eps && fabs(y - a.y) <= eps);
  }
};
typedef Point Vector;
Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator-(Vector A, Vector B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / p); }
int sign(double x) {  // 符号函数
  if (fabs(x) < eps) return 0;
  if (x < 0) return -1;
  return 1;
}
int cmp(double x, double y) {  // 比较函数
  if (fabs(x - y) < eps) return 0;
  if (x < y) return -1;
  return 1;
}
double dot(Point a, Point b) {  // 向量点积
  return a.x * b.x + a.y * b.y;
}
double cross(Point a, Point b) {  // 向量叉积
  return a.x * b.y - b.x * a.y;
}
double get_length(Point a) {  // 求向量模长
  return sqrt(dot(a, a));
}
double get_angle(Point a, Point b) {  // 求A->B的有向角
  return acos(dot(a, b) / get_length(a) / get_length(b));
}
double area(Point a, Point b,Point c) {
  // A为顶点，向量AB与向量AC的叉积，即三角形ABC的面积的2倍（有向）
  return cross(b - a, c - a);
}
Point rotate(Point a, double angle) {
  // 将向量A顺时针旋转angle度
  return Point(a.x * cos(angle) + a.y * sin(angle),
    -a.x * sin(angle) + a.y * cos(angle));
}
Point get_line_intersection(Point p, Vector v, Point q, Vector w) {
  // 两直线的交点
  // 使用前提，直线必须有交点
  // cross(v, w) == 0则两直线平行或者重合
  Vector u = p - q;
  double t = cross(w, u) / cross(v, w);
  return p + v * t;
}
double distance_to_line(Point p, Point a, Point b) {
  // 点到直线的距离，直线为AB所在直线
  Vector v1 = b - a, v2 = p - a;
  return fabs(cross(v1, v2) / get_length(v1));
}
double distance_to_segment(Point p, Point a, Point b) {
  // 点到线段的距离，线段为线段AB
  if (a == b) return get_length(p - a);
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (sign(dot(v1, v2)) < 0) return get_length(v2);
  if (sign(dot(v1, v3)) > 0) return get_length(v3);
  return distance_to_line(p, a, b);
}
Point get_line_projection(Point p, Point a, Point b) {
  // 点在直线上的投影，直线为AB所在直线
  Vector v = b - a;
  return a + v * (dot(v, p - a) / dot(v, v));
}
bool on_segment(Point p, Point a, Point b) {
  // 点是否在线段上
  return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
}
bool segment_intersection(Point a1, Point a2, Point b1, Point b2) {
  // 判断两个线段是否相交
  double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1);
  double c3 = cross(b2 - b1, a2 - b1), c4 = cross(b2 - b1, a1 - b1);
  return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
```

```
}
// 多边形
double polygon_area(Point p[], int n) {  // 求多边形面积
  double s = 0;
  for (int i = 1; i + 1 < n; i++) s += cross(p[i] - p[0], p[i + 1] - p[i]);
  return s / 2;
}
}  // namespace Geometry
```

7.4 点类

```
using T = i64;
struct Point {
  T x;
  T y;
  Point(T x = 0, T y = 0) : x(x), y(y) {}
  Point &operator+=(const Point &p) {
    x += p.x, y += p.y;
    return *this;
  }
  Point &operator-=(const Point &p) {
    x -= p.x, y -= p.y;
    return *this;
  }
  Point &operator*=(const T &v) {
    x *= v, y *= v;
    return *this;
  }
  friend Point operator-(const Point &p) {
    return Point(-p.x, -p.y);
  }
  friend Point operator+(Point lhs, const Point &rhs) {
    return lhs += rhs;
  }
  friend Point operator-(Point lhs, const Point &rhs) {
    return lhs -= rhs;
  }
  friend Point operator*(Point lhs, const T &rhs) {
    return lhs *= rhs;
  }
};
T dot(const Point &a, const Point &b) {
  return a.x * b.x + a.y * b.y;
}
T cross(const Point &a, const Point &b) {
  return a.x * b.y - a.y * b.x;
}
```

7.5 分数类

```
struct Fraction {
  long long num;
  long long den;
  Fraction(long long num=0,long long den=1) {
    if(den<0) {
      num=-num;
      den=-den;
    }
    assert(den!=0);
    long long g=gcd(abs(num),den);
    this->num=num/g;
    this->den=den/g;
  }
  Fraction operator +(const Fraction &o)const {
    return Fraction(num*o.den+o.num,den*o.den);
  }
  Fraction operator -(const Fraction &o)const {
    return Fraction(num*o.den-den*o.num,den*o.den);
  }
  Fraction operator *(const Fraction &o)const {
    return Fraction(num*o.num,den*o.den);
  }
  Fraction operator /(const Fraction &o)const {
    return Fraction(num*o.den,den*o.num);
  }
  bool operator <(const Fraction &o)const {
    return num*o.den< den*o.num;
  }
  bool operator ==(const Fraction &o)const {
    return num*o.den==den*o.num;
  }
};
```

7.6 取模

```
template <class T>
T power(T a, i64 b) {
  T res = 1;
  for (; b; b /= 2, a *= a) {
    if (b % 2) {
      res *= a;
    }
  }
  return res;
}

template <int P>
struct MInt {
  int x;
  MInt() : x{} {}
  MInt(i64 x) : x{norm(x % P)} {}

  int norm(int x) {
    if (x < 0) x += P;
    if (x >= P) x -= P;
    return x;
  }
  int val() const {
    return x;
  }
  MInt operator-() const {
    MInt res;
    res.x = norm(P - x);
    return res;
  }
  MInt inv() const {
    assert(x != 0);
    return power(*this, P - 2);
  }
  MInt &operator*=(const MInt &rhs) {
    x = 1LL * x * rhs.x % P;
    return *this;
  }
  MInt &operator+=(const MInt &rhs) {
    x = norm(x + rhs.x);
    return *this;
  }
  MInt &operator-=(const MInt &rhs) {
    x = norm(x - rhs.x);
    return *this;
  }
  MInt &operator/=(const MInt &rhs) {
    return *this *= rhs.inv();
  }
  friend MInt operator*(const MInt &lhs, const MInt &rhs) {
    MInt res = lhs;
    res *= rhs;
    return res;
  }
  friend MInt operator+(const MInt &lhs, const MInt &rhs) {
    MInt res = lhs;
    res += rhs;
    return res;
  }
  friend MInt operator-(const MInt &lhs, const MInt &rhs) {
    MInt res = lhs;
    res -= rhs;
    return res;
  }
  friend MInt operator/(const MInt &lhs, const MInt &rhs) {
    MInt res = lhs;
    res /= rhs;
    return res;
  }
  friend std::istream &operator>>(std::istream &is, MInt &a) {
    i64 v;
    is >> v;
    a = MInt(v);
    return is;
```

```
    }
    friend std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
};

constexpr int P = 1000000007;
using Z = MInt<P>;
```

7.7 离散化

```
//数组离散化 含重复元素
std::sort(sub_a, sub_a+n);
int size = std::unique(sub_a, sub_a+n) - sub_a;//size为离散化后元素个数
for (i = 0; i < n; i++) {
    a[i] = std::lower_bound(sub_a, sub_a+size, a[i]) - sub_a + 1;//k为b[i] 经离散化后对应的值
}

//坐标离散化
int compress(int *x1, int *x2, int w){
    std::vector<int> xs;
    for (int i = 0; i < N; i++) {
        for (int d = -1; d <= 1; d++) {
            int tx1 = x1[i] + d, tx2 = x2[i] + d;
            if (1 <= tx1 && tx1 <= w) xs.push_back(tx1);
            if (1 <= tx2 && tx2 <= w) xs.push_back(tx2);
        }
    }
    std::sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end()), xs.end());
    for (int i = 0; i < N; i++) {
        x1[i] = find(xs.begin(), xs.end(), x1[i]) - xs.begin();
        x2[i] = find(xs.begin(), xs.end(), x2[i]) - xs.begin();
    }
    return xs.size();
}
```

7.8 Dancing Link

```
struct DLX{
    const static int maxn=20010;
    #define FF(i,A,s) for(int i = A[s];i != s;i = A[i])
    int L[maxn],R[maxn],U[maxn],D[maxn];
    int size,col[maxn],row[maxn],s[maxn],H[maxn];
    bool vis[70];
    int ans[maxn],cnt;
    void init(int m){
        for(int i=0;i<=m;i++){
            L[i]=i-1;R[i]=i+1;U[i]=D[i]=i;s[i]=0;
        }
        memset(H,-1,sizeof(H));
        L[0]=m;R[m]=0;size=m+1;
    }
    void link(int r,int c){
        U[size]=c;D[size]=D[c];U[D[c]]=size;D[c]=size;
        if(H[r]<0)H[r]=L[size]=R[size]=size;
        else {
            L[size]=H[r];R[size]=R[H[r]];
            L[R[H[r]]]=size;R[H[r]]=size;
        }
        s[c]++;col[size]=c;row[size]=r;size++;
    }
    void del(int c){//精确覆盖
        L[R[c]]=L[c];R[L[c]]=R[c];
        FF(i,D,c)FF(j,R,i)U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
    }
    void add(int c){  //精确覆盖
        R[L[c]]=L[R[c]]=c;
        FF(i,U,c)FF(j,L,i)++s[col[U[D[j]]=D[U[j]]=j]];
    }
    bool dfs(int k){//精确覆盖
        if(!R[0]){
            cnt=k;return 1;
        }
        int c=R[0];FF(i,R,0)if(s[c]>s[i])c=i;
        del(c);
        FF(i,D,c){
            FF(j,R,i)del(col[j]);
            ans[k]=row[i];if(dfs(k+1))return true;
            FF(j,L,i)add(col[j]);
        }
        add(c);
        return 0;
    }
    void remove(int c){//重复覆盖
        FF(i,D,c)L[R[i]]=L[i],R[L[i]]=R[i];
    }
     void resume(int c){//重复覆盖
        FF(i,U,c)L[R[i]]=R[L[i]]=i;
    }
    int A(){//估价函数
        int res=0;
        memset(vis,0,sizeof(vis));
        FF(i,R,0)if(!vis[i]){
            res++;vis[i]=1;
            FF(j,D,i)FF(k,R,j)vis[col[k]]=1;
        }
        return res;
    }
    void dfs(int now,int &ans){//重复覆盖
        if(R[0]==0)ans=min(ans,now);
        else if(now+A()<ans){
            int temp=INF,c;
            FF(i,R,0)if(temp>s[i])temp=s[i],c=i;
            FF(i,D,c){
                remove(i);FF(j,R,i)remove(j);
                dfs(now+1,ans);
                FF(j,L,i)resume(j);resume(i);
            }
        }
    }
}dlx;
```

7.9 __builtin

```
— Built-in Function: int __builtin_ffs (unsigned int x)
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

— Built-in Function: int __builtin_clz (unsigned int x)
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result is undefined.

— Built-in Function: int __builtin_ctz (unsigned int x)
Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is 0, the result is undefined.

— Built-in Function: int __builtin_popcount (unsigned int x)
Returns the number of 1-bits in x.

— Built-in Function: int __builtin_parity (unsigned int x)
Returns the parity of x, i.e. the number of 1-bits in x modulo 2.

— Built-in Function: int __builtin_ffsl (unsigned long)
Similar to __builtin_ffs, except the argument type is unsigned long.

— Built-in Function: int __builtin_clzl (unsigned long)
Similar to __builtin_clz, except the argument type is unsigned long.

— Built-in Function: int __builtin_ctzl (unsigned long)
Similar to __builtin_ctz, except the argument type is unsigned long.

— Built-in Function: int __builtin_popcountl (unsigned long)
Similar to __builtin_popcount, except the argument type is unsigned long.

— Built-in Function: int __builtin_parityl (unsigned long)
Similar to __builtin_parity, except the argument type is unsigned long.
```

```
— Built-in Function: int __builtin_ffsll (unsigned long long)
Similar to __builtin_ffs, except the argument type is unsigned long long.

— Built-in Function: int __builtin_clzll (unsigned long long)
Similar to __builtin_clz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_ctzll (unsigned long long)
Similar to __builtin_ctz, except the argument type is unsigned long long.

— Built-in Function: int __builtin_popcountll (unsigned long long)
Similar to __builtin_popcount, except the argument type is unsigned long long.

— Built-in Function: int __builtin_parityll (unsigned long long)
Similar to __builtin_parity, except the argument type is unsigned long long.

随机数函数：
default_random_engine：随机非负数（不建议单独使用）。
uniform_int_distribution：指定范围的随机非负数。
uniform_real_distribution：指定范围的随机实数。
bernoulli_distribution：指定概率的随机布尔值。
示例:default_random_engine e;
    uniform_int_distribution<int>u(0,9);
    cout<<u(e)<<endl;
```

7.10 随机

```cpp
std::vector<int> permutation(100);
for (int i = 0; i < 100; i++) {
  permutation[i] = i+1;
}
std::mt19937_64 mt1(1); //64位
std::mt19937 mt2(2); //32位
shuffle(permutation.begin(), permutation.end(), mt2); // 打乱序列
for (auto it: permutation) {
  std::cout << it << " ";
}
```

7.11 程序计时

```
(double)clock() / CLOCKS_PER_SEC
```

7.12 Zeller 日期公式

```cpp
// weekday=(id+1)%7;{Sun=0,Mon=1,...}   getId(1, 1, 1) = 0
int getId(int y, int m, int d) {
  if (m < 3) { y --; m += 12; }
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307; }
//  y<0: 统一加400的倍数年
auto date(int id) {
  int x=id+1789995, n, i, j, y, m, d;
  n = 4 * x / 146097; x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001; x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447; d = x - 2447 * j / 80; x = j / 11;
  m = j + 2 - 12 * x; y = 100 * (n - 49) + i + x;
  return make_tuple(y, m, d); }
```

7.13 模拟退火

```cpp
int n; double ansx,ansy;
struct point{ double x,y,w; } p[MAXN];
double f(double x,double y){
  double sum=0.0; for(int i=1;i<=n;i++){
    double dx=x-p[i].x,dy=y-p[i].y;
    sum+=sqrt(dx*dx+dy*dy)*p[i].w;
  } return sum; }
void SA(){
  double T=3000,d=0.999,tt=1e-15;
  while(T>tt){
    double newx=ansx+(2*rand()-RAND_MAX)*T;
    double newy=ansy+(2*rand()-RAND_MAX)*T;
    double delta=f(newx,newy)-f(ansx,ansy);
    if(delta<0 || exp(-delta/T)*RAND_MAX>rand()){
      ansx=newx; ansy=newy; }
    T*=d; }}

int main(){
  srand(time(0)); scanf("%d", &n);
  for(int i=1;i<=n;i++){
    scanf("%lf %lf %lf", &p[i].x, &p[i].y, &p[i].w);
  } ansx=ansy=0.0; for(int i=1;i<=n;i++){
    ansx+=p[i].x; ansy+=p[i].y; }
  ansx/=n; ansy/=n; SA();
  printf("%.3lf %.3lf\n", ansx, ansy); return 0; }
```

7.14 双端队列BFS

```cpp
// 适用范围：边权值为可能有，也可能没有（由于 BFS 适用于权值为 1 的图，所以一般是 0 or 1 或者能够转化为这种边权值的最短路问题。例一束光线从 (n,m) 点向左射出，遇到# 可以选择方向不变或者向上下左右反射光线，问最少选择反射几次能从第一行向左射出

char a[maxn][maxn];
struct node{
  int x,y,step,d;
};
int dir[4][2]={1,0,-1,0,0,1,0,-1};
bool vis[maxn][maxn][4];
bool bfs(int n,int m){
  deque<node>q;
  q.push_front({n,m,0,3});
  while(!q.empty()){
    node now=q.front(),next;q.pop_front();
    if(now.x==1&&now.d==3){cout<<now.step<<endl;return 1;}
    if(vis[now.x][now.y][now.d]) continue;
    vis[now.x][now.y][now.d]=1;
    next=now;
    next.x+=dir[now.d][0],next.y+=dir[now.d][1];
    if(next.x>=1&&next.x<=n&&next.y>=1&&next.y<=m) q.push_front(next);
    if(a[now.x][now.y]=='#'){
      for(int i=0;i<4;i++){
        next=now;
        next.x+=dir[i][0];
        next.y+=dir[i][1];
        next.step++;next.d=i;
        if(next.x<1||next.x>n||next.y<1||next.y>m||
          vis[next.x][next.y][i]) continue;
        q.push_back(next);
      }
    }
  }
  return 0;
}
```

7.15 A*

```cpp
int Hash[9]={1,1,2,6,24,120,720,5040,40320};
int dir[4][2]={-1,0,1,0,0,-1,0,1};
char d[5]="udlr";
int vis[maxn];
struct node{
  int f[3][3];
  int g,h,hashval,x,y;
  bool operator <(const node a) const{
    return a.g+a.h<g+h;
  }
};
struct path{
  int pre;
  char ch;
}p[maxn];
int get_h(int f[][3]){
  int ans=0;
  for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
      if(f[i][j]){
        ans+=abs(i-(f[i][j]-1)/3)+abs(j-(f[i][j]-1)%3);
      }
    }
  }
  return ans;
}
```

```
bool checkedge(node next){
  if(next.x>=0&&next.y>=0&&next.x<3&&next.y<3) return 1;
  return 0;
}
void As_bfs(node e){
  priority_queue<node>q;
  node now,next;
  for(int i=0;i<9;i++) now.f[i/3][i%3]=(i+1)%9;
  int end_ans=get_hash(now);
  e.h=get_h(e);e.g=0;
  e.hashval=get_hash(e);
  p[e.hashval].pre=-1;
  q.push(e);
  while(!q.empty()){
    now=q.top(); q.pop();
    if (now.hashval == end_ans) {
      print(now.hashval);
      cout << endl;
      return;
    }
    if(vis[now.hashval]) continue;vis[now.hashval]=1;
    for(int i=0;i<4;i++){
      next=now;
      next.x=now.x+dir[i][0];
      next.y=now.y+dir[i][1];
      if(checkedge(next)){
        swap(next.f[now.x][now.y], next.f[next.x][next.y]);
        next.hashval = get_hash(next);
        if(vis[next.hashval]) continue;
        next.g++; next.h = get_h(next);
        p[next.hashval].pre=now.hashval;
        p[next.hashval].ch=d[i];
        q.push(next);
      }
    }
  }
}
```

7.16 std::bitset

```
a ^ b //Xor
a & b //And
a | b //Or
bs.any()         //是否存在1
bs.none()        //是否都为0
bs.count()       //1的个数
b.size()         //二进制位的个数
b[pos]           //第 pos 位二进制数
b.test(pos)      //第 pos 位是否为 1
b.set()          //全设为 1
b.set(pos)       //将 pos 处设为 1
b.reset()        //全设为 0
b.reset(pos)     //将 pos 处设为 0
b.flip()         //全部取反
b.flip(pos)      //将 pos 处取反
b.to_ulong()     //返回一个 unsigned long 值
b._Find_first() //返回第一个1的位置
b._Find_next(x) //返回x之后下一个1的位置
```

7.17 priority_queue的重载运算符

```
struct cmp {
  bool operator()(int x, int y) { return pos[x] > pos[y]; }
};
priority_queue<int, vector<int>, cmp> q;
```

7.18 对拍

```
#!/bin/bash
for i in $(seq 1 100000);do
  ./gen
  ./a
  ./a1
  if diff a.out a1.out; then
    echo $i "AC"
  else
    echo $i "WA"
    exit 0
  fi
done
```

8. 模版题

8.1 三分

给出一个 N 次函数，保证在范围 $[l,r]$ 内存在一点 x，使得 $[l,x]$ 上单调增，$[x,r]$ 上单调减。试求出 x 的值。

第一行一次包含一个正整数 N 和两个实数 l,r，含义如题目描述所示。

第二行包含 $N+1$ 个实数，从高到低依次表示该 N 次函数各项的系数。

```
const double dif = 1e-7;
const int MAXN = 13;
int n;
double coe[MAXN + 5];
double check(double x)  //不用秦九韶公式，普通计算……
{
  double temp = 1.0, ans = 0.0;
  for (int i = n; i >= 0; i--, temp *= x) ans += coe[i] * temp;
  return ans;
}
double Find(double L, double R) {
  double Lmid, Rmid;
  while (L + dif <= R) {
    Lmid = L + (R - L) / 3.0;
    Rmid = R - (R - L) / 3.0;
    if (check(Lmid) > check(Rmid))
      R = Rmid;
    else
      L = Lmid;
  }
  return (Lmid + Rmid) / 2.0;
}
int main() {
  double L, R;
  scanf("%d", &n), cin >> L >> R;
  for (int i = 0; i <= n; i++) cin >> coe[i];
  printf("%.5lf\n", Find(L, R));
  // printf("%.5lf\n",check(-0.41421));
  return 0;
}
```

8.2 传递闭包

给定一张点数为 n 的有向图的邻接矩阵，图中不包含自环，求该有向图的传递闭包。

一张图的邻接矩阵定义为一个 $n\times n$ 的矩阵 $A=(a_{ij})_{n\times n}$，其中

$$a_{ij}=\begin{cases}1, i\ \ j\\0, i\ \ j\end{cases}$$

一张图的传递闭包定义为一个 $n\times n$ 的矩阵 $B=(b_{ij})_{n\times n}$，其中

$$b_{ij}=\begin{cases}1, i\ \ j\\0, i\ \ j\end{cases}$$

输入数据共 $n+1$ 行。

第一行一个正整数 n。

第 2 到 $n+1$ 行每行 n 个整数，第 $i+1$ 行第 j 列的整数为 a_{ij}。

输出数据共 n 行。

第 1 到 n 行每行 n 个整数，第 i 行第 j 列的整数为 b_{ij}。

对于 100% 的数据，$1\le n\le 100$，保证 $a_{ij}\in\{0,1\}$ 且 $a_{ii}=0$。

```
int d[110][110], n;
int main() {
  cin >> n;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++) cin >> d[i][j];
  for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
        d[i][j] = max(d[i][j], min(d[i][k], d[k][j]));
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) cout << d[i][j] << " ";
    cout << endl;
  }
  return 0;
}
```

8.3 裴蜀定理

给定一个包含 n 个元素的**整数**序列 A，记作 $A_1,A_2,A_3,...,A_n$。

求另一个包含 n 个元素的待定**整数**序列 X，记 $S=\sum_{i=1}^{n} A_i \times X_i$，使得 $S>0$ 且 S 尽可能的小。
第一行一个整数 n，表示序列元素个数。
第二行 n 个整数，表示序列 A。
输出格式一行一个整数，表示 $S>0$ 的前提下 S 的最小值。
对于 100% 的数据，$1 \le n \le 20$，$|A_i| \le 10^5$，且 A 序列不全为 0。

```
inline int gcd(int x, int y) { return y ? gcd(y, x % y) : x; }
int n;
int main() {
  scanf("%d", &n);
  int ans = 0, tmp;
  for (int i = 1; i <= n; i++) {
    scanf("%d", &tmp);
    if (tmp < 0) tmp = -tmp;
    ans = gcd(ans, tmp);
  }
  printf("%d", ans);
}
/*
裴蜀定理内容：ax + by = c, x ∈ Z*, y ∈ Z* 成立的充要条件是 gcd(a, b)|c
*/
```

8.4 矩阵快速幂

一个 $m \times n$ 的**矩阵**是一个由 m 行 n 列元素排列成的矩形阵列。即形如

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

本题中认为矩阵中的元素 a_{ij} 是整数。
两个大小分别为 $m \times n$ 和 $n \times p$ 的矩阵 A, B **相乘**的结果为一个大小为 $m \times p$ 的矩阵。将结果矩阵记作 C，则

$$c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}, \qquad (1 \le i \le m, 1 \le j \le p).$$

而如果 A 的列数与 B 的行数不相等，则无法进行乘法。
可以验证，矩阵乘法满足结合律，即 $(AB)C=A(BC)$。
一个大小为 $n \times n$ 的矩阵 A 可以与自身进行乘法，得到的仍是大小为 $n \times n$ 的矩阵，记作 $A^2=A \times A$。进一步地，还可以递归地定义任意高次方 $A^k=A \times A^{k-1}$，或称 $A^k=\underbrace{A \times A \times \cdots \times A}_{k\text{ 次}}$。

特殊地，定义 A^0 为单位矩阵 $I=\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$。

给定 $n \times n$ 的矩阵 A，求 A^k。
第一行两个整数 n, k。接下来 n 行，每行 n 个整数，第 i 行的第 j 的数表示 $A_{i,j}$。
输出 A^k
共 n 行，每行 n 个数，第 i 行第 j 个数表示 $(A^k)_{i,j}$，每个元素对 10^9+7 取模。
对于 100% 的数据，$1 \le n \le 100$，$0 \le k \le 10^{12}$，$|A_{i,j}| \le 1000$。

```
int n;
struct ahaha {
  ll a[maxn][maxn];  //一定要用long long存矩阵，否则在过程中会爆掉
  ahaha() { memset(a, 0, sizeof a); }
  inline void build() {  //建造单位矩阵
    for (int i = 1; i <= n; ++i) a[i][i] = 1;
  }
} a;
ahaha operator*(const ahaha &x, const ahaha &y) {  //重载运算符
  ahaha z;
  for (int k = 1; k <= n; ++k)
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        z.a[i][j] = (z.a[i][j] + x.a[i][k] * y.a[k][j] % mo) % mo;
  return z;
}
ll k;
inline void init() {
  n = read();
  k = read();
  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j) a.a[i][j] = read();
}
int main() {
  init();
  ahaha ans;
  ans.build();
  do {  //递推快速幂，与普通的递推快速幂无异，但*不能缩写为*=
    if (k & 1) ans = ans * a;
    a = a * a;
    k >>= 1;
  } while (k);
  for (int i = 1; i <= n; putchar('\n'), ++i)
    for (int j = 1; j <= n; ++j) printf("%d ", ans.a[i][j]);
  return 0;
}
```

8.5 欧拉路径

求有向图字典序最小的欧拉路径。
第一行两个整数 n, m 表示有向图的点数和边数。
接下来 m 行每行两个整数 u, v 表示存在一条 $u \to v$ 的有向边。
如果不存在欧拉路径，输出一行 'No'。
否则输出一行 $m+1$ 个数字，表示字典序最小的欧拉路径。对于 100% 的数据，$1 \le u, v \le n \le 10^5$，$m \le 2 \times 10^5$。

```
const int MAX = 100010;
int n, m, u, v, del[MAX];
int du[MAX][2];  //记录入度和出度
stack<int> st;
vector<int> G[MAX];
void dfs(int now) {
  for (int i = del[now]; i < G[now].size(); i = del[now]) {
    del[now] = i + 1;
    dfs(G[now][i]);
  }
  st.push(now);
}
int main() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= m; i++)
    scanf("%d%d", &u, &v), G[u].push_back(v), du[u][1]++, du[v][0]++;
  for (int i = 1; i <= n; i++) sort(G[i].begin(), G[i].end());
  int S = 1, cnt[2] = {0, 0};  //记录
  bool flag = 1;  // flag=1表示，所有的节点的入度都等于出度，
  for (int i = 1; i <= n; i++) {
    if (du[i][1] != du[i][0]) {
      flag = 0;
      if (du[i][1] - du[i][0] == 1 /*出度比入度多1*/)
        cnt[1]++, S = i;
      else if (du[i][0] - du[i][1] == 1 /*入度比出度多1*/)
        cnt[0]++;
      else
        return puts("No"), 0;
    }
  }
  if ((!flag) && !(cnt[0] == cnt[1] && cnt[0] == 1)) return !puts("No"), 0;
  //不满足欧拉回路的判定条件，也不满足欧拉路径的判定条件，直接输出"No"
  dfs(S);
  while (!st.empty()) printf("%d ", st.top()), st.pop();
  return 0;
}
```

8.6 差分约束

给出一组包含 m 个不等式，有 n 个未知数的形如：

$$\begin{cases} x_{c_1}-x_{c_1'} \le y_1 \\ x_{c_2}-x_{c_2'} \le y_2 \\ \cdots \\ x_{c_m}-x_{c_m'} \le y_m \end{cases}$$

的不等式组，求任意一组满足这个不等式组的解。
第一行为两个正整数 n, m，代表未知数的数量和不等式的数量。
接下来 m 行，每行包含三个整数 c, c', y，代表一个不等式 $x_c - x_{c'} \le y$。
输出一行，n 个数，表示 $x_1, x_2 \cdots x_n$ 的一组可行解，如果有多组解，请输出任意一组，无解请输出 'NO'。

对于 100% 的数据，$1 \le n, m \le 5\times10^3$，$-10^4 \le y \le 10^4$，$1 \le c, c' \le n$，$c \ne c'$。

```
int n, m, cnt, elast[5005], dis[5005], num[5005];
bool vis[5005];
struct edge {
  int to, len, next;
} e[10005];
queue<int> q;
void add(int u, int v, int w) {
  e[++cnt].to = v;
  e[cnt].len = w;
  e[cnt].next = elast[u];
  elast[u] = cnt;
}
bool spfa(int x) {
  dis[x] = 0;
  q.push(x);
  vis[x] = true;
  num[x]++;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;
    for (int i = elast[u]; i != 0; i = e[i].next)
      if (dis[e[i].to] > dis[u] + e[i].len) {
        dis[e[i].to] = dis[u] + e[i].len;
        if (!vis[e[i].to]) {
          q.push(e[i].to);
          vis[e[i].to] = true;
          num[e[i].to]++;
          if (num[e[i].to] == n + 1) return false;
        }
      }
  }
  return true;
}
int main() {
  scanf("%d %d", &n, &m);
  for (int i = 1; i <= n; i++) dis[i] = INT_MAX;
  for (int i = 1; i <= m; i++) {
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    add(v, u, w);
  }
  for (int i = 1; i <= n; i++) add(n + 1, i, 0);
  if (!spfa(n + 1)) {
    printf("NO");
    return 0;
  }
  for (int i = 1; i <= n; i++) printf("%d ", dis[i]);
  return 0;
}
```

8.7 全源最短路（Johnson）

给定一个包含 n 个结点和 m 条带权边的有向图，求所有点对间的最短路径长度，一条路径的长度定义为这条路径上所有边的权值和。

注意：

1. 边权**可能**为负，且图中**可能**存在重边和自环；
2. 部分数据卡 n 轮 SPFA 算法。

第 1 行：2 个整数 n, m，表示给定有向图的结点数量和有向边数量。

接下来 m 行：每行 3 个整数 u, v, w，表示有一条权值为 w 的有向边从编号为 u 的结点连向编号为 v 的结点。

若图中存在负环，输出仅一行 -1。

若图中不存在负环：

输出 n 行：令 $dis_{i,j}$ 为从 i 到 j 的最短路，在第 i 行输出 $\sum_{j=1}^{n} j \times dis_{i,j}$，注意这个结果可能超过 int 存储范围。

如果不存在从 i 到 j 的路径，则 $dis_{i,j} = 10^9$；如果 $i = j$，则 $dis_{i,j} = 0$。

对于 100% 的数据，$1 \le n \le 3 \times 10^3$，$1 \le m \le 6 \times 10^3$，$1 \le u, v \le n$，$-3 \times 10^5 \le w \le 3 \times 10^5$。

```
#define INF 1e9
struct edge {
  int v, w, next;
} e[10005];
struct node {
  int dis, id;
  bool operator<(const node& a) const { return dis > a.dis; }
  node(int d, int x) { dis = d, id = x; }
};
int head[5005], vis[5005], t[5005];
int cnt, n, m;
long long h[5005], dis[5005];
void addedge(int u, int v, int w) {
  e[++cnt].v = v;
  e[cnt].w = w;
  e[cnt].next = head[u];
  head[u] = cnt;
}
bool spfa(int s) {
  queue<int> q;
  memset(h, 63, sizeof(h));
  h[s] = 0, vis[s] = 1;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = 0;
    for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].v;
      if (h[v] > h[u] + e[i].w) {
        h[v] = h[u] + e[i].w;
        if (!vis[v]) {
          vis[v] = 1;
          q.push(v);
          t[v]++;
          if (t[v] == n + 1) return false;
        }
      }
    }
  }
  return true;
}
void dijkstra(int s) {
  priority_queue<node> q;
  for (int i = 1; i <= n; i++) dis[i] = INF;
  memset(vis, 0, sizeof(vis));
  dis[s] = 0;
  q.push(node(0, s));
  while (!q.empty()) {
    int u = q.top().id;
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].v;
      if (dis[v] > dis[u] + e[i].w) {
        dis[v] = dis[u] + e[i].w;
        if (!vis[v]) q.push(node(dis[v], v));
      }
    }
  }
  return;
}
int main() {
  ios::sync_with_stdio(false);
  cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    addedge(u, v, w);
  }
  for (int i = 1; i <= n; i++) addedge(0, i, 0);
  if (!spfa(0)) {
    cout << -1 << endl;
    return 0;
  }
  for (int u = 1; u <= n; u++)
    for (int i = head[u]; i; i = e[i].next) e[i].w += h[u] - h[e[i].v];
  for (int i = 1; i <= n; i++) {
    dijkstra(i);
    long long ans = 0;
    for (int j = 1; j <= n; j++) {
      if (dis[j] == INF)
        ans += j * INF;
      else
        ans += j * (dis[j] + h[j] - h[i]);
    }
    cout << ans << endl;
  }
}
```

```
  | return 0;
}
```

8.8 AC自动机（二次加强版）

给你一个文本串 S 和 n 个模式串 $T_{1\sim n}$，请你分别求出每个模式串 T_i 在 S 中出现的次数。
第一行包含一个正整数 n 表示模式串的个数。
接下来 n 行，第 i 行包含一个由小写英文字母构成的非空字符串 T_i。
最后一行包含一个由小写英文字母构成的非空字符串 S。
输出包含 n 行，其中第 i 行包含一个非负整数表示 T_i 在 S 中出现的次数。
对于 100% 的数据，$1 \le n \le 2 \times 10^5$，$T_{1\sim n}$ 的长度总和不超过 2×10^5，S 的长度不超过 2×10^6。

```
#define maxn 2000001
char s[maxn], T[maxn];
int n, cnt, vis[200051], ans, in[maxn], Map[maxn];
struct kkk {
  | int son[26], fail, flag, ans;
  | void clear() { memset(son, 0, sizeof(son)), fail = flag
    ↪ = ans = 0; }
} trie[maxn];
queue<int> q;
void insert(char* s, int num) {
  | int u = 1, len = strlen(s);
  | for (int i = 0; i < len; i++) {
  |   | int v = s[i] - 'a';
  |   | if (!trie[u].son[v]) trie[u].son[v] = ++cnt;
  |   | u = trie[u].son[v];
  | }
  | if (!trie[u].flag) trie[u].flag = num;
  | Map[num] = trie[u].flag;
}
void getFail() {
  | for (int i = 0; i < 26; i++) trie[0].son[i] = 1;
  | q.push(1);
  | while (!q.empty()) {
  |   | int u = q.front();
  |   | q.pop();
  |   | int Fail = trie[u].fail;
  |   | for (int i = 0; i < 26; i++) {
  |   |   | int v = trie[u].son[i];
  |   |   | if (!v) {
  |   |   |   | trie[u].son[i] = trie[Fail].son[i];
  |   |   |   | continue;
  |   |   | }
  |   |   | trie[v].fail = trie[Fail].son[i];
  |   |   | in[trie[v].fail]++;
  |   |   | q.push(v);
  |   | }
  | }
}
void topu() {
  | for (int i = 1; i <= cnt; i++)
  |   | if (in[i] == 0) q.push(i);
  | while (!q.empty()) {
  |   | int u = q.front();
  |   | q.pop();
  |   | vis[trie[u].flag] = trie[u].ans;
  |   | int v = trie[u].fail;
  |   | in[v]--;
  |   | trie[v].ans += trie[u].ans;
  |   | if (in[v] == 0) q.push(v);
  | }
}
void query(char* s) {
  | int u = 1, len = strlen(s);
  | for (int i = 0; i < len; i++) u = trie[u].son[s[i] -
    ↪ 'a'], trie[u].ans++;
}
int main() {
  | scanf("%d", &n);
  | cnt = 1;
  | for (int i = 1; i <= n; i++) {
  |   | scanf("%s", s);
  |   | insert(s, i);
  | }
  | getFail();
  | scanf("%s", T);
  | query(T);
  | topu();
  | for (int i = 1; i <= n; i++) printf("%d\n",
    ↪ vis[Map[i]]);
}
```

8.9 Dirichlet 前缀和

给定一个长度为 n 的数列 $a_1, a_2, a_3, \ldots, a_n$。
现在你要求出一个长度为 n 的数列 $b_1, b_2, b_3, \ldots, b_n$，满足

$$b_k = \sum_{i|k} a_i$$

由于某些神秘原因，这里的 b_k 要对 2^{32} 取模。
对于 100% 的数据，$1 \le n \le 2 \times 10^7$，$0 \le seed < 2^{32}$。

```
/*
  | 设 i = ∏ p_k^{α_k}, j = ∏ p_k^{β_k}
  | 那么 a_i 贡献到 b_j 当且仅当 ∀k, α_k ≤ β_k
  | 发现这个实际上就是一个关于质因子分解后的指数的高维前缀和，
    ↪ 使用类似FMT的方法就可以了。
  | 根据埃氏筛的时间复杂度分析，
  | T(n) = ∑_p ⌊n/p⌋ = O(n log log n)。
*/
for (int i = 1; i <= len; i++)
  for (int j = 1; p[i] * j <= n; j++) a[p[i] * j] += a[j];
```

8.10 Stoer-Wagner

定义无向图 G 的最小割为：一个去掉后可以使 G 变成两个连通分量，且边权和最小的边集的边权和。
给出一张无向图 G，求其最小割。
第一行，两个数 n, m，表示点数与边数。
接下来 m 行，每行三个正整数 a_i, b_i, w_i，表示 a_i 到 b_i 有一条边权为 w_i 的边。
输出一行，一个整数表示 G 的最小割，如果图不联通，请输出 '0'。
对于 100% 的数据，$n \le 600, m \le \frac{n\times(n-1)}{2}$，保证 $\sum_{i=1}^{m} w_i \le 10^9$。

```
const int MAXN = 610, INF = 1e9;
int n, m, x, y, z, s, t, dis[MAXN][MAXN], w[MAXN],
  ↪ dap[MAXN], vis[MAXN],
  | ord[MAXN];
int proc(int x) {
  | memset(vis, 0, sizeof(vis));
  | memset(w, 0, sizeof(w));
  | w[0] = -1;
  | for (int i = 1; i <= n - x + 1; i++) {
  |   | int mx = 0;
  |   | for (int j = 1; j <= n; j++) {
  |   |   | if (!dap[j] && !vis[j] && w[j] > w[mx]) {
  |   |   |   | mx = j;
  |   |   | }
  |   | }
  |   | vis[mx] = 1, ord[i] = mx;
  |   | for (int j = 1; j <= n; j++) {
  |   |   | if (!dap[j] && !vis[j]) {
  |   |   |   | w[j] += dis[mx][j];
  |   |   | }
  |   | }
  | }
  | s = ord[n - x], t = ord[n - x + 1];
  | return w[t];
}
int sw() {
  | int res = INF;
  | for (int i = 1; i < n; i++) {
  |   | res = min(res, proc(i));
  |   | dap[t] = 1;
  |   | for (int j = 1; j <= n; j++) {
  |   |   | dis[s][j] += dis[t][j];
  |   |   | dis[j][s] += dis[j][t];
  |   | }
  | }
  | return res;
}
int main() {
  | scanf("%d%d", &n, &m);
  | for (int i = 1; i <= m; i++) {
  |   | scanf("%d%d%d", &x, &y, &z);
  |   | dis[x][y] += z, dis[y][x] += z;
  | }
  | printf("%d\n", sw());
  | return 0;
}
```

8.11 2-SAT

有 n 个布尔变量 x_1~x_n，另有 m 个需要满足的条件，每个条件的形式都是「x_i 为 'true' / 'false' 或 x_j 为 'true' / 'false'」。比如「x_1 为真或 x_3 为假」、「x_7 为假或 x_2 为假」。
2-SAT 问题的目标是给每个变量赋值使得所有条件得到满足。
第一行两个整数 n 和 m，意义如题面所述。
接下来 m 行每行 4 个整数 i, a, j, b，表示「x_i 为 a 或 x_j 为 b」$(a, b \in \{0, 1\})$
如无解，输出 'IMPOSSIBLE'；否则输出 'POSSIBLE'。
下一行 n 个整数 $x_1 \sim x_n$（$x_i \in \{0, 1\}$），表示构造出的解。

```
n = read(), m = read();
for (int i = 0; i < m; ++i) {
  // 笔者习惯对 x 点标号为 x，-x 标号为 x+n，当然也可以反过来。
  int a = read(), va = read(), b = read(), vb = read();
  if (va && vb) {  // a, b 都真，-a -> b, -b -> a
    g[a + n].push_back(b);
    g[b + n].push_back(a);
  } else if (!va && vb) {  // a 假 b 真，a -> b, -b -> -a
    g[a].push_back(b);
    g[b + n].push_back(a + n);
  } else if (va && !vb) {  // a 真 b 假，-a -> -b, b -> a
    g[a + n].push_back(b + n);
    g[b].push_back(a);
  } else if (!va && !vb) {  // a, b 都假，a -> -b, b -> -a
    g[a].push_back(b + n);
    g[b].push_back(a + n);
  }
}
// 注意所有东西都要开两倍空间，因为每个变量存了两次
void tarjan(int u) {
  low[u] = dfn[u] = ++dfsClock;
  stk.push(u);
  ins[u] = true;
  for (const auto &v : g[u]) {
    if (!dfn[v])
      tarjan(v), low[u] = std::min(low[u], low[v]);
    else if (ins[v])
      low[u] = std::min(low[u], dfn[v]);
  }
  if (low[u] == dfn[u]) {
    ++sccCnt;
    do {
      color[u] = sccCnt;
      u = stk.top();
      stk.pop();
      ins[u] = false;
    } while (low[u] != dfn[u]);
  }
}
// 笔者使用了 Tarjan 找环，得到的 color[x] 是 x 所在的 scc 的拓扑逆序。
for (int i = 1; i <= (n << 1); ++i)
  if (!dfn[i]) tarjan(i);
for (int i = 1; i <= n; ++i)
  if (color[i] == color[i + n]) {  // x 与 -x 在同一强连通分量内，一定无解
    puts("IMPOSSIBLE");
    exit(0);
  }
puts("POSSIBLE");
for (int i = 1; i <= n; ++i)
  print((color[i] < color[i + n])),
    putchar(' ');  // 如果不使用 Tarjan 找环，请改成大于号
puts("");
```

8.12 点分治

给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。
第一行两个数 n, m。
第 2 到第 n 行，每行三个整数 u, v, w，代表树上存在一条连接 u 和 v 边权为 w 的路径。
接下来 m 行，每行一个整数 k，代表一次询问。
对于每次询问输出一行一个字符串代表答案，存在输出 'AYE'，否则输出 'NAY'。
- 对于 100% 的数据，保证 $1 \le n \le 10^4$，$1 \le m \le 100$，$1 \le k \le 10^7$，$1 \le u, v \le n$，$1 \le w \le 10^4$。

```
void chkmax(T &a, T b) { a > b ? (a = a) : (a = b); }
void chkmin(T &a, T b) { a > b ? (a = b) : (a = a); }
int n, m;
int S;
int size[10101];
struct edge {
  int from, to, cost, net;
  edge(int f = 0, int t = 0, int cost = 0, int nex = 0) {
    from = f;
    to = t;
    this->cost = cost;
    net = nex;
  }
} edges[1010101];
int tot, head[101001], mx[101011], minn = 0x3f3f3f3f, root;
int vis[1010110];
void add(int x, int y, int z) {
  edges[++tot] = edge(x, y, z, head[x]);
  head[x] = tot;
}
void find(int x, int fa) {
  size[x] = 1;
  mx[x] = 0;
  for (int i = head[x]; i; i = edges[i].net) {
    edge v = edges[i];
    if (v.to == fa || vis[v.to]) continue;
    find(v.to, x);
    size[x] += size[v.to];
    chkmax(mx[x], size[v.to]);
  }
  chkmax(mx[x], S - size[x]);
  if (mx[x] < mx[root]) {
    root = x;
  }
}
int que[1010110], ans[102210101];
int dis[1010101], hhd, a[10101101];
void get_dis(int x, int len, int fa) {
  dis[++hhd] = a[x];
  for (int i = head[x]; i; i = edges[i].net) {
    edge v = edges[i];
    if (vis[v.to] || v.to == fa) continue;
    a[v.to] = len + edges[i].cost;
    get_dis(v.to, len + edges[i].cost, x);
  }
}
void solve(int s, int len, int w) {
  hhd = 0;
  a[s] = len;
  get_dis(s, len, 0);
  for (int i1 = 1; i1 <= hhd; i1++)
    for (int i2 = 1; i2 <= hhd; i2++) {
      if (i1 != i2) {
        ans[dis[i1] + dis[i2]] += w;
      }
    }
}
void Divide(int x) {
  solve(x, 0, 1);
  vis[x] = 1;
  for (int i = head[x]; i; i = edges[i].net) {
    edge v = edges[i];
    if (vis[v.to]) continue;
    solve(v.to, edges[i].cost, -1);
    S = size[x];
    root = 0;
    mx[0] = n;
    find(v.to, x);
    Divide(root);
  }
}
int main() {
  read(n);
  read(m);
  for (int i = 1; i < n; i++) {
    int x, y, z;
    read(x);
    read(y);
    read(z);
    add(x, y, z);
    add(y, x, z);
  }
  S = n;
  mx[0] = n;
  root = 0;
```

```
  // minn = 0x3f3f3f3f;
  find(1, 0);
  // printf("%d\n",mx[root]);
  Devede(root);
  for (int i = 1; i <= m; i++) {
    int k;
    read(k);
    printf("%s\n", (ans[k]) ? "AYE" : "NAY");
    // printf("%d\n",ans[k]);
  }
  return 0;
}
```

8.13 拉格朗日插值

由小学知识可知 n 个点 (x_i, y_i) 可以唯一地确定一个多项式 $y = f(x)$。现在，给定这 n 个点，请你确定这个多项式，并求出 $f(k) \bmod 998244353$ 的值。

第一行两个整数 n, k。

接下来 n 行，第 i 行两个整数 x_i, y_i。

输出一行一个整数，表示 $f(k) \bmod 998244353$ 的值。

$1 \le n \le 2 \times 10^3$，$1 \le x_i, y_i, k < 998244353$，$x_i$ 两两不同。

```
LL n, k;
LL x[2005], y[2005];
LL ksm(LL a, LL b) {
  LL ans = 1ll, y = a;
  while (b) {
    if (b & 1) ans *= y, ans %= mod;
    y *= y;
    y %= mod;
    b >>= 1;
  }
  return ans % mod;
}  //快速幂，用于求逆元
LL inv(LL x) { return ksm(x, mod - 2); }  //逆元qwq
LL calc(LL t) {
  LL ans = 0;
  for (int i = 1; i <= n; i++) {
    LL p = y[i] % mod, q = 1ll;
    for (int j = 1; j <= n; j++) {
      if (i == j) continue;
      p = p * (t - x[j]) % mod;
      q = q * (x[i] - x[j]) % mod;  //直接照着公式算qwq
    }
    ans += p * inv(q) % mod;
    ans %= mod;
  }
  return (ans % mod + mod) % mod;
}
int main(void) {
  cin >> n >> k;
  for (int i = 1; i <= n; i++) cin >> x[i] >> y[i];
  cout << calc(k) << endl;
  return 0;
}
```

8.14 中国剩余定理

假如有 16 头母猪，如果建了 3 个猪圈，剩下 1 头猪就没有地方安家了。如果建造了 5 个猪圈，但是仍然有 1 头猪没有地方去，然后如果建造了 7 个猪圈，还有 2 头没有地方去。你作为曹总的私人秘书理所当然要将准确的猪数报给曹总，你该怎么办?

第一行包含一个整数 n ——建立猪圈的次数，接下来 n 行，每行两个整数 a_i, b_i，表示建立了 a_i 个猪圈，有 b_i 头猪没有去处。你可以假定 $a_1 \sim a_n$ 互质。

输出包含一个正整数，即为曹冲至少养母猪的数目。

$1 \le n \le 10, 0 \le b_i < a_i \le 100000, 1 \le \prod a_i \le 10^{18}$

```
ll n, a[16], m[16], Mi[16], mul = 1, X;
void exgcd(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return;
  }
  exgcd(b, a % b, x, y);
  int z = x;
  x = y, y = z - y * (a / b);
}
int main() {
  n = rd();
  for (int t = 1; t <= n; ++t) {
    int M = rd();
    m[t] = M;
    mul *= M;
    a[t] = rd();
  }
  for (int t = 1; t <= n; ++t) {
    Mi[t] = mul / m[t];
    ll x = 0, y = 0;
    exgcd(Mi[t], m[t], x, y);
    X += a[t] * Mi[t] * (x < 0 ? x + m[t] : x);
  }
  printf("%lld", X % mul);
  return 0;
}
```

8.15 线性基

给定 n 个整数（数字可能重复），求在这些数中选取任意个，使得他们的异或和最大。

第一行一个数 n，表示元素个数

接下来一行 n 个数

输出一行，表示答案。

$1 \le n \le 50, 0 \le S_i < 2^{50}$

```
const int MN = 60;
ll a[61], tmp[61];
bool flag;
void ins(ll x) {
  for (reg int i = MN; ~i; i--)
    if (x & (1ll << i))
      if (!a[i]) {
        a[i] = x;
        return;
      } else
        x ^= a[i];
  flag = true;
}
bool check(ll x) {
  for (reg int i = MN; ~i; i--)
    if (x & (1ll << i))
      if (!a[i])
        return false;
      else
        x ^= a[i];
  return true;
}
ll qmax(ll res = 0) {
  for (reg int i = MN; ~i; i--) res = max(res, res ^
    ↪ a[i]);
  return res;
}
ll qmin() {
  if (flag) return 0;
  for (reg int i = 0; i <= MN; i++)
    if (a[i]) return a[i];
}
ll query(ll k) {  // k小
  reg ll res = 0;
  reg int cnt = 0;
  k -= flag;
  if (!k) return 0;
  for (reg int i = 0; i <= MN; i++) {
    for (int j = i - 1; ~j; j--)
      if (a[i] & (1ll << j)) a[i] ^= a[j];
    if (a[i]) tmp[cnt++] = a[i];
  }
  if (k >= (1ll << cnt)) return -1;
  for (reg int i = 0; i < cnt; i++)
    if (k & (1ll << i)) res ^= tmp[i];
  return res;
}
int main() {
  int n;
  ll x;
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) scanf("%lld", &x), ins(x);
  printf("%lld\n", qmax());
  return 0;
}
```

8.16 扫描线

求 n 个四边平行于坐标轴的矩形的面积并。

第一行一个正整数 n。
接下来 n 行每行四个非负整数 x_1, y_1, x_2, y_2，表示一个矩形的四个端点坐标为 $(x_1, y_1), (x_1, y_2), (x_2, y_2), (x_2, y_1)$。
输出行一个正整数，表示 n 个矩形的并集覆盖的总面积。
对于 100% 的数据，$1 \le n \le 10^5$，$0 \le x_1 < x_2 \le 10^9$，$0 \le y_1 < y_2 \le 10^9$。

```
#define lson (x << 1)
#define rson (x << 1 | 1)
const int MAXN = 1e6 + 10;
typedef long long ll;
int n, cnt = 0;
ll x1, y1, x2, y2, X[MAXN << 1];
struct ScanLine {
  ll l, r, h;
  int mark;
  //  mark用于保存权值 (1 / -1)
  bool operator<(const ScanLine &rhs) const { return h < rhs.h; }
} line[MAXN << 1];
struct SegTree {
  int l, r, sum;
  ll len;
  //  sum: 被完全覆盖的次数;
  //  len: 区间内被截的长度。
} tree[MAXN << 2];
void build_tree(int x, int l, int r) {
  //  我觉得最不容易写错的一种建树方法
  tree[x].l = l, tree[x].r = r;
  tree[x].len = 0;
  tree[x].sum = 0;
  if (l == r) return;
  int mid = (l + r) >> 1;
  build_tree(lson, l, mid);
  build_tree(rson, mid + 1, r);
  return;
}
void pushup(int x) {
  int l = tree[x].l, r = tree[x].r;
  if (tree[x].sum /* 也就是说被覆盖过 */) tree[x].len = X[r + 1] - X[l];
  //      更新长度
  else
    tree[x].len = tree[lson].len + tree[rson].len;
  //      合并儿子信息
}
void edit_tree(int x, ll L, ll R, int c) {
  int l = tree[x].l, r = tree[x].r;
  //  注意，l、r和L、R的意义完全不同
  //  l、r表示这个节点管辖的下标范围
  //  而L、R则表示需要修改的真实区间
  if (X[r + 1] <= L || R <= X[l]) return;
  //  这里加等号的原因：
  //  假设现在考虑 [2,5], [5,8] 两条线段，要修改 [1,5] 区间的sum
  //  很明显，虽然5在这个区间内，[5,8] 却并不是我们希望修改的线段
  //  所以总结一下，就加上了等号
  if (L <= X[l] && X[r + 1] <= R) {
    tree[x].sum += c;
    pushup(x);
    return;
  }
  edit_tree(lson, L, R, c);
  edit_tree(rson, L, R, c);
  pushup(x);
}
int main() {
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) {
    scanf("%lli %lli %lli %lli", &x1, &y1, &x2, &y2);
    X[2 * i - 1] = x1, X[2 * i] = x2;
    line[2 * i - 1] = (ScanLine){x1, x2, y1, 1};
    line[2 * i] = (ScanLine){x1, x2, y2, -1};
    //      一条线段含两个端点，一个矩形的上下边都需要扫描线扫过
  }
  n <<= 1;
  //  直接把 n <<= 1 方便操作
  sort(line + 1, line + n + 1);
  sort(X + 1, X + n + 1);
  int tot = unique(X + 1, X + n + 1) - X - 1;
  //  去重最简单的方法：使用unique!（在<algorithm> 库中）
  build_tree(1, 1, tot - 1);
  //  为什么是 tot - 1 ：
  //  因为右端点的对应关系已经被篡改了嘛…
  //  [1, tot - 1] 描述的就是 [X[1], X[tot]]
  ll ans = 0;
  for (int i = 1; i < n /* 最后一条边是不用管的 */; i++) {
    edit_tree(1, line[i].l, line[i].r, line[i].mark);
    //      先把扫描线信息导入线段树
    ans += tree[1].len * (line[i + 1].h - line[i].h);
    //      然后统计面积
  }
  printf("%lli", ans);
  return 0;
}
```

8.17 失配树

给定一个字符串 s，定义它的 **k 前缀** pre_k 为字符串 $s_{1\ldots k}$，**k 后缀** suf_k 为字符串 $s_{|s|-k+1\ldots|s|}$，其中 $1 \le k \le |s|$。
定义 $\mathbf{Border}(s)$ 为**对于 $i \in [1, |s|)$，满足 $pre_i = suf_i$** 的字符串 pre_i 的集合。$\mathbf{Border}(s)$ 中的每个元素都称之为字符串 s 的 border。
有 m 组询问，每组询问给定 p, q，求 s 的 **$\boldsymbol{p}$ 前缀** 和 **$\boldsymbol{q}$ 前缀** 的 **最长公共 border** 的长度。
第一行一个字符串 s。
第二行一个整数 m。
接下来 m 行，每行两个整数 p, q。
对于每组询问，输出一行一个整数，表示答案。若不存在公共 border，请输出 0。
对于 100% 的数据，$1 \le p, q \le |s| \le 10^6$，$1 \le m \le 10^5$，$s_i \in [\mathtt{a}, \mathtt{z}]$。

```
const int MaxN = 1000000 + 5, MaxM = 100000 + 5;
const int MaxLog = 20;
int N, M;
char S[MaxN];
int Fail[MaxN];
int Fa[MaxLog + 1][MaxN], Dep[MaxN];
void init() {
  scanf("%s", S + 1);
  N = strlen(S + 1);
  scanf("%d", &M);
}
inline int getLca(int u, int v) {
  if (Dep[u] < Dep[v]) std::swap(u, v);
  int d = Dep[u] - Dep[v];
  for (int i = MaxLog; i >= 0; --i)
    if (d & (1 << i)) u = Fa[i][u];
  if (u == v) return u;
  for (int i = MaxLog; i >= 0; --i)
    if (Fa[i][u] != Fa[i][v]) {
      u = Fa[i][u];
      v = Fa[i][v];
    }
  return Fa[0][u];
}
void solve() {
  for (int i = 2, j = 0; i <= N; ++i) {
    while (j > 0 && S[i] != S[j + 1]) j = Fail[j];
    if (S[i] == S[j + 1]) j++;
    Fail[i] = j;
  }
  for (int u = 1; u <= N; ++u) {
    Fa[0][u] = Fail[u];
    Dep[u] = Dep[Fail[u]] + 1;
    for (int i = 1; (1 << i) <= Dep[u]; ++i) Fa[i][u] = Fa[i - 1][Fa[i - 1][u]];
  }
  for (int q = 1; q <= M; ++q) {
    int x, y;
    scanf("%d %d", &x, &y);
    int l = getLca(x, y);
    if (l == x || l == y)
      printf("%d\n", Fa[0][l]);
    else
      printf("%d\n", l);
  }
}
int main() {
  init();
  solve();
  return 0;
}
```

8.18 左偏树/可并堆

如题，一开始有 n 个小根堆，每个堆包含且仅包含一个数。接下来需要支持两种操作：

1. ‘1 x y‘：将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在用一个堆内，则无视此操作）。
2. ‘2 x‘：输出第 x 个数所在的堆最小数，并将这个最小数删除（若有多个最小数，优先删除先输入的；若第 x 个数已经被删除，则输出 -1 并无视删除操作）。

第一行包含两个正整数 n, m，分别表示一开始小根堆的个数和接下来操作的个数。

第二行包含 n 个正整数，其中第 i 个正整数表示第 i 个小根堆初始时包含且仅包含的数。

接下来 m 行每行 2 个或 3 个正整数，表示一条操作，格式如下：

操作 1：‘1 x y‘

操作 2：‘2 x‘

输出包含若干行整数，分别依次对应每一个操作 2 所得的结果。

对于 100% 的数据：$n \leq 10^5$，$m \leq 10^5$，初始时小根堆中的所有数都在‘int‘ 范围内。

```
#define MAXN 150010
#define ls S[x].Son[0]
#define rs S[x].Son[1]
struct Tree {
  int dis, val, Son[2], rt;
} S[MAXN];
int N, T, A, B, C, i;
inline int Merge(int x, int y);
inline int Get(int x) { return S[x].rt == x ? x : S[x].rt = Get(S[x].rt); }
inline void Pop(int x) {
  S[x].val = -1, S[ls].rt = ls, S[rs].rt = rs, S[x].rt = Merge(ls, rs);
}
inline int Merge(int x, int y) {
  if (!x || !y) return x + y;
  if (S[x].val > S[y].val || (S[x].val == S[y].val && x > y)) swap(x, y);
  rs = Merge(rs, y);
  if (S[ls].dis < S[rs].dis) swap(ls, rs);
  S[ls].rt = S[rs].rt = S[x].rt = x, S[x].dis = S[rs].dis + 1;
  return x;
}
int main() {
  cin >> N >> T;
  S[0].dis = -1;
  for (i = 1; i <= N; ++i) S[i].rt = i, scanf("%d", &S[i].val);
  for (i = 1; i <= T; ++i) {
    scanf("%d%d", &A, &B);
    if (A == 1) {
      scanf("%d", &C);
      if (S[B].val == -1 || S[C].val == -1) continue;
      int f1 = Get(B), f2 = Get(C);
      if (f1 != f2) S[f1].rt = S[f2].rt = Merge(f1, f2);
    } else {
      if (S[B].val == -1)
        printf("-1\n");
      else
        printf("%d\n", S[Get(B)].val), Pop(Get(B));
    }
  }
  return 0;
}
```

8.19 扩展 KMP/exKMP（Z 函数）

给定两个字符串 a, b，你要求出两个数组：

- b 的 z 函数数组 z，即 b 与 b 的每一个后缀的 LCP 长度。- b 与 a 的每一个后缀的 LCP 长度数组 p。

对于一个长度为 n 的数组 a，设其权值为 $\operatorname{xor}_{i=1}^{n} i \times (a_i + 1)$。

输入两行两个字符串 a, b。

输出第一行一个整数，表示 z 的权值。

第二行一个整数，表示 p 的权值。

对于 100% 的数据，$1 \leq |a|, |b| \leq 2 \times 10^7$，所有字符均为小写字母。

```
const int N = 2e7 + 10;
ll nxt[N], ext[N];
void qnxt(char *c) {
  int len = strlen(c);
  int p = 0, k = 1, l;  //我们会在后面先逐位比较出 nxt[1] 的值，这里先设 k 为 1
  //如果 k = 0，p 就会锁定在 |c| 不会被更改，无法达成算法优化的效果啦
  nxt[0] = len;  //以 c[0] 开始的后缀就是 c 本身，最长公共前缀自然为 |c|
  while (p + 1 < len && c[p] == c[p + 1]) p++;
  nxt[1] = p;  //先逐位比较出 nxt[1] 的值
  for (int i = 2; i < len; i++) {
    p = k + nxt[k] - 1;  //定义
    l = nxt[i - k];      //定义
    if (i + l <= p)
      nxt[i] = l;  //如果灰方框小于初始的绿方框，直接确定 nxt[i] 的值
    else {
      int j = max(0, p - i + 1);
      while (i + j < len && c[i + j] == c[j]) j++;  //否则进行逐位比较
      nxt[i] = j;
      k = i;  //此时的 x + nxt[x] - 1 一定刷新了最大值，于是我们要重新赋值 k
    }
  }
}
void exkmp(char *a, char *b) {
  int la = strlen(a), lb = strlen(b);
  int p = 0, k = 0, l;
  while (p < la && p < lb && a[p] == b[p]) p++;  //先算出初值用于递推
  ext[0] = p;
  for (int i = 1; i < la; i++)  //下面都是一样的逻辑啦
  {
    p = k + ext[k] - 1;
    l = nxt[i - k];
    if (i + l <= p)
      ext[i] = l;
    else {
      int j = max(0, p - i + 1);
      while (i + j < la && j < lb && a[i + j] == b[j]) j++;
      ext[i] = j;
      k = i;
    }
  }
}
int la, lb;
char a[N], b[N];
ll ans;
int main() {
  cin >> a >> b;
  qnxt(b);
  exkmp(a, b);
  la = strlen(a), lb = strlen(b), ans = 0;
  for (int i = 0; i < lb; i++)  //要注意下标从 0 开始
  {
    ans ^= (i + 1) * (nxt[i] + 1);
  }
  cout << ans << "\n";
  ans = 0;
  for (int i = 0; i < la; i++) {
    ans ^= (i + 1) * (ext[i] + 1);
  }
  cout << ans;
  return 0;
}
```

8.20 BSGS

给定一个质数 p，以及一个整数 b，一个整数 n，现在要求你计算一个最小的非负整数 l，满足 $b^l \equiv n \pmod{p}$。

输入仅一行，有 3 个整数，依次代表 p, b, n。

输出仅一行，如果有 l 满足该要求，输出最小的 l，否则输出‘no solution‘。

- 对于所有的测试点，保证 $2 \leq b, n < p < 2^{31}$。

```
long long a, b, p;
long long power(long long a, long long b, long long c) {  //快速幂
  if (b == 0) return 1 % c;
  long long ans = 1, t = a;
  while (b > 0) {
    if (b % 2 == 1) ans = ans * t % c;
    b /= 2;
    t = t * t % c;
  }
```

```
  | return ans;
}
long long bsgs(long long a, long long b, long long p) {  //bsgs
  | map<long long, long long> hash;
  | hash.clear();  //建立一个Hash表
  | b %= p;
  | long long t = sqrt(p) + 1;
  | for (register long long i = 0; i < t; ++i)
  |   | hash[(long long)b * power(a, i, p) % p] = i;  //将每个j对应的值插入Hash表
  | a = power(a, t, p);
  | if (!a) return b == 0 ? 1 : -1;  //特判
  | for (register long long i = 1; i <= t;
  |   |   ++i) {  //在Hash表中查找是否有i对应的j值
  |   | long long val = power(a, i, p);
  |   | int j = hash.find(val) == hash.end() ? -1 : hash[val];
  |   | if (j >= 0 && i * t - j >= 0) return i * t - j;
  | }
  | return -1;  //无解返回-1
}
signed main() {
  | scanf("%lld%lld%lld", &p, &a, &b);
  | long long ans = bsgs(a, b, p);
  | if (ans == -1)
  |   | printf("no solution\n");
  | else
  |   | printf("%lld\n", ans);
  | return 0;
}
```

8.21 线段树分治

神犇有一个 n 个节点的图。
因为神犇是神犇，所以在 k 时间内有 m 条边会出现后消失。
神犇要求出每一时间段内这个图是否是二分图。
这么简单的问题神犇当然会做了，于是他想考考你。
第一行三个整数 n, m, k。
接下来 m 行，每行四个整数 x, y, l, r，表示有一条连接 x, y 的边在 l 时刻出现 r 时刻消失。
输出 k 行，第 i 行一个字符串 'Yes' 或 'No'，表示在第 i 时间段内这个图是否是二分图。
$n, k = 10^5$，$m = 2 \times 10^5$。$1 \le x, y \le n$，$0 \le l \le r \le k$。

```
const int N = 10101010;
int n, m, k, fa[N], height[N], top;
struct E {
  | int x, y;
} e[N];
struct Stack {
  | int x, y, add;
} st[N];
vector<int> t[N];
int findfa(int x) {
  | while (x != fa[x]) x = fa[x];
  | return fa[x];
}
void debug() {
  | printf("\n***************\n下标");
  | for (int i = 1; i <= n * 2; i++) printf("%d ", i);
  | printf("\n父亲");
  | for (int i = 1; i <= n * 2; i++) printf("%d ", fa[i]);
  | printf("\n祖先 (代表元)");
  | for (int i = 1; i <= n * 2; i++) printf("%d ", findfa(i));
}
void merge(int x, int y) {
  | int fx = findfa(x), fy = findfa(y);
  | if (height[fx] > height[fy]) swap(fx, fy);
  | st[++top] = (Stack){fx, fy, height[fx] == height[fy]};
  | fa[fx] = fy;
  | if (height[fx] == height[fy]) height[fy]++;
}
void update(int u, int l, int r, int L, int R, int x) {
  | if (l > R || r < L) return;
  | if (L <= l && r <= R) {
  |   | t[u].push_back(x);
  |   | return;
  | }
  | int mid = l + r >> 1;
  | update(u << 1, l, mid, L, R, x);
  | update(u << 1 | 1, mid + 1, r, L, R, x);
}
void solve(int u, int l, int r) {
  | //  debug();
  | int ans = 1;
  | int lasttop = top;
  | for (int i = 0; i < t[u].size(); i++) {
  |   | int a = findfa(e[t[u].at(i)].x);
  |   | int b = findfa(e[t[u].at(i)].y);
  |   | if (a == b) {
  |   |   | for (int k = l; k <= r; k++) printf("No\n");
  |   |   | ans = 0;
  |   |   | break;
  |   | }
  |   | merge(e[t[u].at(i)].x, e[t[u].at(i)].y + n);
  |   | merge(e[t[u].at(i)].y, e[t[u].at(i)].x + n);
  | }
  | if (ans) {
  |   | if (l == r)
  |   |   | printf("Yes\n");
  |   | else {
  |   |   | int mid = l + r >> 1;
  |   |   | solve(u << 1, l, mid);
  |   |   | solve(u << 1 | 1, mid + 1, r);
  |   | }
  | }
  | while (top > lasttop) {
  |   | height[fa[st[top].x]] -= st[top].add;
  |   | fa[st[top].x] = st[top].x;
  |   | top--;
  | }
  | return;
}
int main() {
  | n = read();
  | m = read();
  | k = read();
  | for (int i = 1; i <= m; i++) {
  |   | e[i].x = read();
  |   | e[i].y = read();
  |   | int l = read() + 1, r = read();
  |   | update(1, 1, k, l, r, i);
  | }
  | for (int i = 1; i <= 2 * n; i++) fa[i] = i, height[i] = 1;
  | solve(1, 1, k);
  | return 0;
}
```

8.22 莫队

具体来说，小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R 的袜子中随机选出两只来穿。尽管小 Z 并不在意两只袜子是不是完整的一双，他却很在意袜子的颜色，毕竟穿两只不同色的袜子会很尴尬。
你的任务便是告诉小 Z，他有多大的概率抽到两只颜色相同的袜子。当然，小 Z 希望这个概率尽量高，所以他可能会询问多个 (L, R) 以方便自己选择。
然而数据中有 $L = R$ 的情况，请特判这种情况，输出 '0/1'。
输入文件第一行包含两个正整数 N 和 M。N 为袜子的数量，M 为小 Z 所提的询问的数量。接下来一行包含 N 个正整数 C_i，其中 C_i 表示第 i 只袜子的颜色，相同的颜色用相同的数字表示。再接下来 M 行，每行两个正整数 L, R 表示一个询问。
输出 M 行，对于每个询问在一行中输出分数 A/B 表示从该询问的区间 $[L, R]$ 中随机抽出两只袜子颜色相同的概率。若该概率为 0 则输出 '0/1'，否则输出的 A/B 必须为最简分数。（详见样例）
100% 的数据中，$N, M \le 50000$，$1 \le L < R \le N$，$C_i \le N$。

```
const int maxn = 50005;
int n, m, pos[maxn], c[maxn];
LL s[maxn], ans;
struct data {
  | int l, r, id;
  | LL a, b;
} a[maxn];
bool cmp(const data& a, const data& b) {
  | if (pos[a.l] == pos[b.l]) return a.r < b.r;
  | return a.l < b.l;
}
bool cmp_id(const data& a, const data& b) { return a.id < b.id; }
void update(int p, int add) {
  | ans -= s[c[p]] * s[c[p]];
  | s[c[p]] += add;
  | ans += s[c[p]] * s[c[p]];
```

```
}
void solve() {
  for (int i = 1, l = 1, r = 0; i <= m; i++) {
    for (; r < a[i].r; r++) update(r + 1, 1);
    for (; r > a[i].r; r--) update(r, -1);
    for (; l < a[i].l; l++) update(l, -1);
    for (; l > a[i].l; l--) update(l - 1, 1);
    if (a[i].l == a[i].r) {
      a[i].a = 0;
      a[i].b = 1;
      continue;
    }
    a[i].a = ans - (a[i].r - a[i].l + 1);
    a[i].b = (a[i].r - a[i].l + 1) * 1LL * (a[i].r -
      ↪ a[i].l);
    LL g = __gcd(a[i].a, a[i].b);
    a[i].a /= g;
    a[i].b /= g;
  }
}
int main() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
  int block = sqrt(n);
  for (int i = 1; i <= n; i++) pos[i] = (i - 1) / block +
    ↪ 1;
  for (int i = 1; i <= m; i++) {
    scanf("%d%d", &a[i].l, &a[i].r);
    a[i].id = i;
  }
  sort(a + 1, a + m + 1, cmp);
  solve();
  sort(a + 1, a + m + 1, cmp_id);
  for (int i = 1; i <= m; i++) printf("%lld/%lld\n",
    ↪ a[i].a, a[i].b);
  return 0;
}
```

8.23 最小表示法

他们正在玩一种神奇的游戏，叫 Minecraft。
他们现在要做一个由方块构成的长条工艺品。但是方块现在是乱的，而且由于机器的要求，他们只能做到把这个工艺品最左边的方块放到最右边。
他们想，在仅这一个操作下，最漂亮的工艺品能多漂亮。
两个工艺品美观的比较方法是，从头开始比较，如果第 i 个位置上方块不一样那么谁的瑕疵度小，那么谁就更漂亮，如果一样那么继续比较第 $i+1$ 个方块。如果全都一样，那么这两个工艺品就一样漂亮。
第一行一个整数 n，代表方块的数目。
第二行 n 个整数，每个整数按从左到右的顺序输出方块瑕疵度的值。
输出一行 n 个整数，代表最美观工艺品从左到右瑕疵度的值。
- 对于 100% 的数据，$n \leq 3 \times 10^5$。

```
const int inf = 1e9 + 7;
int n, ans, A[300009];
int Min_show()
//最小表示法求出串的最小表示
{
  int i = 0, j = 1, k = 0;
  //两个指针i,j, 任意时刻i!=j, 当前匹配长度为k
  //循环同构串要复制一遍字串（成环）接在A序列后面
  //由于数组过大,(i+k)%n和（j+k)%n代表了字串
  //复制一遍接在原序列后各字符的对应位置
  while (i < n && j < n && k < n) {
    if (A[(i + k) % n] == A[(j + k) % n])
      //两个位置数值相等，匹配长度k++
      k++;
    else {
      if (A[(i + k) % n] > A[(j + k) % n])
        //[i,i+k-1] 与 [j,j+k-1] 相同
        //那么在 [i,i+k-1] 中不可能有最小表示
        //则i+=k+1, 令k=0
        i += k + 1;
      else
        j += k + 1;
      //同上
      if (i == j) i++;
      //任何时候都要满足i!=j
      k = 0;  //匹配长度k归零
    }
  }
  return min(i, j);  //返回最小表示
}
int main() {
  n = read();
  for (int i = 0; i < n; i++)  //初始字串
    A[i] = read();
  ans = Min_show();            //求最小表示
  for (int i = 0; i < n; i++)  //输出最小表示
    printf("%d ", A[(i + ans) % n]);
  return 0;
}
```

8.24 后缀排序

读入一个长度为 n 的由大小写英文字母或数字组成的字符串，请把这个字符串的所有非空后缀按字典序（用 ASCII 数值比较）从小到大排序，然后按顺序输出后缀的第一个字符在原串中的位置。位置编号为 1 到 n。
输出一行，共 n 个整数，表示答案。
$1 \leq n \leq 10^6$。

```
#define rint register int
#define inv inline void
#define ini inline int
#define maxn 1000050
char s[maxn];
int y[maxn], x[maxn], c[maxn];
int sa[maxn], rk[maxn], height[maxn], wt[30];
int n, m;
inv putout(int x) {
  if (!x) {
    putchar(48);
    return;
  }
  rint l = 0;
  while (x) wt[++l] = x % 10, x /= 10;
  while (l) putchar(wt[l--] + 48);
}
inv get_SA() {
  for (rint i = 1; i <= n; ++i) ++c[x[i] = s[i]];
  // c数组是桶
  // x[i] 是第i个元素的第一关键字
  for (rint i = 2; i <= m; ++i) c[i] += c[i - 1];
  //做c的前缀和，我们就可以得出每个关键字最多是在第几名
  for (rint i = n; i >= 1; --i) sa[c[x[i]]--] = i;
  for (rint k = 1; k <= n; k <<= 1) {
    rint num = 0;
    for (rint i = n - k + 1; i <= n; ++i) y[++num] = i;
    // y[i] 表示第二关键字排名为i的数，第一关键字的位置
    //第n-k+1到第n位是没有第二关键字的 所以排名在最前面
    for (rint i = 1; i <= n; ++i)
      if (sa[i] > k) y[++num] = sa[i] - k;
    //排名为i的数 在数组中是否在第k位以后
    //如果满足 (sa[i]>k)
    //那么它可以作为别人的第二关键字，就把它的第一关键字的位置添加进y就行了
    //所以i枚举的是第二关键字的排名，第二关键字靠前的先入队
    for (rint i = 1; i <= m; ++i) c[i] = 0;
    //初始化c桶
    for (rint i = 1; i <= n; ++i) ++c[x[i]];
    //因为上一次循环已经算出了这次的第一关键字 所以直接加就行了
    for (rint i = 2; i <= m; ++i)
      c[i] += c[i - 1];  //第一关键字排名为1~i的数有多少个
    for (rint i = n; i >= 1; --i) sa[c[x[y[i]]]--] =
      y[i], y[i] = 0;
    //因为y的顺序是按照第二关键字的顺序来排的
    //第二关键字靠后的，在同一个第一关键字桶中排名越靠后
    //基数排序
    swap(x, y);
    //这里不用想太多，因为要生成新的x时要用到旧的，就把旧的复制下来，没别的意思
    x[sa[1]] = 1;
    num = 1;
    for (rint i = 2; i <= n; ++i)
      x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] +
        k] == y[sa[i - 1] + k])
                 ? num
                 : ++num;
    //因为sa[i] 已经排好序了，所以可以按排名枚举，生成下一次的第一关键字
    if (num == n) break;
    m = num;
    //这里就不用那个122了，因为都有新的编号了
  }
  for (rint i = 1; i <= n; ++i) putout(sa[i]), putchar('
    ↪ ');
```

```
}
inv get_height() {
  rint k = 0;
  for (rint i = 1; i <= n; ++i) rk[sa[i]] = i;
  for (rint i = 1; i <= n; ++i) {
    if (rk[i] == 1) continue;  //第一名height为0
    if (k) --k;                // h[i]>=h[i-1]+1;
    rint j = sa[rk[i] - 1];
    while (j + k <= n && i + k <= n && s[i + k] == s[j +
      ↪ k]) ++k;
    height[rk[i]] = k;  // h[i]=height[rk[i]];
  }
  putchar(10);
  for (rint i = 1; i <= n; ++i) putout(height[i]),
    ↪ putchar(' ');
}
int main() {
  gets(s + 1);
  n = strlen(s + 1);
  m = 122;
  //因为这个题不读入n和m所以要自己设
  // n表示原字符串长度，m表示字符个数，ascll('z')=122
  //我们第一次读入字符直接不用转化，按原来的ascll码来就可以了
  //因为转化数字和大小写字母还得分类讨论，怪麻烦的
  get_SA();
  // get_height();
}
```

8.25 FMT/FWT

给定长度为 2^n 两个序列 A, B，设

$$C_i = \sum_{j \oplus k = i} A_j \times B_k$$

分别当 $\oplus$ 是 or,and,xor 时求出 C

输入第一行一个数n。第二行 2^n 个数 $A_0..A_{2^n-1}$ 第三行 2^n 个数 $B_0..B_{2^n-1}$

输出三行每行 2^n 个数，分别代表 $\oplus$ 是or,and,xor时 $C_0..C_{2^n-1}$ 的值 mod 998244353

$n \leq 17$。

```
const int N = 1 << 17 | 1;
int n, m;
modint A[N], B[N], a[N], b[N];
inline void in() {
  for (int i = 0; i < n; i++) a[i] = A[i], b[i] = B[i];
}
inline void get() {
  for (int i = 0; i < n; i++) a[i] *= b[i];
}
inline void out() {
  for (int i = 0; i < n; i++) print(a[i], " \n"[i == n -
    ↪ 1]);
}
inline void OR(modint *f, modint x = 1) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; j++) f[i + j + k] += f[i +
        ↪ j] * x;
}
inline void AND(modint *f, modint x = 1) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; j++) f[i + j] += f[i + j +
        ↪ k] * x;
}
inline void XOR(modint *f, modint x = 1) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; j++)
        f[i + j] += f[i + j + k],
        f[i + j + k] = f[i + j] - f[i + j + k] -
          ↪ f[i + j + k],
        f[i + j] *= x, f[i + j +
          ↪ k] *= x;
}
int main() {
  rd(m), n = 1 << m;
  for (int i = 0; i < n; i++) rd(A[i]);
  for (int i = 0; i < n; i++) rd(B[i]);
  in(), OR(a), OR(b), get(), OR(a, P - 1), out();
  in(), AND(a), AND(b), get(), AND(a, P - 1), out();
  in(), XOR(a), XOR(b), get(), XOR(a, (modint)1 / 2),
    ↪ out();
  return 0;
}
```

8.26 LGV 引理

有一个 $n \times n$ 的棋盘，左下角为 $(1,1)$，右上角为 (n,n)，若一个棋子在点 (x,y)，那么走一步只能走到 $(x+1,y)$ 或 $(x,y+1)$。

现在有 m 个棋子，第 i 个棋子一开始放在 $(a_i,1)$，最终要走到 (b_i,n)。问有多少种方案，使得每个棋子都能从起点走到终点，且对于所有棋子，走过路径上的点互不相交。输出方案数 mod 998244353 的值。

两种方案不同当且仅当存在至少一个棋子所经过的点不同。

第一行一个整数 T，表示该测试点的数据组数。

对于每组数据：

第一行两个整数 n, m，分别表示棋盘的大小和起点终点的数量。

接下来 m 行，每行 2 个整数 a_i, b_i，其意义已在题目描述中说明。

对于每一组数据，输出一行一个整数表示方案数 mod 998244353 的值。

- 对于 100% 的数据，$T \leq 5$，$2 \leq n \leq 10^6$，$1 \leq m \leq 100$，$1 \leq a_1 \leq a_2 \leq \cdots \leq a_m \leq n$，$1 \leq b_1 \leq b_2 \leq \cdots \leq b_m \leq n$。

```
int n, m;
const int max_n = 1e6 + 5;
int fac[max_n << 1], inv[max_n << 1], inv_fac[max_n << 1],
  ↪ now = 1;
const int max_m = 100 + 5;
int a[max_m], b[max_m];
const int mod = 998244353;
inline int qpow(int a, int n) {
  int res = 1;
  while (n) {
    if (n & 1) res = 1ll * res * a % mod;
    a = 1ll * a * a % mod;
    n >>= 1;
  }
  return res;
}
inline int C(int n, int m) {
  return 1ll * fac[n] * inv_fac[m] % mod * inv_fac[n - m]
    ↪ % mod;
}
int M[max_m][max_m];
inline int det() {
  int res = 1;
  bool flag_neg = false;
  for (int i = 1; i <= m; ++i) {
    int k = i;
    while (k <= m && !M[k][i]) ++k;
    if (k > m) return 0;
    if (k != i) {
      for (int j = i; j <= m; ++j) swap(M[i][j], M[k]
        ↪ [j]);
      flag_neg ^= 1;
    }
    res = 1ll * res * (M[i][i] + mod) % mod;
    int t = qpow(M[i][i], mod - 2);
    for (int k = i + 1; k <= m; ++k) {
      int t0 = 1ll * t * M[k][i] % mod;
      for (int j = i; j <= m; ++j)
        M[k][j] = (M[k][j] - 1ll * M[i][j] * t0) % mod;
    }
  }
  return flag_neg ? mod - res : res;
}
int main() {
  fac[0] = inv_fac[0] = 1;
  fac[1] = inv_fac[1] = inv[1] = 1;
  int T;
  scanf("%d", &T);
  while (T--) {
    scanf("%d%d", &n, &m);
    while (now < 2 * n) {
      ++now;
      fac[now] = 1ll * fac[now - 1] * now % mod;
      inv[now] = 1ll * (mod - mod / now) * inv[mod %
        ↪ now] % mod;
      inv_fac[now] = 1ll * inv_fac[now - 1] * inv[now] %
        ↪ mod;
    }
    for (int i = 1; i <= m; ++i) scanf("%d%d", a + i, b +
      ↪ i);
    for (int i = 1; i <= m; ++i)
      for (int j = 1; j <= m; ++j)
```

```
  |  |  |  | M[i][j] = a[i] <= b[j] ? C(b[j] - a[i] + n - 1,
                 ↪n - 1) : 0;
  |  | printf("%d\n", det());
  | }
  | return 0;
}
```

8.27 Meissel-Lehmer

给定整数 n，求出 $\pi(n)$ 的值。
$\pi(n)$ 表示 $1 \sim n$ 的整数中质数的个数。
输入一行，一个整数 n。
输出一行，一个整数，表示所求的值。
对于 100% 的数据，$1 \le n \le 10^{13}$。

```
const int M = 3e7, Lim = 8.5e7 + 10;
int l2, tot, lim, BIT[Lim];
ull n, g[M], w[M];
double inv[M];
bool e[Lim];
inline void Add(int x) {
  | e[x] = 1;
  | while (x <= lim) ++BIT[x], x += x & -x;
}
inline ull Query(int x) {
  | ull sum = x;
  | while (x) sum -= BIT[x], x ^= x & -x;
  | return sum;
}
signed main() {
  | register int i, j, tl, tl2, tl3;
  | register ull t, r, x0;
  | scanf("%llu", &n);
  | lim = min(max(max((pow(n / log2(n), 0.66)), l2 =
    ↪sqrtl(n) + 1), 10000), n);
  | for (i = 1; i <= l2; i++) w[i] = i - 1, inv[i] = 1. / i;
  | for (tot = 1; 1ull * lim * tot < n; tot++) g[tot] = n *
    ↪inv[tot] + 1e-6 - 1;
  | --tot;
  | Add(1);
  | for (i = 2; 1ull * i * i <= n; i++) {
  |  | if (e[i]) continue;
  |  | x0 = w[i - 1];
  |  | t = n / i;
  |  | r = 1ull * i * i;
  |  | tl = min(n / r, (ull)tot);
  |  | tl2 = min(tl, n / (1ull * l2 * i));
  |  | tl3 = min(tl2, tot / i);
  |  | for (j = 1; j <= tl3; ++j) g[j] -= g[j * i] - x0;
  |  | for (j = tl3 + 1; j <= tl2; j++) g[j] -= Query(t *
       ↪inv[j] + 1e-6) - x0;
  |  | for (j = tl2 + 1; j <= tl; ++j) g[j] -= w[int(t *
       ↪inv[j] + 1e-6)] - x0;
  |  | for (j = l2; j >= r; --j) w[j] -= w[int(j * inv[i] +
       ↪1e-6)] - x0;
  |  | if (1ull * i * i <= lim) {
  |  |  | for (j = i * i; j <= lim; j += i)
  |  |  |  | if (!e[j]) Add(j);
  |  | }
  | }
  | if (!tot) g[1] = Query(n);
  | printf("%llu", g[1]);
}
```

8.28 线段树合并

首先村落里的一共有 n 座房屋，并形成一个树状结构。然后救济粮分 m 次发放，每次选择两个房屋 (x, y)，然后对于 x 到 y 的路径上（含 x 和 y）每座房子里发放一袋 z 类型的救济粮。
然后深绘里想知道，当所有的救济粮发放完毕后，每座房子里存放的最多的是哪种救济粮。
输入的第一行是两个用空格隔开的正整数，分别代表房屋的个数 n 和救济粮发放的次数 m。
第 2 到第 n 行，每行有两个用空格隔开的整数 a, b，代表存在一条连接房屋 a 和 b 的边。
第 $(n+1)$ 到第 $(n+m)$ 行，每行有三个用空格隔开的整数 x, y, z，代表一次救济粮的发放是从 x 到 y 路径上的每栋房子发放了一袋 z 类型的救济粮。
输出 n 行，每行一个整数，第 i 行的整数代表 i 号房屋存放最多的救济粮的种类，如果有多种救济粮都是存放最多的，输出种类编号最小的一种。
如果某座房屋没有救济粮，则输出 0。
- 对于 100% 测试数据，保证 $1 \le n, m \le 10^5$，$1 \le a, b, x, y \le n$，$1 \le z \le 10^5$。

```
const int N = 1e5 + 10;
int n;
int m;
int v[2 * N];
int x[2 * N];
int ct;
int al[N];
int fa[N][22];
int dep[N];
int ans[N];
struct data  //存最大值的结构体
{
  | int u;
  | int cnt;
  | friend bool operator<(data a, data b) {
  |  | return (a.cnt == b.cnt) ? a.u > b.u : a.cnt < b.cnt;
  | }
  | friend data operator+(data a, data b) { return (data)
    ↪{a.u, a.cnt + b.cnt}; }
};
vector<data> ve[N];
inline void add(int u, int V) {
  | v[++ct] = V;
  | x[ct] = al[u];
  | al[u] = ct;
}
struct linetree  //动态开点线段树合并
{
  | int s[40 * N][2];
  | data v[40 * N];
  | int ct;
  | inline void ih() { ct = n; }
  | inline void ins(int p, int l, int r, data va)  //插入
  | {
  |  | if (r - l == 1) {
  |  |  | v[p] = va + v[p];
  |  |  | return;
  |  | }
  |  | int mid = (l + r) / 2;
  |  | if (va.u <= mid)
  |  |  | ins(s[p][0] = s[p][0] ? s[p][0] : ++ct, l, mid,
          ↪va);
  |  | else
  |  |  | ins(s[p][1] = s[p][1] ? s[p][1] : ++ct, mid, r,
          ↪va);
  |  | v[p] = max(v[s[p][0]], v[s[p][1]]);
  | }
  | inline void mg(int p1, int p2, int l, int r)  //合并
  | {
  |  | if (r - l == 1) {
  |  |  | v[p1] = v[p1] + v[p2];
  |  |  | return;
  |  | }
  |  | int mid = (l + r) / 2;  //分情况讨论下
  |  | if (s[p1][0] && s[p2][0])
  |  |  | mg(s[p1][0], s[p2][0], l, mid);
  |  | else if (s[p2][0])
  |  |  | s[p1][0] = s[p2][0];
  |  | if (s[p1][1] && s[p2][1])
  |  |  | mg(s[p1][1], s[p2][1], mid, r);
  |  | else if (s[p2][1])
  |  |  | s[p1][1] = s[p2][1];
  |  | v[p1] = max(v[s[p1][0]], v[s[p1][1]]);
  | }
} lt;
inline void dfs1(int u)  //处理lca
{
  | for (int i = 0; fa[u][i]; i++) fa[u][i + 1] = fa[fa[u]
    ↪[i]][i];
  | dep[u] = dep[fa[u][0]] + 1;
  | for (int i = al[u]; i; i = x[i])
  |  | if (v[i] != fa[u][0]) fa[v[i]][0] = u, dfs1(v[i]);
}
inline int lca(int u, int v)  //求lca
{
  | if (dep[u] < dep[v]) swap(u, v);
  | for (int d = dep[u] - dep[v], i = 0; d; d >>= 1, i++)
  |  | if (d & 1) u = fa[u][i];
  | if (u == v) return u;
  | for (int i = 18; i >= 0; i--)
  |  | if (fa[u][i] != fa[v][i]) u = fa[u][i], v = fa[v][i];
```

```
  return fa[u][0];
}
inline void dfs2(int u)  //线段树合并
{
  for (int i = al[u]; i; i = x[i])
    if (v[i] != fa[u][0]) dfs2(v[i]), lt.mg(u, v[i], 0, 1e5);
  vector<data>::iterator it;
  for (it = ve[u].begin(); it != ve[u].end(); ++it) {
    lt.ins(u, 0, 1e5, *it);
  }
  ans[u] = lt.v[u].u;
}
int main() {
  scanf("%d%d", &n, &m);
  lt.ih();
  for (int i = 1, u, v; i < n; i++) {
    scanf("%d%d", &u, &v);
    add(u, v), add(v, u);
  }
  dfs1(1);
  for (int i = 1, u, v, va; i <= m; i++)  //拆成4个操作
  {
    scanf("%d%d%d", &u, &v, &va);
    int lc = lca(u, v);
    ve[u].push_back((data){va, 1});
    ve[v].push_back((data){va, 1});
    ve[lc].push_back((data){va, -1});
    ve[fa[lc][0]].push_back((data){va, -1});
  }
  dfs2(1);
  for (int i = 1; i <= n; i++) printf("%d\n", ans[i]);
  return 0;  //拜拜程序~
}
```

8.29 莫队二次离线

珂朵莉给了你一个序列 a，每次查询给一个区间 $[l,r]$，查询 $l \le i < j \le r$，且 $a_i \oplus a_j$ 的二进制表示下有 k 个 1 的二元组 (i,j) 的个数。$\oplus$ 是指按位异或。

第一行三个数表示 n, m, k。

第二行 n 个数表示序列 a。

之后 m 行，每行两个数 l, r 表示一次查询。

输出 m 行，每行一个数表示查询的结果。

对于100% 的数据，$1 \le n, m \le 100000$，$0 \le a_i, k < 16384$。

```
const int maxn = 1e5 + 100;
int blo[maxn], a[maxn];
struct Qry {
  int l, r, id;
  int64_t ans;
  inline bool operator<(const Qry& q) {
    return blo[l] == blo[q.l] ? r < q.r : l < q.l;
  }
} Q[maxn];
vector<tuple<int, int, int>> v[maxn];
int main() {
  int n, m, k;
  scanf("%d%d%d", &n, &m, &k);
  if (k > 14) {
    for (int i = 1; i <= m; ++i) puts("0");
    return 0;
  }
  for (int i = 1; i <= n; ++i) scanf("%d", a + i);
  for (int i = 1; i <= m; ++i) scanf("%d%d", &Q[i].l, &Q[i].r), Q[i].id = i;
  vector<int> buc;
  for (int i = 0; i < 16384; ++i)
    if (__builtin_popcount(i) == k) buc.push_back(i);
  // for (auto x:buc)
  //     fprintf(stderr,"%d ",x);
  int T = sqrt(n);
  for (int i = 1; i <= n; ++i) blo[i] = (i - 1) / T + 1;
  sort(Q + 1, Q + m + 1);
  /***************/
  /*  L右移: l正r负 *\
  /*  l左移: l负r正 *\
  /*  r右移: r正l负 *\
  /*  r左移: r负l正 *\
  /***************/
  static int t[maxn];
  static int pref[maxn];
  for (int i = 1; i <= n; ++i) {
    for (auto x : buc) ++t[a[i] ^ x];
    pref[i] = t[a[i + 1]];
  }
  memset(t, 0, sizeof(t));
  // 预处理前缀贡献
  for (int i = 1, L = 1, R = 0; i <= m; ++i) {
    int l = Q[i].l, r = Q[i].r;
    if (L < l) v[R].emplace_back(L, l - 1, -i);
    while (L < l) {
      Q[i].ans += pref[L - 1];
      ++L;
    }
    if (L > l) v[R].emplace_back(l, L - 1, i);
    while (L > l) {
      Q[i].ans -= pref[L - 2];
      --L;
    }
    if (R < r) v[L - 1].emplace_back(R + 1, r, -i);
    while (R < r) {
      Q[i].ans += pref[R];
      ++R;
    }
    if (R > r) v[L - 1].emplace_back(r + 1, R, i);
    while (R > r) {
      Q[i].ans -= pref[R - 1];
      --R;
    }
  }
  //模拟莫队，算出来第一类贡献，对第二类贡献打上标记
  static int64_t ans[maxn];
  for (int i = 1, id, l, r; i <= n; ++i) {
    for (auto x : buc) ++t[a[i] ^ x];
    for (const auto& x : v[i]) {
      std::tie(l, r, id) = x;
      // if (i>=l) fprintf(stderr,"%d %d\n",i,l);
      for (int j = l, tmp = 0; j <= r; ++j) {
        tmp = t[a[j]];
        if (j <= i && k == 0)
          --tmp;  // 这里（按我的蒟蒻写法）k==0的时候需要特判，因为x^x永远是0，但自己对自己不能产生贡献。
        if (id < 0)
          Q[-id].ans -= tmp;
        else
          Q[id].ans += tmp;
      }
    }
  }
  for (int i = 1; i <= m; ++i) Q[i].ans += Q[i - 1].ans;
  for (int i = 1; i <= m; ++i) ans[Q[i].id] = Q[i].ans;
  for (int i = 1; i <= m; ++i) printf("%lld\n", ans[i]);
}
```

8.30 二分图最大权完美匹配

给定一张二分图，左右部均有 n 个点，共有 m 条带权边，且保证有完美匹配。

求一种完美匹配的方案，使得最终匹配边的边权之和最大。

第一行两个整数 n, m，含义见题目描述。

第 $2 \sim m+1$ 行，每行三个整数 y, c, h 描述了图中的一条从左部的 y 号结点到右部的 c 号节点，边权为 h 的边。

输出第一行一个整数 ans 表示答案。

第二行共 n 个整数 $a_1, a_2, a_3 \cdots a_n$，其中 a_i 表示完美匹配下与**右部**第 i 个点相匹配的左部点的编号。如果存在多种方案，请输出任意一种。

- 对于 100% 的数据，满足 $1 \le n \le 500$，$1 \le m \le n^2$，$-19980731 \le h \le 19980731$ 。且保证没有重边。

```
// Data
const int N = 500;
int n, m, e[N + 7][N + 7];
// KM
int mb[N + 7], vb[N + 7], ka[N + 7], kb[N + 7], p[N + 7], c[N + 7];
int qf, qb, q[N + 7];
void Bfs(int u) {
  int a, v = 0, vl = 0, d;
  for (int i = 1; i <= n; i++) p[i] = 0, c[i] = inf;
  mb[v] = u;
  do {
    a = mb[v], d = inf, vb[v] = 1;
    for (int b = 1; b <= n; b++)
      if (!vb[b]) {
```

```
    |   |   |   | if (c[b] > ka[a] + kb[b] - e[a][b])
    |   |   |   |   | c[b] = ka[a] + kb[b] - e[a][b], p[b] = v;
    |   |   |   | if (c[b] < d) d = c[b], vl = b;
    |   |   | }
    |   | for (int b = 0; b <= n; b++)
    |   |   | if (vb[b])
    |   |   |   | ka[mb[b]] -= d, kb[b] += d;
    |   |   | else
    |   |   |   | c[b] -= d;
    |   | v = vl;
    | } while (mb[v]);
    | while (v) mb[v] = mb[p[v]], v = p[v];
}
ll KM() {
    | for (int i = 1; i <= n; i++) mb[i] = ka[i] = kb[i] = 0;
    | for (int a = 1; a <= n; a++) {
    |   | for (int b = 1; b <= n; b++) vb[b] = 0;
    |   | Bfs(a);
    | }
    | ll res = 0;
    | for (int b = 1; b <= n; b++) res += e[mb[b]][b];
    | return res;
}
// Main
int main() {
    | n = ri, m = ri;
    | for (int a = 1; a <= n; a++)
    |   | for (int b = 1; b <= n; b++) e[a][b] = -inf;
    | for (int i = 1; i <= m; i++) {
    |   | int u = ri, v = ri, w = ri;
    |   | e[u][v] = max(e[u][v], w);
    | }
    | printf("%lld\n", KM());
    | for (int u = 1; u <= n; u++) printf("%d ", mb[u]);
    | puts("");
    | return 0;
}
```

8.31 旋转卡壳

给定平面上 n 个点，求凸包直径。

第一行一个正整数 n。接下来 n 行，每行两个整数 x, y，表示一个点的坐标。

输出一行一个整数，表示答案的平方。

对于 100% 的数据，$2 \le n \le 50000$，$|x|, |y| \le 10^4$。

```
int n;
struct node {
    | double x, y;
} p[100005], s[100005];
double d(node p1, node p2)  //两点间距离
{
    | double ans = (p2.y - p1.y) * (p2.y - p1.y) + (p2.x -
        ↪ p1.x) * (p2.x - p1.x);
    | if (ans > 0) {
    |   | return ans;
    | } else {
    |   | return ans * (-1);
    | }
}
double check1(node a1, node a2, node b1,
    |   |   |   |   |   |   | node b2)  //检查叉积是否大于0，如果是a就
                            ↪ 逆时针转到b
//因为ACxAB和ABxAC的正负值是不同的
{
    | return (a2.x - a1.x) * (b2.y - b1.y) - (b2.x - b1.x) *
        ↪ (a2.y - a1.y);
}
double check2(
    |   | node a1, node a2, node b1,
    |   | node
    |   |   |   | b2)  //用于在构造凸包时进行判断，如果在同一直线上，
                    ↪ 就返回两线段长度的差，显然如果s[tot] 此时较短，
                    ↪ 那么它就会被弹出，由p[i] 取而代之
{
    | double ans = (a2.x - a1.x) * (b2.y - b1.y) - (b2.x -
        ↪ b1.x) * (a2.y - a1.y);
    | if (ans == 0) {
    |   | return d(a1, a2) - d(b1, b2);
    | }
    | return ans;
}
bool cmp(node p1, node p2)  //排序函数，按照极角排序
{
    | double tmp = check1(p[1], p1, p[1], p2);
    | if (tmp > 0)  // p1转到p2所以p1在前
    |   | return 1;
    | if (tmp == 0 &&
    |   |   | d(p[0], p1) < d(p[0], p2))  //虽然角度相同但是p1距
                ↪ 离更远，也是p1在前
    |   | return 1;
    | return 0;
}
double mianji(node a, node b,
    |   |   |   |   |   |   | node c) {  //求三点组成的三角形的面积 蒟
                            ↪ 蒻只会用割补法
    | double xi = min(a.x, min(b.x, c.x));
    | double yi = min(a.y, min(b.y, c.y));
    | double xa = max(a.x, max(b.x, c.x));
    | double ya = max(a.y, max(b.y, c.y));
    | double ansi = (xa - xi) * (ya - yi);
    | ansi -= (max(a.x, b.x) - min(a.x, b.x)) * (max(a.y, b.y)
        ↪ - min(a.y, b.y)) / 2;
    | ansi -= (max(a.x, c.x) - min(a.x, c.x)) * (max(a.y, c.y)
        ↪ - min(a.y, c.y)) / 2;
    | ansi -= (max(c.x, b.x) - min(c.x, b.x)) * (max(c.y, b.y)
        ↪ - min(c.y, b.y)) / 2;
    | return ansi;
}
int tot = 1;
int main() {
    | scanf("%d", &n);
    | double tmp;
    | for (int i = 1; i <= n; i++) {
    |   | scanf("%lf%lf", &p[i].x, &p[i].y);
    |   | if (i != 1)  //记录最低最右的点
    |   | {
    |   |   | if (p[i].y < p[1].y || (p[i].y == p[1].y) &&
                ↪ p[i].x < p[1].x) {
    |   |   |   | tmp = p[1].y;
    |   |   |   | p[1].y = p[i].y;
    |   |   |   | p[i].y = tmp;
    |   |   |   | tmp = p[1].x;
    |   |   |   | p[1].x = p[i].x;
    |   |   |   | p[i].x = tmp;
    |   |   | }
    |   | }
    | }
    | sort(p + 2, p + 1 + n, cmp);
    | s[1] = p[1];
    | for (int i = 2; i <= n; i++) {
    |   | while (tot != 1 && (check2(s[tot - 1], s[tot],
            ↪ s[tot], p[i]) <= 0)) {
    |   |   | tot--;
    |   | }
    |   | tot++;
    |   | s[tot] = p[i];
    | }
    | s[tot + 1] = p[1];  //最后要封闭图形
    |   |   |   |   |   |   |   |   |   |   | //    for(int
                                            ↪ i=1;i<=tot;i++){
    |   |   |   |   |   |   |   |   |   |   | //      cout<<s[i].x<<"
                                            ↪ "<<s[i].y<<endl;
    |   |   |   |   |   |   |   |   |   |   | //  }
    | if (tot == 2) {     //假如只有两个点
    |   | cout << (long long)(d(s[1], s[2])) << endl;
    |   | return 0;
    | }
    | double ans = 0;
    | int num = 3;  //先确定的边是 1 2 ，那么对应点从3开始枚举
    | for (int i = 1; i <= tot; i++) {  //枚举边
    |   | ans = max(ans,
    |   |   |   |   |   |   |   | d(s[i], s[i + 1]));  //邻边取max，可以
                                ↪ 过你谷更新的三角形取邻边情况
    |   | while (mianji(s[i], s[i + 1], s[num]) <
    |   |   |   |   |   mianji(s[i], s[i + 1], s[num + 1])) {  //假
                        ↪ 如对应点能继续走
    |   |   | num++;
    |   |   | if (num ==
    |   |   |   |   | tot +
    |   |   |   |   |   |   | 1) {  //这是之前错误的地方之一，我的程序
                                ↪ 记录的编号是从1-tot,tot+1就是1号点
    |   |   |   | num = 1;  //所以假如对踵点等于tot+1，就要使其等于
                    ↪ 一，便于下次自增
```

```
    | | | }
    | | }
    | | //      cout<<i<<" "<<i+1<<" "<<num<<endl;
    | | ans = max(ans, max(d(s[i], s[num]), d(s[i + 1],
        ↪ s[num])));
  | }
  | cout << (long long)(ans) << endl;
  | return 0;
}
```

8.32 半平面交

逆时针给出 n 个凸多边形的顶点坐标，求它们交的面积。
第一行有一个整数 n，表示凸多边形的个数，以下依次描述各个多边形。第 i 个多边形的第一行包含一个整数 m_i，表示多边形的边数，以下 m_i 行每行两个整数，逆时针给出各个顶点的坐标。
输出文件仅包含一个实数，表示相交部分的面积，保留三位小数。
对于 100% 的数据：$2 \le n \le 10$，$3 \le m_i \le 50$，每维坐标为 $[-1000, 1000]$ 内的整数。

```
#define db double
int n, cnt, tot, top, back;
db ans;
const db eps =
  | | 1e-7;  //因为是实数范围，有精度误差，所以不能直接用“==”,
      ↪ 而是取绝对值和一个很小的值进行比对。
struct node {
  | db x, y;
  | node() {}
  | node(db _x, db _y) { x = _x, y = _y; }
  | bool operator<(const node &t) const {
  | | return y < t.y || (y == t.y && x < t.x);
  | }
  | node operator-(node &t) { return node(x - t.x, y - t.y);
    ↪ }
  | bool operator==(const node &t) const { return x == t.x
    ↪ && y == t.y; }
} _P, N[55], Ans[505];  //存储点
db CPr(node A, node B) { return A.x * B.y - A.y * B.x; }
db CPr(node A, node B, node C) { return CPr(B - A, C - A);
  ↪ }  //向量叉积
struct edge {
  | node start, end;
  | db angle;
  | edge() {}
  | edge(node A, node B) {
  | | start = A, end = B;                    //起点和终点
  | | angle = atan2((B - A).y, (B - A).x);  //极角
  | }
  | bool operator<(const edge &t) const {
  | | if (fabs(angle - t.angle) <= eps)
  | | | return CPr(start, t.start, t.end) > 0;  //极角相同
      ↪ 比位置
  | | return angle < t.angle;                  //否则比极角
  | }
} e[505], dq[505];  //存储向量
db S1, S2;
node getnode(edge A, edge B) {
  | S1 = CPr(A.start, B.end, A.end);
  | S2 = CPr(A.start, B.start, A.end);
  | return node((S1 * B.start.x - S2 * B.end.x) / (S1 - S2),
  | | | | | | | (S1 * B.start.y - S2 * B.end.y) / (S1
      ↪ - S2));
}
bool ch(edge A, edge B, edge C) {
  | _P = getnode(B, C);
  | return CPr(_P, A.start, A.end) < 0;
}  //求交点
signed main() {
  | scanf("%d", &n);
  | for (int i = 1, m; i <= n; i++) {
  | | scanf("%d", &m);
  | | for (int j = 1; j <= m; j++) scanf("%lf%lf", &N[j].x,
      ↪ &N[j].y);
  | | for (int j = 1; j <= m; j++)
  | | | e[++cnt] = edge(N[j], N[j % m + 1]);  //读点，构建
      ↪ 向量
  | }
  | sort(e + 1, e + cnt + 1);  //排序
  | tot = 1;
  | for (int i = 2; i <= cnt; i++)
  | | if (fabs(e[i].angle - e[i - 1].angle) > eps) e[++tot]
      ↪ = e[i];  //去重
  | top = 2, back = 1;
  | dq[1] = e[1];
  | dq[2] = e[2];
  | for (int i = 3; i <= tot; i++) {
  | | while (back < top && ch(e[i], dq[top], dq[top - 1]))
      ↪ top--;
  | | while (back < top && ch(e[i], dq[back], dq[back +
      ↪ 1])) back++;
  | | dq[++top] = e[i];  //增量
  | }
  | while (back < top && ch(dq[back], dq[top - 1], dq[top]))
    ↪ top--;
  | while (back < top && ch(dq[top], dq[back], dq[back +
    ↪ 1]))
  | | back++;  //弹出不合法的向量
  | for (int i = back; i < top; i++)
  | | Ans[i - back + 1] = getnode(dq[i], dq[i + 1]);  //求
      ↪ 交点
  | if (top - back > 1) Ans[top - back + 1] =
    ↪ getnode(dq[top], dq[back]);
  | tot = top - back + 1;
  | for (int i = 1; i <= tot; i++) ans += CPr(Ans[i], Ans[i
    ↪ % tot + 1]);  //算面积
  | printf("%.3lf", fabs(ans) / 2);
  | return 0;
}
```

8.33 最小斯坦纳树

给定一个包含 n 个结点和 m 条带权边的无向连通图 $G = (V, E)$。
再给定包含 k 个结点的点集 S，选出 G 的子图 $G' = (V', E')$，使得：
1. $S \subseteq V'$；
2. G' 为连通图；
3. E' 中所有边的权值和最小。
你只需要求出 E' 中所有边的权值和。
第一行：三个整数 n, m, k，表示 G 的结点数、边数和 S 的大小。
接下来 m 行：每行三个整数 u, v, w，表示编号为 u, v 的点之间有一条权值为 w 的无向边。
接下来一行：k 个互不相同的正整数，表示 S 的元素。
输出一行：一个整数，表示 E' 中边权和的最小值。
对于 100% 的数据，$1 \le n \le 100$，$1 \le m \le 500$，$1 \le k \le 10$，$1 \le u, v \le n$，$1 \le w \le 10^6$。
保证给出的无向图连通，但 **可能** 存在重边和自环。

```
template <typename T>
void checkmax(T &x, T y) {
  | if (x < y) x = y;
}
template <typename T>
void checkmin(T &x, T y) {
  | if (x > y) x = y;
}
int n, m, k, dp[1024][101];
std::vector<std::pair<int, int>> g[101];
bool vis[101];
void Spfa(int S) {
  | std::queue<int> q;
  | for (int i = 0; i < n; i++)
  | | if (dp[S][i] != 0x3f3f3f3f) q.emplace(i), vis[i] =
      ↪ true;
  | while (!q.empty()) {
  | | int u = q.front();
  | | q.pop(), vis[u] = false;
  | | for (auto &&[v, w] : g[u])
  | | | if (dp[S][u] + w < dp[S][v]) {
  | | | | dp[S][v] = dp[S][u] + w;
  | | | | if (!vis[v]) q.emplace(v), vis[v] = true;
  | | | }
  | }
}
int main(int argc, char const *argv[]) {
  | std::ios_base::sync_with_stdio(false);
  | std::cin.tie(nullptr), std::cout.tie(nullptr);
  | std::cin >> n >> m >> k;
  | for (int i = 1; i <= m; i++) {
  | | int u, v, w;
  | | std::cin >> u >> v >> w, u--, v--;
  | | g[u].emplace_back(v, w), g[v].emplace_back(u, w);
  | }
  | std::memset(dp, 0x3f, sizeof(dp));
  | for (int i = 0; i < k; i++) {
```

```
    int x;
    std::cin >> x, x--;
    dp[1 << i][x] = 0;
  }
  for (int S = 1; S < (1 << k); S++) {
    for (int T = S & (S - 1); T; T = (T - 1) & S) {
      if (T < (S ^ T)) break;
      for (int i = 0; i < n; i++) checkmin(dp[S][i],
        ↪ dp[T][i] + dp[T ^ S][i]);
    }
    Spfa(S);
  }
  std::cout << *std::min_element(dp[(1 << k) - 1], dp[(1
    ↪ << k) - 1] + n);
  return 0;
}
```

8.34 李超线段树

要求在平面直角坐标系下维护两个操作：
1. 在平面上加入一条线段。记第 i 条被插入的线段的标号为 i。2. 给定一个数 k，询问与直线 $x = k$ 相交的线段中，交点纵坐标最大的线段的编号。
本题输入强制在线。
输入的第一行是一个整数 n，代表操作的个数。
接下来 n 行，每行若干个用空格隔开的整数，第 $i+1$ 行的第一个整数为 op，代表第 i 次操作的类型。
若 $op = 0$，则后跟一个整数 k，代表本次操作为查询所所有与直线 $x = (k + lastans - 1) \bmod 39989 + 1$ 相交的线段中，交点纵坐标最大的线段编号。
若 $op = 1$，则后跟四个整数 x_0, y_0, x_1, y_1，记 $x_i' = (x_i + lastans - 1) \bmod 39989 + 1$，$y_i' = (y_i + lastans - 1) \bmod 10^9 + 1$。本次操作为插入一条两端点分别为 (x_0', y_0')，(x_1', y_1') 的线段。
其中 $lastans$ 为上次询问的答案，初始时，$lastans = 0$。
对于每次查询，输出一行一个整数，代表交点纵坐标最大的线段的编号。若不存在任何一条线段与查询直线有交，则输出 0；若有多条线段与查询直线的交点纵坐标都是最大的，则输出编号最小的线段，同时 $lastans$ 也应更新为编号最小的一条线段。
对于 100% 的数据，保证 $1 \le n \le 10^5$，$1 \le k, x_0, x_1 \le 39989$，$1 \le y_0, y_1 \le 10^9$。

```
#define MOD1 39989
#define MOD2 1000000000
#define MAXT 40000
using namespace std;
typedef pair<double, int> pdi;
const double eps = 1e-9;
int cmp(double x, double y) {
  if (x - y > eps) return 1;
  if (y - x > eps) return -1;
  return 0;
}
struct line {
  double k, b;
} p[100005];
int s[160005];
int cnt;
double calc(int id, int d) { return p[id].b + p[id].k * d;
  ↪ }
void add(int x0, int y0, int x1, int y1) {
  cnt++;
  if (x0 == x1)  // 特判直线斜率不存在的情况
    p[cnt].k = 0, p[cnt].b = max(y0, y1);
  else
    p[cnt].k = 1.0 * (y1 - y0) / (x1 - x0), p[cnt].b = y0
      ↪ - p[cnt].k * x0;
}
void upd(int root, int cl, int cr, int u) {  // 对线段完全覆
  ↪ 盖到的区间进行修改
  int &v = s[root], mid = (cl + cr) >> 1;
  int bmid = cmp(calc(u, mid), calc(v, mid));
  if (bmid == 1 || (!bmid && u < v)) swap(u, v);
  int bl = cmp(calc(u, cl), calc(v, cl)), br = cmp(calc(u,
    ↪ cr), calc(v, cr));
  if (bl == 1 || (!bl && u < v)) upd(root << 1, cl, mid,
    ↪ u);
  if (br == 1 || (!br && u < v)) upd(root << 1 | 1, mid +
    ↪ 1, cr, u);
}
void update(int root, int cl, int cr, int l, int r,
  int u) {  // 定位插入线段完全覆盖到的区间
  if (l <= cl && cr <= r) {
    upd(root, cl, cr, u);
    return;
  }
  int mid = (cl + cr) >> 1;
  if (l <= mid) update(root << 1, cl, mid, l, r, u);
  if (mid < r) update(root << 1 | 1, mid + 1, cr, l, r,
    ↪ u);
}
pdi pmax(pdi x, pdi y) {  // pair max函数
  if (cmp(x.first, y.first) == -1)
    return y;
  else if (cmp(x.first, y.first) == 1)
    return x;
  else
    return x.second < y.second ? x : y;
}
pdi query(int root, int l, int r, int d) {  // 查询
  if (r < d || d < l) return {0, 0};
  int mid = (l + r) >> 1;
  double res = calc(s[root], d);
  if (l == r) return {res, s[root]};
  return pmax({res, s[root]},
    pmax(query(root << 1, l, mid, d), query(root < < 1 |
      ↪ 1, mid + 1, r, d)));
}
int main() {
  ios::sync_with_stdio(false);
  int n, lastans = 0;
  cin >> n;
  while (n--) {
    int op;
    cin >> op;
    if (op == 1) {
      int x0, y0, x1, y1;
      cin >> x0 >> y0 >> x1 >> y1;
      x0 = (x0 + lastans - 1 + MOD1) % MOD1 + 1,
      x1 = (x1 + lastans - 1 + MOD1) % MOD1 + 1;
      y0 = (y0 + lastans - 1 + MOD2) % MOD2 + 1,
      y1 = (y1 + lastans - 1 + MOD2) % MOD2 + 1;
      if (x0 > x1) swap(x0, x1), swap(y0, y1);
      add(x0, y0, x1, y1);
      update(1, 1, MOD1, x0, x1, cnt);
    } else {
      int x;
      cin >> x;
      x = (x + lastans - 1 + MOD1) % MOD1 + 1;
      cout << (lastans = query(1, 1, MOD1, x).second) <<
        ↪ endl;
    }
  }
  return 0;
}
```

8.35 动态树（LCT）

给定 n 个点以及每个点的权值，要你处理接下来的 m 个操作。操作有四种，操作从 0 到 3 编号。点从 1 到 n 编号。
- `0 x y` 代表询问从 x 到 y 的路径上的点的权值的 xor 和。保证 x 到 y 是联通的。- `1 x y` 代表连接 x 到 y，若 x 到 y 已经联通则无需连接。- `2 x y` 代表删除边 (x, y)，不保证边 (x, y) 存在。- `3 x y` 代表将点 x 上的权值变成 y。
第一行两个整数，分别为 n 和 m，代表点数和操作数。
接下来 n 行，每行一个整数，第 $(i+1)$ 行的整数 a_i 表示节点 i 的权值。
接下来 m 行，每行三个整数，分别代表操作类型和操作所需的量。
对于每一个 0 号操作，你须输出一行一个整数，表示 x 到 y 的路径上点权的 xor 和。
- $1 \le n \le 10^5$，$1 \le m \le 3 \times 10^5$，$1 \le a_i \le 10^9$。- 对于操作 0, 1, 2，保证 $1 \le x, y \le n$。- 对于操作 3，保证 $1 \le x \le n$，$1 \le y \le 10^9$。

```
#define R register int
#define I inline void
#define G         \
  if (++ip == ie) \
    if (fread(ip = buf, 1, SZ, stdin))
#define lc c[x][0]
#define rc c[x][1]
using namespace std;
const int SZ = 1 << 19, N = 3e5 + 9;
char buf[SZ], *ie = buf + SZ, *ip = ie - 1;
inline int in() {
  G;
  while (*ip < '-') G;
  R x = *ip & 15;
```

```
  | G;
  | while (*ip > '-') {
  |   | x *= 10;
  |   | x += *ip & 15;
  |   | G;
  | }
  | return x;
}
int f[N], c[N][2], v[N], s[N], st[N];
bool r[N];
inline bool nroot(R x) {  //判断节点是否为一个Splay的根（与普通Splay的区别1）
  | return c[f[x]][0] == x || c[f[x]][1] == x;
}  //原理很简单，如果连的是轻边，他的父亲的儿子里没有它
I pushup(R x) {  //上传信息
  | s[x] = s[lc] ^ s[rc] ^ v[x];
}
I pushr(R x) {
  | R t = lc;
  | lc = rc;
  | rc = t;
  | r[x] ^= 1;
}  //翻转操作
I pushdown(R x) {  //判断并释放懒标记
  | if (r[x]) {
  |   | if (lc) pushr(lc);
  |   | if (rc) pushr(rc);
  |   | r[x] = 0;
  | }
}
I rotate(R x) {  //一次旋转
  | R y = f[x], z = f[y], k = c[y][1] == x, w = c[x][!k];
  | if (nroot(y)) c[z][c[z][1] == y] = x;
  | c[x][!k] = y;
  | c[y][k] =
  |   |   | w;  //额外注意if(nroot(y)) 语句，此处不判断会引起致命错误（与普通Splay的区别2）
  | if (w) f[w] = y;
  | f[y] = x;
  | f[x] = z;
  | pushup(y);
}
I splay(
  |   | R x) {  //只传了一个参数，因为所有操作的目标都是该Splay的根（与普通Splay的区别3）
  | R y = x, z = 0;
  | st[++z] =
  |   |   | y;  // st为栈，暂存当前点到根的整条路径，pushdown时一定要从上往下放标记（与普通Splay的区别4）
  | while (nroot(y)) st[++z] = y = f[y];
  | while (z) pushdown(st[z--]);
  | while (nroot(x)) {
  |   | y = f[x];
  |   | z = f[y];
  |   | if (nroot(y)) rotate((c[y][0] == x) ^ (c[z][0] == y) ? x : y);
  |   | rotate(x);
  | }
  | pushup(x);
}
/*当然了，其实利用函数堆栈也很方便，代替上面的手工栈，就像这样
I pushall(R x){
  |   |   |   |   |   |   |   | if(nroot(x))pushall(f[x]);
  |   |   |   |   |   |   |   | pushdown(x);
}*/
I access(R x) {  //访问
  | for (R y = 0; x; x = f[y = x]) splay(x), rc = y, pushup(x);
}
I makeroot(R x) {  //换根
  | access(x);
  | splay(x);
  | pushr(x);
}
int findroot(R x) {  //找根（在真实的树中的）
  | access(x);
  | splay(x);
  | while (lc) pushdown(x), x = lc;
  | splay(x);
  | return x;
}
I split(R x, R y) {  //提取路径
  | makeroot(x);
  | access(y);
  | splay(y);
}
I link(R x, R y) {  //连边
  | makeroot(x);
  | if (findroot(y) != x) f[x] = y;
}
I cut(R x, R y) {  //断边
  | makeroot(x);
  | if (findroot(y) == x && f[y] == x && !c[y][0]) {
  |   | f[y] = c[x][1] = 0;
  |   | pushup(x);
  | }
}
int main() {
  | R n = in(), m = in();
  | for (R i = 1; i <= n; ++i) v[i] = in();
  | while (m--) {
  |   | R type = in(), x = in(), y = in();
  |   | switch (type) {
  |   |   | case 0:
  |   |   |   | split(x, y);
  |   |   |   | printf("%d\n", s[y]);
  |   |   |   | break;
  |   |   | case 1:
  |   |   |   | link(x, y);
  |   |   |   | break;
  |   |   | case 2:
  |   |   |   | cut(x, y);
  |   |   |   | break;
  |   |   | case 3:
  |   |   |   | splay(x);
  |   |   |   | v[x] = y;  //先把x转上去再改，不然会影响Splay信息的正确性
  |   | }
  | }
  | return 0;
}
```

8.36 扩展卢卡斯定理/exLucas

求

$$C_n^m \bmod p$$

其中 C 为组合数。
输入一行三个整数 n, m, p，含义由题所述。
输出一行一个整数，表示答案。
对于 100% 的数据，$1 \le m \le n \le 10^{18}$，$2 \le p \le 10^6$，**不保证** p 是质数。

```
void exgcd(ll a, ll b, ll &x, ll &y) {
  | if (!b) return (void)(x = 1, y = 0);
  | exgcd(b, a % b, x, y);
  | ll tmp = x;
  | x = y;
  | y = tmp - a / b * y;
}
ll gcd(ll a, ll b) {
  | if (b == 0) return a;
  | return gcd(b, a % b);
}
inline ll INV(ll a, ll p) {
  | ll x, y;
  | exgcd(a, p, x, y);
  | return (x + p) % p;
}
inline ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
inline ll mabs(ll x) { return (x > 0 ? x : -x); }
inline ll fast_mul(ll a, ll b, ll p) {
  | ll t = 0;
  | a %= p;
  | b %= p;
  | while (b) {
  |   | if (b & 1LL) t = (t + a) % p;
  |   | b >>= 1LL;
  |   | a = (a + a) % p;
  | }
  | return t;
}
inline ll fast_pow(ll a, ll b, ll p) {
  | ll t = 1;
```

```
  | a %= p;
  | while (b) {
  |  | if (b & 1LL) t = (t * a) % p;
  |  | b >>= 1LL;
  |  | a = (a * a) % p;
  | }
  | return t;
}
inline ll F(ll n, ll P, ll PK) {
  | if (n == 0) return 1;
  | ll rou = 1;  //循环节
  | ll rem = 1;  //余项
  | for (ll i = 1; i <= PK; i++) {
  |  | if (i % P) rou = rou * i % PK;
  | }
  | rou = fast_pow(rou, n / PK, PK);
  | for (ll i = PK * (n / PK); i <= n; i++) {
  |  | if (i % P) rem = rem * (i % PK) % PK;
  | }
  | return F(n / P, P, PK) * rou % PK * rem % PK;
}
inline ll G(ll n, ll P) {
  | if (n < P) return 0;
  | return G(n / P, P) + (n / P);
}
inline ll C_PK(ll n, ll m, ll P, ll PK) {
  | ll fz = F(n, P, PK);
  | ll fm1 = INV(F(m, P, PK), PK);
  | ll fm2 = INV(F(n - m, P, PK), PK);
  | ll mi = fast_pow(P, G(n, P) - G(m, P) - G(n - m, P),
    ↪ PK);
  | return fz * fm1 % PK * fm2 % PK * mi % PK;
}
ll A[1001], B[1001];
// x=B(mod A)
inline ll exLucas(ll n, ll m, ll P) {
  | ll ljc = P, tot = 0;
  | for (ll tmp = 2; tmp * tmp <= P; tmp++) {
  |  | if (!(ljc % tmp)) {
  |  |  | ll PK = 1;
  |  |  | while (!(ljc % tmp)) {
  |  |  |  | PK *= tmp;
  |  |  |  | ljc /= tmp;
  |  |  | }
  |  |  | A[++tot] = PK;
  |  |  | B[tot] = C_PK(n, m, tmp, PK);
  |  | }
  | }
  | if (ljc != 1) {
  |  | A[++tot] = ljc;
  |  | B[tot] = C_PK(n, m, ljc, ljc);
  | }
  | ll ans = 0;
  | for (ll i = 1; i <= tot; i++) {
  |  | ll M = P / A[i], T = INV(M, A[i]);
  |  | ans = (ans + B[i] * M % P * T % P) % P;
  | }
  | return ans;
}
signed main() {
  | ll n = read(), m = read(), P = read();
  | printf("%lld\n", exLucas(n, m, P));
  | return 0;
}
```

8.37 分治 FFT

给定序列 $g_{1\dots n-1}$，求序列 $f_{0\dots n-1}$。

其中 $f_i=\sum_{j=1}^{i} f_{i-j}g_j$，边界为 $f_0=1$。

答案对 998244353 取模。

输入第一行一个整数 n。

第二行 $n-1$ 个整数 $g_{1\dots n-1}$。

输出一行 n 个整数，表示 $f_{0\dots n-1}$ 对 998244353 取模后的值。

$2\le n\le 10^5$，$0\le g_i<998244353$。

```
const int maxlogn = 18;
const int maxn = (1 << maxlogn) | 1;
const int G0 = 15311432;
const int kcz = 998244353;
int n, rev[maxn];
ll G[2][24], f[maxn], g[maxn], a[maxn], b[maxn];
void gcd(ll a, ll b, ll &x, ll &y) {
  | if (!b)
  |  | x = 1, y = 0;
  | else
  |  | gcd(b, a % b, y, x), y -= x * (a / b);
}
inline ll inv(ll a) {
  | ll x, y;
  | gcd(a, kcz, x, y);
  | return (x + kcz) % kcz;
}
inline void calcrev(int logn) {
  | register int i;
  | rev[0] = 0;
  | for (i = 1; i < (1 << logn); i++)
  |  | rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (logn -
    ↪ 1));
}
inline void FFT(ll *a, int logn, int flag) {
  | register int i, j, k, mid;
  | register ll t1, t2, t;
  | for (i = 0; i < (1 << logn); i++)
  |  | if (rev[i] < i) swap(a[rev[i]], a[i]);
  | for (i = 1; i <= logn; i++)
  |  | for (mid = 1 << (i - 1), j = 0; j < (1 << logn); j +=
    ↪ 1 << i)
  |  |  | for (k = 0, t = 1; k < mid; k++, t = t * G[flag]
    ↪ [i] % kcz) {
  |  |  |  | t1 = a[j | k], t2 = t * a[j | k | mid];
  |  |  |  | a[j | k] = (t1 + t2) % kcz, a[j | k | mid] =
    ↪ (t1 - t2) % kcz;
  |  |  | }
}
void solve(int l, int r, int logn) {
  | if (logn <= 0) return;
  | if (l >= n) return;
  | int mid = (l + r) >> 1, i;
  | ll t = inv(r - l);
  | solve(l, mid, logn - 1);  // 计算左区间
  | calcrev(logn);
  | memset(a + (r - l) / 2, 0, sizeof(ll) * (r - l) / 2);
    ↪ // 拷贝左区间
  | memcpy(a, f + l, sizeof(ll) * (r - l) / 2);
    ↪ // 填充0
  | memcpy(b, g, sizeof(ll) * (r - l));
    ↪ // 拷贝g
  | FFT(a, logn, 0), FFT(b, logn, 0);
    ↪ // 卷积
  | for (i = 0; i < r - l; i++) a[i] = a[i] * b[i] % kcz;
  | FFT(a, logn, 1);
  | for (i = 0; i < r - l; i++) a[i] = a[i] * t % kcz;
  | for (i = (r - l) / 2; i < r - l; i++)
  |  | f[l + i] = (f[l + i] + a[i]) % kcz;  // 把卷积后的右半
    ↪ 段的数加到f数组后半段
  | // 可能你会注意到，这个卷积是 (r-l)/2的长度卷一个r-l的长度，
    ↪ 而我卷积时最终结果当作r-l的长度来存，这会不会有影响？注
    ↪ 意到超出部分是 (r-l)/2左右，根据fft的实现，超出部分是会
    ↪ 重新从0开始填的，所以只会影响结果的前半段，与后半段无关
  | solve(mid, r, logn - 1);  // 计算右区间
}
int main() {
  | int logn, i;
  | G[1][23] = inv(G[0][23] = G0);
  | for (i = 22; i >= 0; i--) {
  |  | G[0][i] = G[0][i + 1] * G[0][i + 1] % kcz;
  |  | G[1][i] = G[1][i + 1] * G[1][i + 1] % kcz;
  | }
  | scanf("%d", &n);
  | for (logn = 0; (1 << logn) < n; logn++)
  |  | ;
  | for (i = 1; i < n; i++) scanf("%lld", &g[i]);
  | for (i = 0; i < n; i++) f[i] = !i;
  | solve(0, 1 << logn, logn);
  | for (i = 0; i < n; i++) printf("%lld ", (f[i] + kcz) %
    ↪ kcz);
  | printf("\n");
  | return 0;
}
```

8.38 扩展 BSGS/exBSGS

给定 a,p,b，求满足 $a^x b \pmod p$ 的最小自然数 x。

每个测试文件中包含若干组测试数据，保证 $\sum\sqrt{p}\le 5\times 10^6$。

每组数据中，每行包含 3 个正整数 a, p, b 。
当 $a = p = b = 0$ 时，表示测试数据读入完全。
对于每组数据，输出一行。
如果无解，输出 'No Solution'，否则输出最小自然数解。
对于 100% 的数据，$1 \le a, p, b 10^9$ 或 $a = p = b = 0$

```cpp
#define int long long
int a, p, b;
map<int, int> f;
int gcd(int a, int b)  //最大公约数
{
  if (!b) return a;
  return gcd(b, a % b);
}
void exgcd(int a, int b, int &x, int &y)  //扩欧
{
  if (!b)
    x = 1, y = 0;
  else {
    exgcd(b, a % b, x, y);
    int t = x;
    x = y;
    y = t - a / b * y;
  }
}
int inv(int a, int b)  //逆元
{
  int x, y;
  exgcd(a, b, x, y);
  return (x % b + b) % b;
}
int mypow(int a, int x, int p) {
  int s = 1;
  while (x) {
    if (x & 1) s = s * a % p;
    a = a * a % p;
    x >>= 1;
  }
  return s;
}
int bsgs(int a, int b, int p)  // BSGS算法
{
  f.clear();
  int m = ceil(sqrt(p));
  b %= p;
  for (int i = 1; i <= m; i++) {
    b = b * a % p;
    f[b] = i;
  }
  int tmp = mypow(a, m, p);
  b = 1;
  for (int i = 1; i <= m; i++) {
    b = b * tmp % p;
    if (f[b]) return (i * m - f[b] + p) % p;
  }
  return -1;
}
int exbsgs(int a, int b, int p) {
  if (b == 1 || p == 1) return 0;  //特殊情况，x=0时最小解
  int g = gcd(a, p), k = 0, na = 1;
  while (g > 1) {
    if (b % g != 0) return -1;  //无法整除则无解
    k++;
    b /= g;
    p /= g;
    na = na * (a / g) % p;
    if (na == b) return k;  // na=b说明前面的a的次数为0，只需要返回k
    g = gcd(a, p);
  }
  int f = bsgs(a, b * inv(na, p) % p, p);
  if (f == -1) return -1;
  return f + k;
}
signed main() {
  cin >> a >> p >> b;
  while (a || b || p) {
    a %= p;
    b %= p;
    int t = exbsgs(a, b, p);
    if (t == -1)
      cout << "No Solution" << endl;
    else
      cout << t << endl;
    cin >> a >> p >> b;
  }
  return 0;
}
```

8.39 三维偏序

有 n 个元素，第 i 个元素有 a_i, b_i, c_i 三个属性，设 $f(i)$ 表示满足 $a_j \le a_i$ 且 $b_j \le b_i$ 且 $c_j \le c_i$ 且 $j \ne i$ 的 j 的数量。
对于 $d \in [0, n)$，求 $f(i) = d$ 的数量。
输入第一行两个整数 n, k，表示元素数量和最大属性值。
接下来 n 行，每行三个整数 a_i, b_i, c_i，分别表示三个属性值。
输出 n 行，第 $d + 1$ 行表示 $f(i) = d$ 的 i 的数量。
$1 \le n \le 10^5$，$1 \le a_i, b_i, c_i \le k \le 2 \times 10^5$。

```cpp
const int MAXN = 1e5 + 5;
const int MAXK = 2e5;
int n, k, cnt[MAXN];
struct Data {
  int x, y, z;
  int operator<(const Data &o) const {
    return x != o.x ? (x < o.x) : (y != o.y ? (y < o.y) : (z < o.z));
  }
  int operator==(const Data &o) const {
    return x == o.x && y == o.y && z == o.z;
  }
} data[MAXN];
struct Seg {
  struct Node {
    int val;
    Node *ch[2];
    Node(int val = 0) : val(val) { ch[0] = ch[1] = NULL; }
  };
  Node *rt;
  Seg() { rt = NULL; }
  void Modify(Node *&now, int pos, int val, int nl, int nr) {
    if (!now) now = new Node();
    if (nl == nr) {
      now->val += val;
      return;
    }
    int mid = nl + nr >> 1;
    if (pos <= mid)
      Modify(now->ch[0], pos, val, nl, mid);
    else
      Modify(now->ch[1], pos, val, mid + 1, nr);
    now->val =
        (now->ch[0] ? now->ch[0]->val : 0) +
        (now->ch[1] ? now->ch[1]->val : 0);
  }
  int Query(Node *now, int l, int r, int nl, int nr) {
    if (!now) return 0;
    if (l == nl && r == nr) return now->val;
    int mid = nl + nr >> 1;
    if (r <= mid)
      return Query(now->ch[0], l, r, nl, mid);
    else if (l > mid)
      return Query(now->ch[1], l, r, mid + 1, nr);
    return Query(now->ch[0], l, mid, nl, mid) +
           Query(now->ch[1], mid + 1, r, mid + 1, nr);
  }
};
/*
Seg tree[MAXK * 4 + 5];

void Modify(int now, int posx, int posy, int val, int nl = 1, int nr = MAXK) {
                        tree[now].Modify(tree[now].rt, posy, val);
                        if (nl == nr) return;
                        int mid = nl + nr >> 1;
                        if (posx <= mid) Modify(now << 1, posx, posy, val, nl, mid);
                        else Modify(now << 1 | 1, posx, posy, val, mid + 1, nr);
}

```

```
int Query(int now, int xl, int xr, int yl, int yr, int nl = 1, int nr = MAXK) {
  if (xl == nl && xr == nr) return tree[now].Query(tree[now].rt, yl, yr); int mid = nl + nr >> 1; if (xr <= mid) return Query(now << 1, xl, xr, yl, yr, nl, mid); else if (nl > mid) return Query(now << 1 | 1, xl, xr, yl, yr, mid + 1, nr); return Query(now << 1, xl, mid, yl, yr, nl, mid) + Query(now << 1 | 1, mid + 1, xr, yl, yr, mid + 1, nr);
}
*/
Seg tree[MAXK + 5];
int LB(int x) { return x & (-x); }
void Modify(int posx, int posy, int val) {
  for (int i = posx; i <= k; i += LB(i))
    tree[i].Modify(tree[i].rt, posy, val, 1, k);
}
int Query(int x, int y) {
  int ret = 0;
  for (int i = x; i; i -= LB(i)) ret += tree[i].Query(tree[i].rt, 1, y, 1, k);
  return ret;
}
int main() {
  cin >> n >> k;
  for (int i = 1; i <= n; i++) cin >> data[i].x >> data[i].y >> data[i].z;
  sort(data + 1, data + n + 1);
  int sum = 1;
  for (int i = 1; i <= n; i++) {
    if (data[i + 1] == data[i]) {
      sum++;
      continue;
    }
    Modify(data[i].y, data[i].z, sum);
    int res = Query(data[i].y, data[i].z);
    cnt[res] += sum;
    sum = 1;
  }
  for (int i = 1; i <= n; i++) cout << cnt[i] << endl;
  return 0;
}
```

8.40 舞蹈链（DLX）

给定一个 N 行 M 列的矩阵，矩阵中每个元素要么是 1，要么是 0。
你需要在矩阵中挑选出若干行，使得对于矩阵的每一列 j，在你挑选的这些行中，有且仅有一行的第 j 个元素为 1。
第一行两个数 N, M。
接下来 N 行，每行 M 个数字 0 或 1，表示这个矩阵。
一行输出若干个数表示答案，两个数之间用空格隔开，输出任一可行方案均可，顺序随意。
若无解，输出 'No Solution'。
对于 100% 的数据，$N, M \leq 500$，保证矩阵中 1 的数量不超过 5000 个。

```
const int mx = 250501;  // n*m+m<=250500
inline int Read() {
  int x = 0;
  char c = getchar();
  while (c > '9' || c < '0') c = getchar();
  while (c >= '0' && c <= '9') x = x * 10 + c - '0', c = getchar();
  return x;
}
int n, m;
int cnt;
int l[mx], r[mx], u[mx], d[mx], col[mx],
    row[mx];        //每个点的左右上下指针，所在行列
int h[mx];          //每行的头结点
int s[mx];          //每列的节点数
int ansk[mx];       //选了那些集合
void init(int m) {  // m个元素
  for (register int i = 0; i <= m; i++) {
    r[i] = i + 1;
    l[i] = i - 1;
    u[i] = d[i] = i;
  }
  r[m] = 0;
  l[0] = m;
  memset(h, -1, sizeof(h));
  memset(s, 0, sizeof(s));
  cnt = m + 1;
}  //初始化
inline void link(int R, int C) {  // R行C列插入点
  s[C]++;
  row[cnt] = R;
  col[cnt] = C;
  u[cnt] = C;
  d[cnt] = d[C];
  u[d[C]] = cnt;
  d[C] = cnt;
  if (h[R] == -1)
    h[R] = r[cnt] = l[cnt] = cnt;  //该行没有点，直接加入
  else {
    r[cnt] = h[R];
    l[cnt] = l[h[R]];
    r[l[h[R]]] = cnt;
    l[h[R]] = cnt;
  }
  cnt++;
  return;
}
inline void remove(int C) {  //删除涉及C列的集合
  r[l[C]] = r[C], l[r[C]] = l[C];
  for (int i = d[C]; i != C; i = d[i]) {
    for (int j = r[i]; j != i; j = r[j]) {
      u[d[j]] = u[j];
      d[u[j]] = d[j];
      s[col[j]]--;
    }
  }
}
inline void resume(int C) {  //恢复涉及C列的集合
  for (int i = u[C]; i != C; i = u[i]) {
    for (int j = l[i]; j != i; j = l[j]) {
      u[d[j]] = j;
      d[u[j]] = j;
      s[col[j]]++;
    }
  }
  r[l[C]] = C;
  l[r[C]] = C;
}
bool dance(int deep) {
  if (r[0] == 0) {
    register int i = 0;
    for (i = 0; i < deep; i++) printf("%d ", ansk[i]);
    return 1;
  }
  int c = r[0];
  int i, j;
  for (i = r[0]; i != 0; i = r[i])
    if (s[i] < s[c]) c = i;
  remove(c);
  for (i = d[c]; i != c; i = d[i]) {
    ansk[deep] = row[i];
    for (j = r[i]; j != i; j = r[j]) remove(col[j]);
    if (dance(deep + 1)) return 1;
    for (j = l[i]; j != i; j = l[j]) resume(col[j]);
  }
  resume(c);
  return 0;
}
int main() {
  //  freopen("cin.txt","r",stdin);
  n = Read(), m = Read();
  register int i, j;
  int f;
  init(m);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= m; j++) {
      f = Read();
      if (f) link(i, j);
    }
  }
  if (!dance(0)) printf("No Solution!");
  return 0;
}
```

8.41 最小树形图

给定包含 n 个结点，m 条有向边的一个图。试求一棵以结点 r 为根的最小树形图，并输出最小树形图每条边的权值之和，如果没有以 r 为根的最小

树形图，输出 -1。
第一行包含三个整数 n, m, r，意义同题目所述。
接下来 m 行，每行包含三个整数 u, v, w，表示图中存在一条从 u 指向 v 的权值为 w 的有向边。
如果原图中存在以 r 为根的最小树形图，就输出最小树形图每条边的权值之和，否则输出 -1。
对于所有数据，$1 \le u, v \le n \le 100, 1 \le m \le 10^4, 1 \le w \le 10^6$。

```
struct Edge {
  int u, v, w;
} e[M];
const int inf = 2e9;
int n, m, root;
int pre[N], ine[N];
int vis[N], id[N];
int zhuliu() {
  int ans = 0;
  while (1) {
    for (ri i = 1; i <= n; ++i) ine[i] = inf;  // 初始化
    for (ri i = 1; i <= m; ++i) {
      int u = e[i].u, v = e[i].v;
      if (u != v && e[i].w < ine[v])  // 遍历所有边，对每个点找到最小的入边
        ine[v] = e[i].w, pre[v] = u;
    }
    for (ri i = 1; i <= n; ++i)  // 判定无解
      if (i != root && ine[i] == inf) return -1;
    int cnt = 0;
    for (ri i = 1; i <= n; ++i) vis[i] = id[i] = 0;
    for (ri i = 1; i <= n; ++i) {
      if (i == root) continue;
      ans += ine[i];
      int v = i;
      while (vis[v] != i && !id[v] && v != root)  // 找环
      {
        vis[v] = i;
        v = pre[v];
      }
      if (!id[v] && v != root) {
        id[v] = ++cnt;  // 把环上的店标记为同一点
        for (ri u = pre[v]; u != v; u = pre[u]) id[u] = cnt;
      }
    }
    if (cnt == 0) break;  // 无环，得到解
    for (ri i = 1; i <= n; ++i)
      if (!id[i]) id[i] = ++cnt;
    for (ri i = 1; i <= m; ++i) {
      int u = e[i].u, v = e[i].v;
      e[i].u = id[u], e[i].v = id[v];
      if (id[u] != id[v]) e[i].w -= ine[v];  // 修改边权
    }
    root = id[root];
    n = cnt;
  }
  return ans;
}
```

8.42 扩展中国剩余定理（EXCRT）

给定 n 组非负整数 a_i, b_i，求解关于 x 的方程组的最小非负整数解。

$$\begin{cases} x \equiv b_1 \pmod{a_1} \\ x \equiv b_2 \pmod{a_2} \\ \dots \\ x \equiv b_n \pmod{a_n} \end{cases}$$

输入第一行包含整数 n。
接下来 n 行，每行两个非负整数 a_i, b_i。
输出一行，为满足条件的最小非负整数 x。
对于 100% 的数据，$1 \le n \le 10^5$，$1 \le b_i, a_i \le 10^{12}$，保证所有 a_i 的最小公倍数不超过 10^{18}。数据保证有解。

```
from functools import reduce

def gcd(a, b):
  if b==0: return a
  return gcd(b, a%b)

def lcm(a, b):
  return a * b // gcd(a,b)

def exgcd(a, b):
  if b==0: return 1, 0
  x, y = exgcd(b, a%b)
  return y, x - a//b*y

def uni(P, Q):
  r1, m1 = P
  r2, m2 = Q

  d = gcd(m1, m2)
  assert (r2-r1) % d == 0

  l1, l2 = exgcd(m1//d, m2//d)

  return (r1 + (r2-r1)//d*l1*m1) % lcm(m1, m2), lcm(m1, m2)

def CRT(eq):
  return reduce(uni, eq)

if __name__ == "__main__":
  n = int(input())
  eq = [list(map(int, input().strip().split()))[::-1] for x in range(n)]
  print(CRT(eq)[0])
```

8.43 BEST 定理

有 n 个房间，每个房间有若干把钥匙能够打开特定房间的门。
最初你在房间 1。每当你到达一个房间，你可以选择该房间的一把钥匙，前往该钥匙对应的房间，并将该钥匙丢到垃圾桶中。
你希望最终回到房间 1，且垃圾桶中有所有的钥匙。
你需要求出方案数，答案对 10^6+3 取模。两组方案不同，当且仅当使用钥匙的顺序不同。
注意，每把钥匙都是不同的。
第一行一个整数 T，表示数据组数。
对于每组数据：
第一行一个整数 n。
接下来 n 行，第 i 行描述房间 i：
首先一个数 s，表示这个房间的钥匙数目，接下来 s 个数，分别描述每把钥匙能够打开的房间的门。
对于每组数据，一行一个整数，表示答案对 10^6+3 取模后的值。
对于 100% 的数据，$1 \le T \le 15$，$1 \le n \le 100$，$0 \le \sum s \le 2 \times 10^5$。

```
const int N = 110;
const int M = 500010;
const int P = 1000003;
typedef long long LL;
int T;
int n;
int I[M];
int fa[M];
LL res;
LL fac[M];
LL deg[M];
LL D[N][N];
LL K[N][N];
LL A[N][N];
LL Gauss() {
  LL ans = 1;
  for (int i = 1; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      while (K[j][i]) {
        LL t = K[i][i] / K[j][i];
        for (int k = i; k < n; ++k) K[i][k] = (K[i][k] - t * K[j][k]) % P;
        swap(K[i], K[j]), ans = (-ans % P + P) % P;
      }
    }
    if (K[i][i]) (ans *= K[i][i]) %= P;
  }
  return (ans % P + P) % P;
}
void clear() {
  memset(K, 0, sizeof(K));
  memset(A, 0, sizeof(A));
  memset(D, 0, sizeof(D));
  memset(I, 0, sizeof(I));
  for (int i = 1; i <= n; ++i) fa[i] = i;
}
int find_f(int x) { return x == fa[x] ? x : fa[x] = find_f(fa[x]); }
int main() {
  cin >> T;
  fac[0] = 1;
```

```
  | for (LL i = 1; i <= 500000; ++i) fac[i] = fac[i - 1] * i % P;
  | while (T--) {
  |   | scanf("%d", &n);
  |   | clear();
  |   | for (int i = 1; i <= n; ++i) {
  |   |   | int k, s;
  |   |   | cin >> s;
  |   |   | deg[i] = s;
  |   |   | for (int j = 1; j <= s; ++j) {
  |   |   |   | cin >> k;
  |   |   |   | A[i][k]++, D[k][k]++, ++I[k];
  |   |   |   | if (find_f(i) != find_f(k)) fa[find_f(i)] = find_f(k);
  |   |   | }
  |   | }
  |   | for (int i = 1; i <= n; ++i)
  |   |   | for (int j = 1; j <= n; ++j) K[i][j] = ((D[i][j] - A[i][j]) % P + P) % P;
  |   | for (int i = 1; i <= n; ++i)
  |   |   | if (I[i] != deg[i]) goto lalala;
  |   | for (int i = 1; i <= n; ++i)
  |   |   | if (find_f(i) != find_f(1) && deg[i]) goto lalala;
  |   | res = deg[1] * Gauss() % P;
  |   | for (int i = 1; i <= n; ++i)
  |   |   | if (deg[i]) (res *= fac[deg[i] - 1]) %= P;
  |   | for (int i = 2; i <= n; ++i)
  |   |   | if (find_f(i) == find_f(1)) goto lelele;
  |   | cout << fac[deg[1]] << endl;
  |   | continue;
  | lalala:
  |   | puts("0");
  |   | continue;
  | lelele:
  |   | cout << res << endl;
  | }
  | return 0;
}
```

8.44 点分树

在一片土地上有 n 个城市，通过 $n-1$ 条无向边互相连接，形成一棵树的结构，相邻两个城市的距离为 1，其中第 i 个城市的价值为 $value_i$。

不幸的是，这片土地常常发生地震，并且随着时代的发展，城市的价值也往往会发生变动。

接下来你需要在线处理 m 次操作：

‘0 x k‘ 表示发生了一次地震，震中城市为 x，影响范围为 k，所有与 x 距离不超过 k 的城市都将受到影响，该次地震造成的经济损失为所有受影响城市的价值和。

‘1 x y‘ 表示第 x 个城市的价值变成了 y 。

为了体现程序的在线性，操作中的 x、y、k 都需要异或你程序上一次的输出来解密，如果之前没有输出，则默认上一次的输出为 0 。

第一行包含两个正整数 n 和 m 。

第二行包含 n 个正整数，第 i 个数表示 $value_i$ 。

接下来 $n-1$ 行，每行包含两个正整数 u、v，表示 u 和 v 之间有一条无向边。

接下来 m 行，每行包含三个数，表示 m 次操作。

包含若干行，对于每个询问输出一行一个正整数表示答案。

对于 100% 的数据，有 $1 \le n, m \le 10^5, 1 \le u, v, x \le n, 1 \le value_i, y \le 10^4, 0 \le k \le n-1$

```
#define REP(u) for (int i = H[u], v; i, v = E[i].v; i = E[i].n)
const int maxn = 2e5 + 111, inf = 1e9 + 7;
int N, M, tot, ans, rt, sum, minn, cnt;
int val[maxn], H[maxn], sz[maxn], fa[maxn], dep[maxn], pos[maxn],
  | | ol[maxn << 1][21], lg[maxn << 1];
// val表示该城市的价值，sz表示该点子树的大小，fa表示该点点分树上的父亲节点，dep表示该点的深度
bool vis[maxn];
vector<int> C[2][maxn];
// C[0][i] 表示i子树内节点对i的贡献
// C[1][i] 表示i子树内节点对i父亲的贡献
struct edge {
  | int n, v;
} E[maxn << 1];
void add(int u, int v) {
  | E[++tot] = (edge){H[u], v};
  | H[u] = tot;
}
void dfs0(int u, int f) {
  | ol[++cnt][0] = u, pos[u] = cnt;
  | REP(u) if (v ^ f) dep[v] = dep[u] + 1, dfs0(v, u), ol[++cnt][0] = u;
}
int get_min(int a, int b) { return dep[a] < dep[b] ? a : b; }
void get_ol() {
  | for (int i = 1; i <= cnt; ++i) lg[i] = 31 - __builtin_clz(i);
  | for (int t = 1; 1 << t <= cnt; ++t)
  |   | for (int i = 1; i + (1 << t) <= cnt; ++i)
  |   |   | ol[i][t] = get_min(ol[i][t - 1], ol[i + (1 << (t - 1))][t - 1]);
}
int get_dis(int u, int v) {
  | if (pos[u] > pos[v]) swap(u, v);
  | int uu = pos[u], vv = pos[v], len = vv - uu + 1;
  | int lca = get_min(ol[uu][lg[len]], ol[vv - (1 << lg[len]) + 1][lg[len]]);
  | return dep[u] + dep[v] - 2 * dep[lca];
}
#define lowbit(x) (x & -x)
void upd(int u, int opt, int x, int addv) {
  | x++;
  | for (int i = x; i <= sz[u]; i += lowbit(i)) C[opt][u][i] += addv;
}
int qry(int u, int opt, int x) {
  | x++;
  | int res = 0;
  | x = min(x, sz[u]);
  | for (int i = x; i; i -= lowbit(i)) res += C[opt][u][i];
  | return res;
}
void find_rt(int u, int f)  //找重心
{
  | sz[u] = 1;
  | int res = 0;
  | REP(u)
  | if (f ^ v && !vis[v]) find_rt(v, u), sz[u] += sz[v], res = max(res, sz[v]);
  | res = max(res, sum - sz[u]);
  | if (res < minn) minn = res, rt = u;
}
void dfs(int u)  //建立点分树
{
  | vis[u] = 1;
  | sz[u] = sum + 1;
  | C[0][u].resize(sz[u] + 1);
  | C[1][u].resize(sz[u] + 1);
  | REP(u) if (!vis[v]) {
  |   | sum = sz[v], rt = 0, minn = inf;
  |   | find_rt(v, 0);
  |   | fa[rt] = u;
  |   | dfs(rt);
  | }
}
void modify(int u, int w) {
  | for (int i = u; i; i = fa[i]) upd(i, 0, get_dis(u, i), w);
  | for (int i = u; fa[i]; i = fa[i]) upd(i, 1, get_dis(u, fa[i]), w);
}
int main() {
  | int opt, x, y;
  | scanf("%d%d", &N, &M);
  | for (int i = 1; i <= N; ++i) scanf("%d", &val[i]);
  | for (int i = 1; i < N; ++i) scanf("%d%d", &x, &y), add(x, y), add(y, x);
  | dfs0(1, 0);
  | get_ol();
  | sum = N, minn = inf;
  | find_rt(1, 0);
  | dfs(rt);
  | for (int i = 1; i <= N; ++i) modify(i, val[i]);
  | while (M--) {
  |   | scanf("%d%d%d", &opt, &x, &y);
  |   | x ^= ans, y ^= ans;
  |   | if (!opt) {
  |   |   | ans = 0;
  |   |   | ans += qry(x, 0, y);  // x子树内到其距离为y的点对x的贡献
  |   |   | for (int i = x; fa[i]; i = fa[i]) {
```

```
        int dis = get_dis(x, fa[i]);
        if (y >= dis)
          ans += qry(fa[i], 0, y - dis) -
                 qry(i, 1, y - dis);  // x子树外到其距离为y的点
        // fa[i] 子树中除x所在子树对x的贡献 即为 fa[i] 子树对它的总贡献 减去
        // x所在子树对fa[i] 的贡献
      }
      printf("%d\n", ans);
    } else
      modify(x, y - val[x]),
      val[x] = y;  //利用差分的办法修改其对点分树上祖先的贡献
  }
  return 0;
}
```

8.45 树同构

我们把 N 个点，$N-1$ 条边的连通无向图称为树。
若将某个点作为根，从根开始遍历，则其它的点都有一个前驱，这个树就成为有根树。
对于两个树 T_1 和 T_2，如果能够把树 T_1 的所有点重新标号，使得树 T_1 和树 T_2 完全相同，那么这两个树是同构的。也就是说，它们具有相同的形态。
现在，给你 M 个无根树，请你把它们按同构关系分成若干个等价类。
第一行，一个整数 M。
接下来 M 行，每行包含若干个整数，表示一个树。第一个整数 N 表示点数。接下来 N 个整数，依次表示编号为 1 到 N 的每个点的父亲结点的编号。根节点父亲结点编号为 0。
输出 M 行，每行一个整数，表示与每个树同构的树的最小编号。
对于 100% 的数据，$1 \le N, M \le 50$。

```
#define L(i, j, k) for (int i = j, i##E = k; i <= i##E; i++)
#define R(i, j, k) for (int i = j, i##E = k; i >= i##E; i--)
const int N = 55;
const int mod = 1019260817;
const int G = 19491001;
int Pow[N];
int n, m;
struct Tree {
  int A, B;
} f[N];
bool operator==(Tree aa, Tree bb) { return aa.A == bb.A && aa.B == bb.B; }
int head[N], edge_id;
struct edge {
  int to, next;
} e[N << 1];
void add_edge(int u, int v) {
  ++edge_id, e[edge_id].to = v, e[edge_id].next = head[u], head[u] = edge_id;
}
int has[N], siz[N], dep[N], rt, rrt, rtm;
void findrt(int x, int fa) {
  siz[x] = 1;
  int maxn = 0;
  for (int i = head[x]; i; i = e[i].next) {
    int v = e[i].to;
    if (v == fa) continue;
    findrt(v, x), siz[x] += siz[v], maxn = max(maxn, siz[v]);
  }
  maxn = max(maxn, n - siz[x]);
  if (maxn < rtm)
    rtm = maxn, rt = x, rrt = 0;
  else if (maxn == rtm)
    rrt = x;
}
int tot;
pii sav[N];
void dfs(int x, int fa) {
  has[x] = 1ll * dep[x] * Pow[1] % mod, siz[x] = 1;
  for (int i = head[x]; i; i = e[i].next) {
    int v = e[i].to;
    if (v == fa) continue;
    dep[v] = dep[x] + 1, dfs(v, x);
  }
  tot = 0;
  for (int i = head[x]; i; i = e[i].next) {
    int v = e[i].to;
    if (v == fa) continue;
    sav[++tot] = mkp(has[v], siz[v]);
  }
  sort(sav + 1, sav + tot + 1);
  L(i, 1, tot)
  (has[x] += 1ll * sav[i].first * Pow[siz[x]] % mod) %= mod,
    siz[x] += sav[i].second;
}
void In(int x) {
  rtm = mod, rrt = 0;
  scanf("%d", &n);
  L(i, 1, n) {
    int v;
    scanf("%d", &v);
    if (v) add_edge(i, v), add_edge(v, i);
  }
  findrt(1, -1);
  dep[rt] = 1, dfs(rt, -1), f[x].A = has[rt];
  if (rrt) dep[rrt] = 1, dfs(rrt, -1), f[x].B = has[rrt];
  if (f[x].A < f[x].B) swap(f[x].A, f[x].B);
  L(i, 1, n) head[i] = 0;
  edge_id = 0;
}
int mian() {
  Pow[0] = 1;
  L(i, 1, 50) Pow[i] = 1ll * Pow[i - 1] * G % mod;
  scanf("%d", &m);
  L(i, 1, m) In(i);
  L(i, 1, m) L(j, 1, i) if (f[i] == f[j]) {
    printf("%d\n", j);
    break;
  }
  return 0;
}
```

8.46 杜教筛

给定一个正整数，求

$$ans_1 = \sum_{i=1}^{n} \varphi(i)$$

$$ans_2 = \sum_{i=1}^{n} \mu(i)$$

输入的第一行为一个整数，表示数据组数 T。
接下来 T 行，每行一个整数 n，表示一组询问。
对于每组询问，输出一行两个整数，分别代表 ans_1 和 ans_2。
对于全部的测试点，保证 $1 \le T \le 10$，$1 \le n < 2^{31}$。

```
#define mod 1000000007
#define maxn 5000000
ll pai[maxn + 5];
int prim[maxn + 5], cnt, mu[maxn + 5];
bool vis[maxn + 5];
unordered_map<int, int> ans_mu;
unordered_map<int, ll> ans_pai;
il void init() {
  mu[1] = pai[1] = 1;
  for (re int i = 2; i <= maxn; ++i) {
    if (!vis[i]) prim[++cnt] = i, mu[i] = -1, pai[i] = i - 1;
    for (re int j = 1; j <= cnt && prim[j] * i <= maxn; +j) {
      vis[prim[j] * i] = 1;
      if (i % prim[j] == 0) {
        pai[i * prim[j]] = pai[i] * prim[j];
        break;
      }
      pai[i * prim[j]] = pai[prim[j]] * pai[i];
      mu[i * prim[j]] = -mu[i];
    }
  }
  for (re int i = 1; i <= maxn; ++i) mu[i] += mu[i - 1], pai[i] += pai[i - 1];
}
il ll get_pai(ll x) {
  if (x <= maxn) return pai[x];
  if (ans_pai[x]) return ans_pai[x];
  ll ans = ((1ll + x) * x) / 2ll;
  for (re int l = 2, r; l <= x;
```

```
    |   |   |   l = r + 1)  //其实这里可能会爆int，可以改用
              ↪用unsigned int
    | {
    |   | r = x / (x / l);
    |   | ans -= 1ll * (r - l + 1) * get_pai(x / l);
    | }
    | return ans_pai[x] = ans;
}
il int get_mu(int x) {
    | if (x <= maxn) return mu[x];
    | if (ans_mu[x]) return ans_mu[x];
    | int ans = 1;
    | for (re int l = 2, r; l <= x; l = r + 1) {
    |   | r = x / (x / l);
    |   | ans -= (r - l + 1) * get_mu(x / l);
    | }
    | return ans_mu[x] = ans;
}
il void doit() {
    | int T = read();
    | while (T--) {
    |   | int x = read();
    |   | printf("%lld %d\n", get_pai(x), get_mu(x));
    | }
}
signed main() {
    | init(), doit();
    | return 0;
}
```

8.47 Lyndon 分解

读入一个由小写英文字母组成的字符串 s，请把这个字符串分成若干部分 $s = s_1 s_2 s_3 \cdots s_m$，使得每个 s_i 都是 Lyndon Word，且 $\forall 1 \le i < m : s_i \ge s_{i+1}$。输出 s_1 到 s_m 这些串长度的右端点的位置。位置编号为 1 到 n。

一个字符串 s 是一个 Lyndon Word，当且仅当 s 是其所有后缀中最小的字符串。

为了减小输出量，你只需要输出所有右端点的异或和。

输入一行一个长度为 n 的仅包含小写英文字母的字符串 s。

输出一行一个整数，表示所有右端点的异或和。

- 对于 100% 的数据，保证 $1 \le n \le 5 \times 10^6 + 1$。

```
char s[5000005];
int n, ans;
int main() {
    | scanf("%s", s + 1);
    | n = (int)strlen(s + 1);
    | for (int i = 1; i <= n;) {
    |   | int j = i, k = i + 1;  //初始化
    |   | while (k <= n && s[j] <= s[k]) {
    |   |   | if (s[j] < s[k])
    |   |   |   | j = i;  //合并为一整个
    |   |   | else
    |   |   |   | j++;  //保持循环不变式
    |   |   | k++;
    |   | }
    |   | while (i <= j)  //从v的开头重新开始
    |   | {
    |   |   | ans ^= i + k - j - 1;
    |   |   | i += k - j;
    |   | }
    | }
    | printf("%d\n", ans);
    | return 0;
}
```

8.48 线段树分裂

给出一个可重集 a（编号为 1），它支持以下操作：

‘0 p x y‘：将可重集 p 中大于等于 x 且小于等于 y 的值移动到一个新的可重集中（新可重集编号为从 2 开始的正整数，是上一次产生的新可重集的编号+1）。

‘1 p t‘：将可重集 t 中的数放入可重集 p，且清空可重集 t（数据保证在此后的操作中不会出现可重集 t）。

‘2 p x q‘：在 p 这个可重集中加入 x 个数字 q。

‘3 p x y‘：查询可重集 p 中大于等于 x 且小于等于 y 的值的个数。

‘4 p k‘：查询在 p 这个可重集中第 k 小的数，不存在时输出 ‘-1‘。

第一行两个整数 n, m，表示可重集中的数在 $1 \sim n$ 的范围内，有 m 个操作。

接下来一行 n 个整数，表示 $1 \sim n$ 这些数在 a 中出现的次数（$a_i \le m$）。

接下来的 m 行每行若干个整数,第一个数为操作的编号 $opt(0 \le opt \le 4)$，以**题目描述**为准。

依次输出每个查询操作的答案。

对于 100% 的数据，$1 \le n \le 2 \times 10^5$，$1 \le x, y, q \le m \le 2 \times 10^5$。保证数据合法。

```
const int MAXN = 200010;
int n, m, tot, cnt, seq = 1, op, x, y, z, bac[MAXN << 5],
  ↪ch[MAXN << 5][2],
    |   |   |   |   |   |   |   |   | rt[MAXN];
ll val[MAXN << 5];
int newnod() { return (cnt ? bac[cnt--] : ++tot); }
void del(int p) {
    | bac[++cnt] = p, ch[p][0] = ch[p][1] = val[p] = 0;
    | return;
}
void modify(int &p, int l, int r, int pos, int v) {
    | if (!p) {
    |   | p = newnod();
    | }
    | val[p] += v;
    | if (l == r) {
    |   | return;
    | }
    | int mid = (l + r) >> 1;
    | if (pos <= mid) {
    |   | modify(ch[p][0], l, mid, pos, v);
    | } else {
    |   | modify(ch[p][1], mid + 1, r, pos, v);
    | }
    | return;
}
ll query(int p, int l, int r, int xl, int xr) {
    | if (xr < l || r < xl) {
    |   | return 0;
    | }
    | if (xl <= l && r <= xr) {
    |   | return val[p];
    | }
    | int mid = (l + r) >> 1;
    | return query(ch[p][0], l, mid, xl, xr) + query(ch[p][1],
      ↪mid + 1, r, xl, xr);
}
int kth(int p, int l, int r, int k) {
    | if (l == r) {
    |   | return l;
    | }
    | int mid = (l + r) >> 1;
    | if (val[ch[p][0]] >= k) {
    |   | return kth(ch[p][0], l, mid, k);
    | } else {
    |   | return kth(ch[p][1], mid + 1, r, k - val[ch[p][0]]);
    | }
}
int merge(int x, int y) {
    | if (!x || !y) {
    |   | return x + y;
    | }
    | val[x] += val[y];
    | ch[x][0] = merge(ch[x][0], ch[y][0]);
    | ch[x][1] = merge(ch[x][1], ch[y][1]);
    | del(y);
    | return x;
}
void split(int x, int &y, ll k) {
    | if (x == 0) {
    |   | return;
    | }
    | y = newnod();
    | ll v = val[ch[x][0]];
    | if (k > v) {
    |   | split(ch[x][1], ch[y][1], k - v);
    | } else {
    |   | swap(ch[x][1], ch[y][1]);
    | }
    | if (k < v) {
    |   | split(ch[x][0], ch[y][0], k);
    | }
    | val[y] = val[x] - k;
    | val[x] = k;
    | return;
}
int main() {
    | scanf("%d%d", &n, &m);
    | for (int i = 1; i <= n; i++) {
```

```
    scanf("%d", &x);
    modify(rt[1], 1, n, i, x);
  }
  for (int i = 1; i <= m; i++) {
    scanf("%d", &op);
    if (op == 0) {
      scanf("%d%d%d", &x, &y, &z);
      ll k1 = query(rt[x], 1, n, 1, z), k2 =
        ↪ query(rt[x], 1, n, y, z);
      int tmp = 0;
      split(rt[x], rt[++seq], k1 - k2);
      split(rt[seq], tmp, k2);
      rt[x] = merge(rt[x], tmp);
    } else if (op == 1) {
      scanf("%d%d", &x, &y);
      rt[x] = merge(rt[x], rt[y]);
    } else if (op == 2) {
      scanf("%d%d%d", &x, &y, &z);
      modify(rt[x], 1, n, z, y);
    } else if (op == 3) {
      scanf("%d%d%d", &x, &y, &z);
      printf("%lld\n", query(rt[x], 1, n, y, z));
    } else if (op == 4) {
      scanf("%d%d", &x, &y);
      if (val[rt[x]] < y) {
        printf("-1\n");
        continue;
      }
      printf("%d\n", kth(rt[x], 1, n, y));
    }
  }
  return 0;
}
```

8.49 树分块

给定一个 n 个节点的树，每个节点上有一个整数，i 号点的整数为 val_i。
有 m 次询问，每次给出 u', v，您需要将其解密得到 u, v，并查询 u 到 v 的路径上有多少个不同的整数。
解密方式：$u = u' \operatorname{xor} lastans$。
$lastans$ 为上一次询问的答案，若无询问则为 0。
第一行有两个整数 n 和 m。
第二行有 n 个整数。第 i 个整数表示 val_i。
在接下来的 $n-1$ 行中，每行包含两个整数 u, v，描述一条边。
在接下来的 m 行中，每行包含两个整数 u', v，描述一组询问。
对于每个询问，一行一个整数表示答案。
对于 100% 的数据，$1 \le u, v \le n \le 4 \times 10^4$，$1 \le m \le 10^5$，$0 \le u', val_i < 2^{31}$。

```
const int N = 40002;
bitset<N> bt[42][42], nw;
vector<int> vec;
int n, m, B, a[N], fa[N], dep[N], mxd[N], FF[N], sz[N],
  ↪ tp[N], son[N];
int id[N], cnt, head[N], tot, ans, sta[N], top, gg[N];
struct edge {
  int to, nxt;
} e[N << 1];
void dfs(int now) {
  sz[now] = 1, son[now] = 0;
  mxd[now] = dep[now];
  for (int i = head[now]; i; i = e[i].nxt)
    if (!dep[e[i].to]) {
      dep[e[i].to] = dep[now] + 1, fa[e[i].to] = now;
      dfs(e[i].to), sz[now] += sz[e[i].to];
      if (mxd[e[i].to] > mxd[now]) mxd[now] =
        ↪ mxd[e[i].to];
      if (!son[now] || sz[son[now]] < sz[e[i].to])
        ↪ son[now] = e[i].to;
    }
  if (mxd[now] - dep[now] >= 1000) id[now] = ++tot,
    ↪ mxd[now] = dep[now];
}
void dfs2(int now) {
  for (int i = head[now]; i; i = e[i].nxt)
    if (dep[e[i].to] > dep[now]) {
      if (id[e[i].to]) {
        int ip = id[sta[top]], in = id[e[i].to];
        for (int x = e[i].to; x != sta[top]; x = fa[x])
          ↪ bt[ip][in].set(a[x]);
        nw = bt[ip][in];
        for (int i = 1; i < top; ++i) {
          bitset<N>& bs = bt[id[sta[i]]][in];
          bs = bt[id[sta[i]]][ip];
          bs |= nw;
        }
        FF[e[i].to] = sta[top], gg[e[i].to] =
          ↪ gg[sta[top]] + 1;
        sta[++top] = e[i].to;
      }
      dfs2(e[i].to);
      if (id[e[i].to]) --top;
    }
}
void dfs3(int now) {
  if (son[now]) tp[son[now]] = tp[now], dfs3(son[now]);
  for (int i = head[now]; i; i = e[i].nxt)
    if (e[i].to != son[now] && dep[e[i].to] > dep[now])
      dfs3(tp[e[i].to] = e[i].to);
}
inline int LCA(int x, int y) {
  while (tp[x] != tp[y])
    if (dep[tp[x]] > dep[tp[y]])
      x = fa[tp[x]];
    else
      y = fa[tp[y]];
  return dep[x] < dep[y] ? x : y;
}
int main() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; ++i) scanf("%d", a + i),
    ↪ vec.push_back(a[i]);
  sort(vec.begin(), vec.end());
  vec.erase(unique(vec.begin(), vec.end()), vec.end());
  for (int i = 1; i <= n; ++i)
    a[i] = lower_bound(vec.begin(), vec.end(), a[i]) -
      ↪ vec.begin();
  for (int i = 1; i < n; ++i) {
    int u, v;
    scanf("%d%d", &u, &v);
    e[++cnt] = (edge){v, head[u]}, head[u] = cnt;
    e[++cnt] = (edge){u, head[v]}, head[v] = cnt;
  }
  dfs(dep[1] = 1);
  if (!id[1]) id[1] = ++tot;
  sta[top = 1] = gg[1] = 1, dfs2(1), dfs3(1);
  while (m--) {
    int u, v;
    scanf("%d%d", &u, &v), u ^= ans, nw.reset();
    int l = LCA(u, v);
    while (u != l && !id[u]) nw.set(a[u]), u = fa[u];
    while (v != l && !id[v]) nw.set(a[v]), v = fa[v];
    if (u != l) {
      int pre = u;
      while (dep[FF[pre]] >= dep[l]) pre = FF[pre];
      if (pre != u) nw |= bt[id[pre]][id[u]];
      while (pre != l) nw.set(a[pre]), pre = fa[pre];
    }
    if (v != l) {
      int pre = v;
      while (dep[FF[pre]] >= dep[l]) pre = FF[pre];
      if (pre != v) nw |= bt[id[pre]][id[v]];
      while (pre != l) nw.set(a[pre]), pre = fa[pre];
    }
    nw.set(a[l]);
    printf("%d\n", ans = nw.count());
  }
  return 0;
}
```

8.50 回滚莫队&不删除莫队

给定一个序列，多次询问一段区间 $[l, r]$，求区间中**相同的数的最远间隔距离**。
序列中两个元素的**间隔距离**指的是**两个元素下标差的绝对值**。
第一行一个整数 n，表示序列长度。
第二行 n 个整数，描述这个序列。
第三行一个整数 m，表示询问个数。
之后 m 行，每行两个整数 l, r 表示询问区间。
共 m 行，每行一个整数表示答案。如果区间内不存在两个数相同，则输出 0。
对于 100% 的数据，满足 $1 \le n, m \le 2 \cdot 10^5$，$1 \le a_i \le 2 \cdot 10^9$。

```
const int N = 2e5 + 5;
int un, n, m, a[N], ANS[N], ma[N], b[N], bn, num[N], st[N],
  ↪ cn, clear[N];
```

```
//变量的解释下面的代码中都有哦!!!
struct que {
  int l, r, i;
  inline bool operator<(const que &nt) const {
    return (b[l] ^ b[nt.l])
           ? b[l] < b[nt.l]
           : r < nt.r;  //先按左边界所在块排，相同时再按右边界排
  }
} q[N];
inline int max(const int &x, const int &y) { return x > y ? x : y; }
int calc(int l, int r) {  //暴力扫一遍
  int last[N], res = 0;
  for (int i = l; i <= r; i++) last[a[i]] = 0;  //记录每个数最早出现的位置
  for (int i = l; i <= r; i++)
    if (!last[a[i]])
      last[a[i]] = i;
    else
      res = max(res, i - last[a[i]]);
  return res;
}
signed main() {
  read(n);
  int len = sqrt(n);  //块长
  for (int i = 1; i <= n; i++)
    num[i] = read(a[i]),
    b[i] = (i - 1) / len + 1;  // b记录每个下标是在哪个块中的
  bn = b[n];                   //块数
  sort(num + 1, num + 1 + n);
  un = unique(num + 1, num + 1 + n) - num - 1;
  for (int i = 1; i <= n; i++)
    a[i] = lower_bound(num + 1, num + 1 + un, a[i]) - num;  //正常的离散操作
  read(m);
  for (int i = 1; i <= m; i++) {
    read(q[i].l);
    read(q[i].r);
    q[i].i = i;
  }
  sort(q + 1, q + 1 + m);                  //询问排序
  for (int i = 1, j = 1; j <= bn; j++) {  // i枚举询问,j枚举询问的左边界所在块
    int br = min(n, j * len), l = br + 1, r = l - 1,
        ans = 0;                    // br是当前块的右边界
    cn = 0;                         //清空记录数组的指针
    for (; b[q[i].l] == j; i++) {  //枚举当前块内的询问
      if (b[q[i].r] == j) {  //如果左右边界都在同一块内内就暴力做
        ANS[q[i].i] = calc(q[i].l, q[i].r);
        continue;
      }
      while (r < q[i].r) {
        r++;
        ma[a[r]] = r;  //最后出现的位置
        if (!st[a[r]])
          st[a[r]] = r,
          clear[++cn] = a
          [r];  // st是最早出现的位置，clear是出现过的数字，用来清空数字最后出现的位置
        ans = max(ans, r - st[a[r]]);  //情况2
      }
      int tp = ans;  //先保存一下，因为右区间的贡献不会被刷新，但左区间的会
      while (l > q[i].l) {
        l--;
        if (ma[a[l]])
          ans = max(ans, ma[a[l]] - l);
        else
          ma[a[l]] = l;  //最后出现的位置可能在左区间中
      }
      ANS[q[i].i] = ans;
      while (l <= br) {
        if (ma[a[l]] == l) ma[a[l]] = 0;  //去掉左区间的贡献
        l++;
      }
      ans = tp;  //去掉当前左区间的贡献
    }
    for (int i = 1; i <= cn; i++)
      ma[clear[i]] = st[clear[i]] =
          0;  //根据记录数组清空每个数出现位置的各种信息
  }
  for (int i = 1; i <= m; i++) write(ANS[i]), puts("");
}
```

8.51 静态仙人掌

任意一条边至多只出现在一条简单回路的无向连通图称为仙人掌。
给你一个有 n 个点和 m 条边的仙人掌图，和 q 组询问每次询问两个点 u, v，求两点之间的最短路。
第一行三个正整数 n, m, q，意义如题目描述。接下来 m 行，每行三个正整数 u, v, w，表示 u, v 之间有一条权值为 w 的无向边。然后 q 行，每行两个正整数 u, v，询问 u 到 v 的最短路。
q 行，每行一个正整数，对应一次询问的结果。
$1 \le n, q \le 10000$ $1 \le m \le 20000$ $1 \le w \le 10^5$

```
#define N 40003
struct edge {
  int v, w;
  edge(int v = 0, int w = 0) : v(v), w(w) {}
};
vector<edge> g[N], adj[N];
int dfn[N], low[N], fa[N];
int top[N], son[N], size[N], dep[N], b[N], sum[N], dis[N];
int n, m, q, cnt, ext;
signed main() {
  int u, v, p, w, A, B, ans;
  read(n), read(m), read(q);
  ext = n;  // ext 为 extra 的简写，表示额外的节点
  for (int i = 1; i <= m; ++i) {
    read(u), read(v), read(w);
    g[u].push_back(edge(v, w));
    g[v].push_back(edge(u, w));
  }
  tarjan(1, 0);  //找环的同时建树
  dfs1(1, 0);
  dfs2(1, 1);  //树剖的两遍dfs
  while (q--) {
    read(u), read(v);
    p = lca(u, v);
    if (p <= n)
      ans = dis[u] + dis[v] - (dis[p] << 1);  //编号不大于n的节点，即是圆点
    else {
      A = find(u, p), B = find(v, p);  //找到儿子A,B
      ans = dis[u] + dis[v] - dis[A] - dis[B];
      if (sum[A] < sum[B]) swap(A, B);  //防止出现负数，这里要swap一下
      ans += min(sum[A] - sum[B], sum[p] + sum[B] - sum[A]);
    }
    print(ans);
    putchar('\n');
  }
  return 0;
}
inline int find(int u, int f) {
  int res;
  while (top[u] != top[f]) {
    res = top[u];
    u = fa[top[u]];
  }
  return u == f ? res : son[f];
}
inline int lca(int u, int v) {
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) swap(u, v);
    u = fa[top[u]];
  }
  return dep[u] < dep[v] ? u : v;
}
void dfs1(int u, int f) {
  fa[u] = f;
  dep[u] = dep[f] + 1;
  size[u] = 1;
  int v, t = -1, l = adj[u].size();
  for (int i = 0; i < l; ++i) {
    v = adj[u][i].v;
    if (v == f) continue;
    dis[v] = dis[u] + adj[u][i].w;
    dfs1(v, u);
```

```
    size[u] += size[v];
    if (size[v] > t) {
      t = size[v];
      son[u] = v;
    }
  }
}
void dfs2(int u, int f) {
  top[u] = f;
  if (son[u] == 0) return;
  dfs2(son[u], f);
  int v, l = adj[u].size();
  for (int i = 0; i < l; ++i) {
    v = adj[u][i].v;
    if (v == fa[u] || v == son[u]) continue;
    dfs2(v, v);
  }
}
void tarjan(int u, int f) {
  dfn[u] = low[u] = ++cnt;
  int v, w, l = g[u].size();
  for (int i = 0; i < l; ++i) {
    v = g[u][i].v;
    if (v == f) continue;  //求点双时不能走到父亲
    w = g[u][i].w;
    if (!dfn[v]) {
      fa[v] = u, b[v] = w;  //把u->v的边权存到v上
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
    } else
      low[u] = min(low[u], dfn[v]);
    if (low[v] <= dfn[u]) continue;
    //圆点之间的连边，保留原图中数据
    adj[u].push_back(edge(v, w));
    adj[v].push_back(edge(u, w));
  }
  for (int i = 0; i < l; ++i) {
    v = g[u][i].v;
    if (fa[v] == u || dfn[v] <= dfn[u]) continue;
    //找到非树边，然后建方点并连边
    solve(u, v, g[u][i].w);
  }
}
inline void solve(int u, int v, int w) {
  //参数w为非树边的边权
  ++ext;
  int pw, pre = w, i = v;
  while (i != fa[u]) {
    sum[i] = pre;
    pre += b[i];
    i = fa[i];
  }
  sum[ext] = sum[u];  //把整个环的边权和存到方点上
  sum[u] = 0;
  i = v;
  while (i != fa[u]) {
    pw = min(sum[i], sum[ext] - sum[i]);
    //找最短路，建树边
    adj[ext].push_back(edge(i, pw));
    adj[i].push_back(edge(ext, pw));
    i = fa[i];
  }
}
```

8.52 后缀自动机（SAM）

给定一个只包含小写字母的字符串 S。
请你求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。
一行一个仅包含小写字母的字符串 S。
输出一个整数，为所求答案。
对于 100% 的数据，$1 \le |S| \le 10^6$。

```
const int N = 3 * 1e6 + 10;
char mde[N];
int nl;
ll res;
struct suffixautomation {
  int mp[N][30], fa[N], ed, ct, len[N], siz[N];
  suffixautomation() { ed = ct = 1; }
  int v[N], x[N], al[N], cnt;
  inline void add(int u, int V) {
    v[++cnt] = V;
    x[cnt] = al[u];
    al[u] = cnt;
  }
  inline void ins(int c) {
    int p = ed;
    siz[ed = ++ct] = 1;
    len[ed] = nl;  //先初始化size和len
    for (; p && mp[p][c] == 0; p = fa[p]) {
      mp[p][c] = ed;
    }  //然后顺着parent树的路径向上找
    if (p == 0) {
      fa[ed] = 1;
      return;
    }
    int q = mp[p][c];  // case1
    if (len[p] + 1 == len[q]) {
      fa[ed] = q;
      return;
    }                          // case2
    len[++ct] = len[p] + 1;  // case 3
    for (int i = 1; i <= 26; i++) {
      mp[ct][i] = mp[q][i];
    }
    fa[ct] = fa[q];
    fa[q] = ct;
    fa[ed] = ct;
    for (int i = p; mp[i][c] == q; i = fa[i]) {
      mp[i][c] = ct;
    }
  }
  inline void bt() {
    for (int i = 2; i <= ct; i++) {
      add(fa[i], i);
    }
  }                //暴力建树
  void dfs(int u)  // dfs
  {
    for (int i = al[u]; i; i = x[i]) {
      dfs(v[i]);
      siz[u] += siz[v[i]];
    }
    if (siz[u] != 1) {
      res = max(res, (ll)siz[u] * len[u]);
    }
  }
} sam;
int main() {
  scanf("%s", mde + 1);  //没啥好说的，建sam然后dfs
  for (nl = 1; mde[nl] != '\0'; nl++) {
    sam.ins(mde[nl] - 'a' + 1);
  }
  sam.bt();
  sam.dfs(1);
  printf("%lld", res);
  return 0;  //拜拜程序~
}
```

```
struct SAM {
  static constexpr int ALPHABET_SIZE = 26;
  struct Node {
    int len;
    int link;
    std::array<int, ALPHABET_SIZE> next;
    Node() : len{}, link{}, next{} {}
  };
  std::vector<Node> t;
  SAM() { init(); }
  void init() {
    t.assign(2, Node());
    t[0].next.fill(1);
    t[0].len = -1;
  }
  int newNode() {
    t.emplace_back();
    return t.size() - 1;
  }
  int extend(int p, int c) {
    if (t[p].next[c]) {
      int q = t[p].next[c];
      if (t[q].len == t[p].len + 1) {
        return q;
```

```
  |   |   | }
  |   |   | int r = newNode();
  |   |   | t[r].len = t[p].len + 1;
  |   |   | t[r].link = t[q].link;
  |   |   | t[r].next = t[q].next;
  |   |   | t[q].link = r;
  |   |   | while (t[p].next[c] == q) {
  |   |   |   | t[p].next[c] = r;
  |   |   |   | p = t[p].link;
  |   |   | }
  |   |   | return r;
  |   | }
  |   | int cur = newNode();
  |   | t[cur].len = t[p].len + 1;
  |   | while (!t[p].next[c]) {
  |   |   | t[p].next[c] = cur;
  |   |   | p = t[p].link;
  |   | }
  |   | t[cur].link = extend(p, c);
  |   | return cur;
  | }
  | int extend(int p, char c, char offset = 'a') { return
      ↪ extend(p, c - offset); }
  | int next(int p, int x) { return t[p].next[x]; }
  | int next(int p, char c, char offset = 'a') { return
      ↪ next(p, c - 'a'); }
  | int link(int p) { return t[p].link; }
  | int len(int p) { return t[p].len; }
  | int size() { return t.size(); }
};
```

9. Hint

9.1 Formulas 公式表

9.1.1 Mobius Inversion

$$F(n)=\sum_{d|n}f(d)\Rightarrow f(n)=\sum_{d|n}\mu(d)F\left(\frac{n}{d}\right)$$

$$F(n)=\sum_{n|d}f(d)\Rightarrow f(n)=\sum_{n|d}\mu\left(\frac{d}{n}\right)F(d)$$

$$[x=1]=\sum_{d|x}\mu(d),\ \ x=\sum_{d|x}\mu(d)$$

9.1.2 降幂公式

$$a^k\equiv a^{k \bmod \varphi(p)+\varphi(p)},\ k\geq\varphi(p)$$

9.1.3 其他常用公式

$$\sum_{i=1}^{n}[(i,n)=1]\,i=n\frac{\varphi(n)+e(n)}{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}[(i,j)=d]=S_\varphi\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[(i,j)=d]=\sum_{d|k}\mu\left(\frac{k}{d}\right)\left\lfloor\frac{n}{k}\right\rfloor\left\lfloor\frac{m}{k}\right\rfloor$$

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

9.1.4 单位根反演

$$\sum_{i=0}^{\lfloor\frac{n}{k}\rfloor}C_n^{ik}$$

引理

$$\frac{1}{k}\sum_{i=0}^{k-1}\omega_k^{in}=[k\mid n]$$

反演

$$Ans=\sum_{i=0}^{n}C_n^i[k\mid i]$$

$$=\sum_{i=0}^{n}C_n^i\left(\frac{1}{k}\sum_{j=0}^{k-1}\omega_k^{ij}\right)$$

$$=\frac{1}{k}\sum_{i=0}^{n}C_n^i\sum_{j=0}^{k-1}\omega_k^{ij}$$

$$=\frac{1}{k}\sum_{j=0}^{k-1}\left(\sum_{i=0}^{n}C_n^i(\omega_k^j)^i\right)$$

$$=\frac{1}{k}\sum_{j=0}^{k-1}(1+\omega_k^j)^n$$

另, 如果要求的是 $[n\%k=t]$, 其实就是 $[k\mid(n-t)]$. 同理推式子即可.

9.1.5 Arithmetic Function

$$(p-1)!\equiv -1(mod\ p)$$

$$a>1,m,n>0,\text{then}\gcd(a^m-1,a^n-1)=a^{\gcd(n,m)}-1$$

$$\mu^2(n)=\sum_{d^2|n}\mu(d)$$

$$a>b,\gcd(a,b)=1,\text{then}\gcd(a^m-b^m,a^n-b^n)=a^{\gcd(m,n)}-b^{\gcd(m,n)}$$

$$\prod_{k=1,gcd(k,m)=1}^{m}k\equiv\begin{cases}-1 & \bmod\ m,\ m=4,p^q,2p^q\\ 1 & \bmod\ m,\ \text{otherwise}\end{cases}$$

$$\sigma_k(n)=\sum_{d|n}d^k=\prod_{i=1}^{\omega(n)}\frac{p_i^{(a_i+1)k}-1}{p_i^k-1}$$

$$J_k(n)=n^k\prod_{p|n}(1-\frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n}J_k(\delta)=n^k$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1]ij=\sum_{i=1}^{n}i^2\varphi(i)$$

$$\sum_{\delta|n}\delta^sJ_r(\delta)J_s(\frac{n}{\delta})=J_{r+s}(n)$$

$$\sum_{\delta|n}\varphi(\delta)d\left(\frac{n}{\delta}\right)=\sigma(n),\ \sum_{\delta|n}|\mu(\delta)|=2^{\omega(n)}$$

$$\sum_{\delta|n}2^{\omega(\delta)}=d(n^2),\ \sum_{\delta|n}d(\delta^2)=d^2(n)$$

$$\sum_{\delta|n}d\left(\frac{n}{\delta}\right)2^{\omega(\delta)}=d^2(n),\ \sum_{\delta|n}\frac{\mu(\delta)}{\delta}=\frac{\varphi(n)}{n}$$

$$\sum_{\delta|n}\frac{\mu(\delta)}{\varphi(\delta)}=d(n),\ \sum_{\delta|n}\frac{\mu^2(\delta)}{\varphi(\delta)}=\frac{n}{\varphi(n)}$$

$$n|\varphi(a^n-1)$$

$$\sum_{\substack{1\leq k\leq n\\ \gcd(k,n)=1}}f(\gcd(k-1,n))=\varphi(n)\sum_{d|n}\frac{(\mu*f)(d)}{\varphi(d)}$$

$$\varphi(\text{lcm}(m,n))\varphi(\gcd(m,n))=\varphi(m)\varphi(n)$$

$$\sum_{\delta|n}d^3(\delta)=(\sum_{\delta|n}d(\delta))^2$$

$$d(uv)=\sum_{\delta|\gcd(u,v)}\mu(\delta)d(\frac{u}{\delta})d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v)=\sum_{\delta|\gcd(u,v)}\delta^k\sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n)=\sum_{k=1}^{n}[\gcd(k,n)=1]\cos 2\pi\frac{k}{n}$$

$$\varphi(n)=\sum_{k=1}^{n}[\gcd(k,n)=1]=\sum_{k=1}^{n}\gcd(k,n)\cos 2\pi\frac{k}{n}$$

$$\begin{cases}S(n)=\sum_{k=1}^{n}(f*g)(k)\\ \sum_{k=1}^{n}S(\lfloor\frac{n}{k}\rfloor)=\sum_{i=1}^{n}f(i)\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}(g*1)(j)\end{cases}$$

$$\begin{cases}S(n)=\sum_{k=1}^{n}(f\cdot g)(k),g\text{ completely multiplicative}\\ \sum_{k=1}^{n}S\left(\left\lfloor\frac{n}{k}\right\rfloor\right)g(k)=\sum_{k=1}^{n}(f*1)(k)g(k)\end{cases}$$

9.1.6 Binomial Coefficients

C	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

$$\binom{n}{k}\equiv[n\&k=k]\pmod 2$$

$$\sum_{k=0}^{n}\binom{k}{m}=\binom{n+1}{m+1}$$

$$\sqrt{1+z}=1+\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k\times 2^{2k-1}}\binom{2k-2}{k-1}z^k$$

$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}$$

$$C_{n,m}=\binom{n+m}{m}-\binom{n+m}{m-1},n\geq m$$

$$\binom{n}{k}=(-1)^k\binom{k-n-1}{k},\ \sum_{k\leq n}\binom{r+k}{k}=\binom{r+n+1}{n}$$

$$\binom{n_1+\cdots+n_p}{m}=\sum_{k_1+\cdots+k_p=m}\binom{n_1}{k_1}\cdots\binom{n_p}{k_p}$$

9.1.7 Fibonacci Numbers, Lucas Numbers

$$F(z)=\frac{z}{1-z-z^2}$$

$$\hat{\phi}=\frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n}f_k=f_{n+2}-1,\ \sum_{k=1}^{n}f_k^2=f_nf_{n+1}$$

$$\sum_{k=0}^{n}f_kf_{n-k}=\frac{1}{5}(n-1)f_n+\frac{2}{5}nf_{n-1}$$

$$\frac{f_{2n}}{f_n}=f_{n-1}+f_{n+1}$$

$$f_1+2f_2+3f_3+\cdots+nf_n=nf_{n+2}-f_{n+3}+2$$

$$\gcd(f_m,f_n)=f_{\gcd(m,n)}$$

$$f_n^2+(-1)^n=f_{n+1}f_{n-1}$$
$$f_{n+k}=f_nf_{k+1}+f_{n-1}f_k$$
$$f_{2n+1}=f_n^2+f_{n+1}^2$$
$$(-1)^kf_{n-k}=f_nf_{k-1}-f_{n-1}f_k$$
$$\text{Modulo } f_n, f_{mn+r}\equiv\begin{cases}f_r, & m \bmod 4=0;\\ (-1)^{r+1}f_{n-r}, & m \bmod 4=1;\\ (-1)^nf_r, & m \bmod 4=2;\\ (-1)^{r+1+n}f_{n-r}, & m \bmod 4=3.\end{cases}$$

```python
def fib(n): # 返回 F(n) 和 F(n + 1)
    if not n: return (0, 1)
    a, b = fib(n >> 1)
    c, d = a * (2 * b - a), a * a + b * b
    return (d, c + d) if n & 1 else (c, d)
```

$$L_0=2,L_1=1,L_n=L_{n-1}+L_{n-2}=(\frac{1+\sqrt5}{2})^n+(\frac{1-\sqrt5}{2})^n$$
$$L(x)=\frac{2-x}{1-x-x^2}$$

除了 $n=0,4,8,16,L_n$ 是素数, 则 n 是素数.

$$\phi=\frac{1+\sqrt5}{2},\ \hat\phi=\frac{1-\sqrt5}{2}$$
$$F_n=\frac{\phi^n-\hat\phi^n}{\sqrt5},\ L_n=\phi^n+\hat\phi^n$$
$$\frac{L_n+F_n\sqrt5}{2}=\left(\frac{1+\sqrt5}{2}\right)^n$$

9.1.8 Sum of Powers

$$\sum_{i=1}^n i^2=\frac{n(n+1)(2n+1)}{6},\ \sum_{i=1}^n i^3=\left(\frac{n(n+1)}{2}\right)^2$$
$$\sum_{i=1}^n i^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$
$$\sum_{i=1}^n i^5=\frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

9.1.9 Catalan Numbers 1, 1, 2, 5, 14, 42, 132, 429, 1430...

$$c_0=1,c_n=\sum_{i=0}^{n-1}c_ic_{n-1-i}=c_{n-1}\frac{4n-2}{n+1}=\frac{\binom{2n}{n}}{n+1}=\binom{2n}{n}-\binom{2n}{n-1}$$
$$c(x)=\frac{1-\sqrt{1-4x}}{2x}$$

Usage: n 对括号序列; n 个点满二叉树; $n\times n$ 的方格左下到右上不过对角线方案数; 凸 $n+2$ 边形三角形分割数; n 个数的出栈方案数; $2n$ 个顶点连接, 线段两两不交的方案数.

类卡特兰数 从 $(1,1)$ 出发走到 (n,m), 只能向右或者向上走, 不能越过 $y=x$ 这条线 (即保证 $x\geq y$), 合法方案数是 $C_{n+m-2}^n-C_{n+m-2}^{n-1}$.

9.1.10 Motzkin Numbers 1, 1, 2, 4, 9, 21, 51, 127, 323, 835...

圆上 n 点间画不相交弦的方案数. 选 n 个数 $k_1,k_2,...,k_n\in\{-1,0,1\}$, 保证 $\sum_i^a k_i(1\leq a\leq n)$ 非负且所有数总和为 0 的方案数.

$$M_{n+1}=M_n+\sum_i^{n-1}M_iM_{n-1-i}=\frac{(2n+3)M_n+3nM_{n-1}}{n+3}$$
$$M_n=\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\binom{n}{2k}\text{Catlan}(k)$$
$$M(X)=\frac{1-x-\sqrt{1-2x-3x^2}}{2x^2}$$

9.1.11 Derangement 错排数 0, 1, 2, 9, 44, 265, 1854, 14833...

$$D_1=0,D_2=1,D_n=n!(\frac{1}{0!}-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+...+\frac{(-1)^n}{n!})$$
$$D_n=(n-1)(D_{n-1}+D_{n-2})$$

9.1.12 Bell Numbers 1, 1, 2, 5, 15, 52, 203, 877, 4140 ...

n 个元素集合划分的方案数.

$$B_n=\sum_{k=1}^n\begin{Bmatrix}n\\k\end{Bmatrix},\ B_{n+1}=\sum_{k=0}^n\binom{n}{k}B_k$$
$$B_{p^m+n}\equiv mB_n+B_{n+1}\pmod p$$
$$B(x)=\sum_{n=0}^\infty\frac{B_n}{n!}x^n=e^{e^x-1}$$

9.1.13 Stirling Numbers

第一类 n 个元素集合分作 k 个非空轮换方案数.

$$\begin{bmatrix}n+1\\k\end{bmatrix}=n\begin{bmatrix}n\\k\end{bmatrix}+\begin{bmatrix}n\\k-1\end{bmatrix}$$
$$s(n,k)=(-1)^{n-k}\begin{bmatrix}n\\k\end{bmatrix}$$

n\k	0	1	2	3	4	5	6
0	1						
1	0	1					
2	0	1	1				
3	0	2	3	1			
4	0	6	11	6	1		
5	0	24	50	35	10	1	
6	0	120	274	225	85	15	1
7	0	720	1764	1624	735	175	21

$$\begin{bmatrix}n+1\\2\end{bmatrix}=n!H_n\text{ (see 9.1.15)}$$
$$x^{\underline n}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}(-1)^{n-k}x^k$$
$$x^{\overline n}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}x^k$$

For fixed k, EGF:

$$\sum_{n=0}^\infty\begin{bmatrix}n\\k\end{bmatrix}\frac{x^n}{n!}=\frac{x^k}{k!}\left(\frac{\ln(1-x)}{x}\right)^k$$

第二类 把 n 个元素集合分作 k 个非空子集方案数.

$$\begin{Bmatrix}n+1\\k\end{Bmatrix}=k\begin{Bmatrix}n\\k\end{Bmatrix}+\begin{Bmatrix}n\\k-1\end{Bmatrix}$$
$$m!\begin{Bmatrix}n\\m\end{Bmatrix}=\sum_k\binom{m}{k}k^n(-1)^{m-k}$$

n\k	0	1	2	3	4	5	6
0	1						
1	0	1					
2	0	1	1				
3	0	1	3	1			
4	0	1	7	6	1		
5	0	1	15	25	10	1	
6	0	1	31	90	65	15	1
7	0	1	63	301	350	140	21

$$x^n=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}x^{\underline k}$$
$$=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k}x^{\overline k}$$

For fixed k, EGF and OGF:

$$\sum_{n=0}^\infty\begin{Bmatrix}n\\k\end{Bmatrix}\frac{x^n}{n!}=\frac{x^k}{k!}\left(\frac{e^x-1}{x}\right)^k$$
$$\sum_{n=0}^\infty\begin{Bmatrix}n\\k\end{Bmatrix}x^n=x^k\prod_{i=1}^k(1-ix)^{-1}$$

9.1.14 Eulerian Numbers

n\k	0	1	2	3	4	5	6
1	1						
2	1	1					
3	1	4	1				
4	1	11	11	1			
5	1	26	66	26	1		
6	1	57	302	302	57	1	
7	1	120	1191	2416	1191	120	1

$$\left\langle\begin{matrix}n\\k\end{matrix}\right\rangle=(k+1)\left\langle\begin{matrix}n-1\\k\end{matrix}\right\rangle+(n-k)\left\langle\begin{matrix}n-1\\k-1\end{matrix}\right\rangle$$
$$x^n=\sum_k\left\langle\begin{matrix}n\\k\end{matrix}\right\rangle\binom{x+k}{n}$$
$$\left\langle\begin{matrix}n\\m\end{matrix}\right\rangle=\sum_{k=0}^m\binom{n+1}{k}(m+1-k)^n(-1)^k$$

9.1.15 Harmonic Numbers, 1, 3/2, 11/6, 25/12, 137/60 ...

$$H_n=\sum_{k=1}^n\frac{1}{k},\sum_{k=1}^nH_k=(n+1)H_n-n$$
$$\sum_{k=1}^nkH_k=\frac{n(n+1)}{2}H_n-\frac{n(n-1)}{4}$$
$$\sum_{k=1}^n\binom{k}{m}H_k=\binom{n+1}{m+1}(H_{n+1}-\frac{1}{m+1})$$

9.1.16 卡迈克尔函数

卡迈克尔函数表示模 m 剩余系下最大的阶, 即 $\lambda(m)=\max_{a\perp m}\delta_m(a)$. 容易看出, 若 $\lambda(m)=\varphi(m)$, 则 m 存在原根. 该函数可由下述方法计算: 分解质因数 $m=p_1^{\alpha_1}p_2^{\alpha_2}...p_t^{\alpha_t}$. 则 $\lambda(m)=\text{lcm}(\lambda(p_1^{\alpha_1}),p_2^{\alpha_2},...,p_t^{\alpha_t})$. 其中对奇质数 p, $\lambda(p^\alpha)=(p-1)p^{\alpha-1}$. 对2的幂, $\lambda(2^k)=2^{k-2},s.t.k\geq3$. $\lambda(4)=\lambda(2)=2$.

9.1.17 五边形数定理求拆分数

$$\Phi(x)=\prod_{n=1}^\infty(1-x^n)=\sum_{n=-\infty}^\infty(-1)^kx^{k(3k-1)/2}$$

记 $p(n)$ 表示 n 的拆分数,$f(n,k)$ 表示将 n 拆分且每种数字使用次数必须小于 k 的拆分数. 则

$$P(x)\Phi(x)=1,F(x^k)\Phi(x)=1$$

暴力拆开卷积, 可以得到将1,-1,2,-2... 带入五边形数 $(-1)^kx^{k(3k-1)/2}$ 中, 由于小于n的五边形数只有 $\sqrt n$ 个, 可以 $O(n\sqrt n)$ 计算答案:

$$p(n)=p(n-1)+p(n-2)-p(n-5)-p(n-7)+\cdots$$
$$f(n,k)=p(n)-p(n-k)-p(n-2k)+p(n-5k)+p(n-7k)-\cdots$$

9.1.18 Bernoulli Numbers 1, 1/2, 1/6, 0, -1/30, 0, 1/42 ...

$$B(x)=\sum_{i\geq 0}\frac{B_i x^i}{i!}=\frac{x}{e^x-1}$$

$$B_n=[n=0]-\sum_{i=0}^{n-1}\binom{n}{i}\frac{B_i}{n-k+1},\sum_{i=0}^{n}\binom{n+1}{i}B_i=0$$

$$S_n(m)=\sum_{i=0}^{m-1}i^n=\sum_{i=0}^{n}\binom{n}{i}B_{n-i}\frac{m^{i+1}}{i+1}$$

$B_0=1,\ B_1=-\frac{1}{2},\ B_4=-\frac{1}{30},\ B_6=\frac{1}{42},\ B_8=-\frac{1}{30},\ \ldots.$

(除了 $B_1=-\frac{1}{2}$ 以外, 伯努利数的奇数项都是 0.)

自然数幂次和关于次数的EGF:

$$F(x)=\sum_{k=0}^{\infty}\frac{\sum_{i=0}^{n}i^k}{k!}x^k$$

$$=\sum_{i=0}^{n}e^{ix}=\qquad\frac{e^{(n+1)x-1}}{e^x-1}$$

9.1.19 kMAX-MIN反演

$$k\mathrm{MAX}(S)=\sum_{T\subset S,T\neq\emptyset}(-1)^{|T|-k}C_{|T|-1}^{k-1}\mathrm{MIN}(T)$$

代入 $k=1$ 即为MAX-MIN反演:

$$\mathrm{MAX}(S)=\sum_{T\subset S,T\neq\emptyset}(-1)^{|T|-1}\mathrm{MIN}(T)$$

9.1.20 伍德伯里矩阵不等式

$$(A+UCV)^{-1}=A^{-1}-A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}$$

该等式可以动态维护矩阵的逆, 令 $C=[1],U,V$ 分别为 $1\times n$ 和 $n\times 1$ 的向量, 这样可以构造出 UCV 为只有某行或者某列不为0的矩阵, 一次修改复杂度为 $O(n^2)$.

9.1.21 Sum of Squares

$r_k(n)$ 表示用 k 个平方数组成 n 的方案数. 假设:

$$n=2^{a_0}p_1^{2a_1}\cdots p_r^{2a_r}q_1^{b_1}\cdots q_s^{b_s}$$

其中 $p_i\equiv 3 \mod 4, q_i\equiv 1 \mod 4$, 那么

$$r_2(n)=\begin{cases}0 & \text{if any } a_i \text{ is a half-integer}\\ 4\prod_1^r(b_i+1) & \text{if all } a_i \text{ are integers}\end{cases}$$

$r_3(n)>0$ 当且仅当 n 不满足 $4^a(8b+7)$ 的形式 $(a,b$ 为整数).

9.1.22 枚举勾股数 Pythagorean Triple

枚举 $x^2+y^2=z^2$ 的三元组: 可令 $x=m^2-n^2,y=2mn,z=m^2+n^2$, 枚举 m 和 n 即可 $O(n)$ 枚举勾股数. 判断素勾股数方法: m,n 至少一个为偶数并且 m,n 互质, 那么 x,y,z 就是素勾股数.

9.1.23 四面体体积 Tetrahedron Volume

If U, V, W, u, v, w are lengths of edges of the tetrahedron (first three form a triangle; u opposite to U and so on)

$$V=\frac{\sqrt{4u^2v^2w^2-\sum_{cyc}u^2(v^2+w^2-U^2)^2+\prod_{cyc}(v^2+w^2-U^2)}}{12}$$

9.1.24 杨氏矩阵与钩子公式

满足: 格子 (i,j) 没有元素, 则它右边和上边相邻格子也没有元素; 格子 (i,j) 有元素 $a[i][j]$, 则它右边和上边相邻格子要么没有元素, 要么有元素且比 $a[i][j]$ 大.

计数: $F_1=1,F_2=2,F_n=F_{n-1}+(n-1)F_{n-2},F(x)=e^{x+\frac{x^2}{2}}$

钩子公式: 对于给定形状 λ, 不同杨氏矩阵的个数为:

$$d_\lambda=\frac{n!}{\prod h_\lambda(i,j)}$$

$h_\lambda(i,j)$ 表示该格子右边和上边的格子数量加1.

9.1.25 常见博弈游戏

Nim-K游戏 n 堆石子轮流拿, 每次最多可以拿k堆石子, 谁走最后一步输。结论: 把每一堆石子的sg值 (即石子数量) 二进制分解, 先手必败当且仅当每一位二进制位上1的个数是 (k+1) 的倍数.

Anti-Nim游戏 n 堆石子轮流拿, 谁走最后一步输。结论: 先手胜当且仅当1. 所有堆石子数都为1且游戏的SG值为0 (即有偶数个孤单堆-每堆只有1个石子数) 2. 存在某堆石子数大于1且游戏的SG值不为0.

斐波那契博弈 有一堆物品, 两人轮流取物品, 先手最少取一个, 至多无上限, 但不能把物品取完, 之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件, 取走最后一件物品的人获胜. 结论: 先手胜当且仅当物品数 n 不是斐波那契数.

威佐夫博弈 有两堆石子, 博弈双方每次可以取一堆石子中的任意个, 不能不取, 或者取两堆石子中的相同个. 先取完者赢. 结论: 求出两堆石子 A 和 B 的差值 C, 如果 $\left\lfloor C*\frac{\sqrt{5}+1}{2}\right\rfloor=min(A,B)$ 那么后手赢, 否则先手赢.

约瑟夫环 n 个人 $0,\ldots,n-1$, 令 $f_{i,m}$ 为 i 个人报 m 的胜利者, 则 $f_{1,m}=0, f_{i,m}=(f_{i-1,m}+m)\bmod i$.

阶梯Nim 在一个阶梯上, 每次选一个台阶上任意个式子移到下一个台阶上, 不可移动者输. 结论: SG值等于奇数层台阶上石子数的异或和. 对于树形结构也适用, 奇数层节点上所有石子数异或起来即可.

图上博弈 给定无向图, 先手从某点开始走, 只能走相邻且未走过的点, 无法移动者输. 对该图求最大匹配, 若某个点不一定在最大匹配中则先手必败, 否则先手必胜.

最大最小定理求纳什均衡点 在二人零和博弈中, 可以用以下方式求出一个纳什均衡点: 在博弈双方中任选一方, 求混合策略 $\mathbf{p}$ 使得对方选择任意一个纯策略时, 己方的最小收益最大 (等价于对方的最大收益最小). 据此可以求出双方在此局面下的最优期望得分, 分别等于己方最大的最小收益和对方最小的最大收益. 一般而言, 可以得到形如

$$\max_{\mathbf{p}}\min_{i}\sum_{p_j\in\mathbf{p}}p_jw_{i,j},\text{s.t. }p_j\geq 0,\sum p_j=1$$

的形式. 当 $\sum p_jw_{i,j}$ 可以表示成只与 i 有关的函数 $f(i)$ 时, 可以令初始时 $p_i=0$, 不断调整 $\sum p_jw_{i,j}$ 最小的那个i的概率 p_i, 直至无法调整或者 $\sum p_j=1$ 为止.

9.1.26 概率相关

$D(X)=E(X-E(X))^2=E(X^2)-(E(X))^2, D(X+Y)=D(X)+D(Y)D(aX)=a^2D(X)$

$$E[x]=\sum_{i=1}^{\infty}P(X\geq i)$$

m 个数的方差:

$$s^2=\frac{\sum_{i=1}^{m}x_i^2}{m}-\overline{x}^2$$

9.1.27 邻接矩阵行列式的意义

在无向图中取若干个环, 一种取法权值就是边权的乘积, 对行列式的贡献是 $(-1)^{\text{even}}$, 其中 even 是偶环的个数.

9.1.28 Others (某些近似数值公式在这里)

$$S_j=\sum_{k=1}^{n}x_k^j$$

$$h_m=\sum_{1\leq j_1<\cdots<j_m\leq n}x_{j_1}\cdots x_{j_m},\ H_m=\sum_{1\leq j_1\leq\cdots\leq j_m\leq n}x_{j_1}\cdots x_{j_m}$$

$$h_n=\frac{1}{n}\sum_{k=1}^{n}(-1)^{k+1}S_kh_{n-k}$$

$$H_n=\frac{1}{n}\sum_{k=1}^{n}S_kH_{n-k}$$

$$\sum_{k=0}^{n}kc^k=\frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}$$

$$\sum_{i=1}^{n}\frac{1}{n}\approx\ln\left(n+\frac{1}{2}\right)+\frac{1}{24(n+0.5)^2}+\Gamma,(\Gamma\approx 0.57721566490153286060 65)$$

$$n!=\sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$\max\{x_a-x_b,y_a-y_b,z_a-z_b\}-\min\{x_a-x_b,y_a-y_b,z_a-z_b\}$$

$$=\frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|$$

$$(a+b)(b+c)(c+a)=\frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

$$a^3+b^3=(a+b)(a^2-ab+b^2),a^3-b^3=(a-b)(a^2+ab+b^2)$$

$n\bmod 2=1$:

$$a^n+b^n=(a+b)(a^{n-1}-a^{n-2}b+a^{n-3}b^2-\ldots-ab^{n-2}+b^{n-1})$$

划分问题: n 个 $k-1$ 维向量最多把 k 维空间分为 $\sum_{i=0}^{k}C_n^i$ 份.

9.2 Calculus Table 导数表

$(\frac{u}{v})' = \frac{u'v-uv'}{v^2}$
$(a^x)' = (\ln a)a^x$
$(\tan x)' = \sec^2 x$
$(\cot x)' = \csc^2 x$
$(\sec x)' = \tan x \sec x$
$(\csc x)' = -\cot x \csc x$
$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$
$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
$(\arctan x)' = \frac{1}{1+x^2}$
$(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$
$(\operatorname{arccsc} x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\operatorname{arcsec} x)' = \frac{1}{x\sqrt{1-x^2}}$
$(\tanh x)' = \operatorname{sech}^2 x$
$(\coth x)' = -\operatorname{csch}^2 x$
$(\operatorname{sech} x)' = -\operatorname{sech} x \tanh x$
$(\operatorname{csch} x)' = -\operatorname{csch} x \coth x$
$(\operatorname{arcsinh} x)' = \frac{1}{\sqrt{1+x^2}}$
$(\operatorname{arccosh} x)' = \frac{1}{\sqrt{x^2-1}}$
$(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$
$(\operatorname{arccoth} x)' = \frac{1}{x^2-1}$
$(\operatorname{arccsch} x)' = -\frac{1}{|x|\sqrt{1+x^2}}$
$(\operatorname{arcsech} x)' = -\frac{1}{x\sqrt{1-x^2}}$

9.3 Integration Table 积分表

$ax^2+bx+c (a>0)$

1. $\int \frac{\mathrm{d}x}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2<4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln\left|\frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right| + C & (b^2>4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int \frac{\mathrm{d}x}{ax^2+bx+c}$

$\sqrt{\pm ax^2+bx+c} (a>0)$

1. $\int \frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
2. $\int \sqrt{ax^2+bx+c}\mathrm{d}x = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
4. $\int \frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}\mathrm{d}x = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$

$\sqrt{\pm\frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$

1. $\int \frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C\,(a<b)$
2. $\int \sqrt{(x-a)(b-x)}\mathrm{d}x = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}} + C, (a<b)$

三角函数的积分

1. $\int \tan x\mathrm{d}x = -\ln|\cos x| + C$
2. $\int \cot x\mathrm{d}x = \ln|\sin x| + C$
3. $\int \sec x\mathrm{d}x = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x+\tan x| + C$
4. $\int \csc x\mathrm{d}x = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x-\cot x| + C$
5. $\int \sec^2 x\mathrm{d}x = \tan x + C$
6. $\int \csc^2 x\mathrm{d}x = -\cot x + C$
7. $\int \sec x\tan x\mathrm{d}x = \sec x + C$
8. $\int \csc x\cot x\mathrm{d}x = -\csc x + C$
9. $\int \sin^2 x\mathrm{d}x = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
10. $\int \cos^2 x\mathrm{d}x = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
11. $\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x\mathrm{d}x$
12. $\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x\mathrm{d}x$
13. $\int \frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\sin^{n-2}x}$
14. $\int \frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2}x}$
15. $$\begin{aligned} &\int \cos^m x\sin^n x\mathrm{d}x \\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x\mathrm{d}x \\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x\mathrm{d}x \end{aligned}$$
16. $\int \frac{\mathrm{d}x}{a+b\sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2) \\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2) \end{cases}$
17. $\int \frac{\mathrm{d}x}{a+b\cos x} = \begin{cases} \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2) \\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2) \end{cases}$
18. $\int \frac{\mathrm{d}x}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
19. $\int \frac{\mathrm{d}x}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$
20. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
21. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
22. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
23. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

反三角函数的积分 (其中 $a>0$)

1. $\int \arcsin\frac{x}{a}\mathrm{d}x = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}\mathrm{d}x = x\,arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}\mathrm{d}x = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}\mathrm{d}x = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

指数函数的积分

1. $\int a^x\mathrm{d}x = \frac{1}{\ln a}a^x + C$
2. $\int \mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}a^{ax} + C$
3. $\int x\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}x^n\mathrm{e}^{ax} - \frac{n}{a}\int x^{n-1}\mathrm{e}^{ax}\mathrm{d}x$
5. $\int xa^x\mathrm{d}x = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^n a^x\mathrm{d}x = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$
7. $\int \mathrm{e}^{ax}\sin bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(a\sin bx - b\cos bx) + C$
8. $\int \mathrm{e}^{ax}\cos bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(b\sin bx + a\cos bx) + C$
9. $\int \mathrm{e}^{ax}\sin^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\sin^{n-1}bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\sin^{n-2}bx\mathrm{d}x$
10. $\int \mathrm{e}^{ax}\cos^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1}bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\cos^{n-2}bx\mathrm{d}x$

对数函数的积分

1. $\int \ln x\mathrm{d}x = x\ln x - x + C$
2. $\int \frac{\mathrm{d}x}{x\ln x} = \ln\left|\ln x\right| + C$
3. $\int x^n\ln x\mathrm{d}x = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n\mathrm{d}x = x(\ln x)^n - n\int (\ln x)^{n-1}\mathrm{d}x$
5. $\int x^m(\ln x)^n\mathrm{d}x = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}\mathrm{d}x$

9.4 Constant Table 常数表

Random primes generated at Sat Dec 9 11:39:09 2023
1e3 953 967 971 983 991 997 1009 1021 1031 1039 1049 1063 1069
3e4 28289 28547 28603 28631 28663 29243 29333 30013 31147 31963
1e5 96823 97463 98251 98909 103399 104659 105367 105373 106303
5e5 470131 475997 492631 493277 511087 524087 529037 531121
1e6 941351 941653 965843 970861 1000303 1005937 1048123 1050977
2e6 1922773 2002349 2059979 2073347 2115527 2126801 2132587
1e7 9876833 10026743 10294937 10306589 10446679 10618177 10652549
2e7 18964039 19474033 19561957 20016377 20139239 20541629
1e9 945734971 984826559 986759827 990874211 1037413691 1050465593

n	$\log_{10} n$	$n!$	$C(n, n/2)$	LCM$(1 \ldots n)$
2	0.30102999	2	2	2
3	0.47712125	6	3	6
4	0.60205999	24	6	12
5	0.69897000	120	10	60
6	0.77815125	720	20	60
7	0.84509804	5040	35	420
8	0.90308998	40320	70	840
9	0.95424251	362880	126	2520
10	1	3628800	252	2520
11	1.04139269	39916800	462	27720
12	1.07918125	479001600	924	27720
15	1.17609126	1.31e12	6435	360360
20	1.30103000	2.43e18	184756	232792560
25	1.39794001	1.55e25	5200300	26771144400
30	1.47712125	2.65e32	155117520	1.444e14

$n \leq$	10	100	1e3	1e4	1e5	1e6
max$\{\omega(n)\}$	2	3	4	5	6	7
max$\{d(n)\}$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \leq$	1e7	1e8	1e9	1e10	1e11	1e12
max$\{\omega(n)\}$	8	8	9	10	10	11
max$\{d(n)\}$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \leq$	1e13	1e14	1e15	1e16	1e17	1e18
max$\{\omega(n)\}$	12	12	13	13	14	15
max$\{d(n)\}$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim x/\log(x)$					

9.5 现场赛宝典

1. 比赛前晚一定要睡眠充足，至少保证8-10个小时睡眠时间。
2. 热身赛每个人都轮流适应一下比赛机器。同时测试一下打印代码的流程，和提问、提交流程。要求每个队员都熟悉流程，保证提交不会出现错误。
3. 赛前的心情非常非常重要！队友之间一定要互相开玩笑，保持愉快的心情开场。实在所有人都没有心情娱乐，那就脸部保持微笑状态三分钟，保证能改变心情。（强颜欢笑也是乐）
4. 开场后分配好题目（一般前中后三部分），英语好要迅速读完题目，并且指定一个人每两三分钟看一次rank，保证FB后迅速换题）
5. 如果读题过程中能确定某题可出，并且为队友的擅长题型，迅速通知队友，并说清题意、题目条件，讨论是否可敲。如果敲题，这时候读完自己题目后，还继续把队友没读的题读完。
6. 看到有人出题后一定要第一时间阅读该题，有条件最好两人同时读，并且保证题目描述、条件和输入输出都没有异议。讨论该题的可行性，如果完全没有思路，并且是神校出题，再观望一段时间，别盲目敲题。
7. 提交后有一段等待时间，这时候不要全部盯着屏幕等，不管对错，直接打印代码，然后再检查一遍输入输出和数据，检查代码有没有问题。空闲的人继续读题或看榜。
8. 如果提交没过，有其他题在开着，换人敲。之前的人拿着打印的代码检查bug。如不能保证bug很快找到，至少两个人同时debug，不需要双开。
9. 如果某题2次WA，并且找不到原因，这时候一边debug一边让别人重新读一遍题目，逐字分析条件和输入输出，不受其他人影响，然后交流题意等等是否出现问题。题意和算法没问题的情况下，想trick出数据。
10. YES了一道题后，主敲队员最好的放松方法就是去趟洗手间！不管想不想方便，去洗个脸，呼吸一下新鲜空气也是很有必要的。清醒一下，把YES的那道题抛到脑后，关注其他题。还有就是在脑子非常乱的时候，非常憋屈的时候，也去趟洗手间，说不定还能偷瞄到大神的思路……
11. 比赛士气和心情很重要，一定要有一个队员心理素质够硬，负责调动全队状态。小口喝水、身体坐正，双手交叉，互相打气能快速帮你调整状态。
12. 比赛期间以观察rank为出题标准，但是如果认定该题不适合你们，要果断先选择另外一道过的人较多的题！
13. 在没有两个人都足够主敲的情况下，切忌双开敲题，尤其到了最后一两个小时，一般三人全力攻题！出题的同时，空闲队员想好trick数据等等。
14. 封榜后一定不能乱，不要盲目交题，不要重复交同样代码!!
15. 记得比赛比的是心态和状态，如果浮躁，自己就先输了，一定要淡定！如果开始自己或队友发现出现浮躁慌乱，一定要提醒大家。
16. 相信自己，相信队友，和谐相处，共同拿牌!!!

9.6 心态炸时, 检查这些

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 'multiset' 还是 'set'
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小（2倍，4倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 清零的时候全部用 0 到 n+1。
- 模意义下不要用除法
- 上取整以及 GCD 小心负数
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化
- 分块一定要特判在同一块中的情况

Python Example

```python
def IO_and_Exceptions():
    try:
        with open("a.in", mode="r") as fin:
            for line in fin:
                a = list(map(int, line.split()))
                print(a, end = "\n")
    except: exit()
    assert False, '17 cards can\'t kill me'
def Random():
    import random as rand
    rand.normalvariate(0.5, 0.1)
    l = [str(i) for i in range(9)]
    sorted(l), min(l), max(l), len(l)
    rand.shuffle(l)
    l.sort(key=lambda x:x ^ 1,reverse=True)
    import functools as ft
    l.sort(key=ft.cmp_to_key(lambda x, y:(y^1)-(x^1)))
def FractionOperation():
    from fractions import Fraction
    a = Fraction(0.233).limit_denominator()
    a == Fraction("0.233") #True
    a.numerator, a.denominator, str(a)
def DecimalOperation():
    import decimal
    from decimal import Decimal, getcontext
    getcontext().prec = 100
    getcontext().rounding = getattr(decimal,
        ↪ 'ROUND_HALF_EVEN')
    # default; other: FLOOR, CELILING, DOWN, ...
    getcontext().traps[decimal.FloatOperation] = True
    Decimal((0, (1, 4, 1, 4), -3)) # 1.414
    a = Decimal(1<<31) / Decimal(100000)
    print(round(a, 5)) # total digits
    print(a.quantize(Decimal("0.00000")))
    # 21474.83648
    print(a.sqrt(), a.ln(), a.log10(), a.exp(), a ** 2)
def Complex():
    a = 1-2j
    print(a.real, a.imag, a.conjugate())
def FastIO():
    import atexit
    import io
    import sys
    _INPUT_LINES = sys.stdin.read().splitlines()
    input = iter(_INPUT_LINES).__next__
    _OUTPUT_BUFFER = io.StringIO()
    sys.stdout = _OUTPUT_BUFFER
    @atexit.register
    def write():
        sys.__stdout__.write(_OUTPUT_BUFFER.getvalue())
```

Good Luck && Have Fun!