Tips

1. pow 返回浮点数可能会导致误差，慎用！！！自己写。
2. 样例数用 long long 可能会 TLE
3. 多组数据记得初始化
4. 小心过程量 overflow
5. Type convertions
6. 在取余的情况下，要避免减法运算的结果出现负数 (...+ MOD) % MOD; 多加点总不会错?
7. fill 注意初始范围为从 1 到 n 的情况
8. 提交换行
9. c++11: long long int abs (long long int n);

Basic Algorithm

Sort

```
// Bubble Sort
for (int i = 0; i < N; i++)
    for (int j = 0; j < N - i - 1; j++)
        if (A[j] > A[j + 1]) swap(A[j], A[j + 1]);
```

```
// Insertion Sort
for (int i = 1; i < N; i++) {
    int tmp = A[i], j;
    for (j = i - 1; j >= 0 && A[j] > tmp; j--)
        A[j + 1] = A[j];
    A[++j] = tmp;
}
```

```
// Selection Sort
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (A[i] > A[j]) swap(A[i], A[j])
```

```
// Merge Sort
void merge_sort(int A[], int l, int r) {
    if (l >= r) return ;
    int mid = (l + r) / 2;
    merge_sort(A, l, mid);
    merge_sort(A, mid + 1, r);
    int i, j, k;
    i = l; j = mid + 1; k = l;
    while (i <= mid && j <= r) {
        if (A[i] <= A[j]) {
            B[k] = A[i++];
        }
        else {
            B[k] = A[j++];
        }
        k++;
    }
    while (i <= mid) {
        B[k] = A[i++];
        k++;
    }
    while (j <= r) {
        B[k] = A[j++];
        k++;
```

```
    }
    memcpy(A, B, r - l + 1);
    return ;
}
```

```
// Quick Sort
void quicksort(int A[], int l, int r) {
    int i = l, j = r, mid = A[(r - l) / 2 + l];
    while (i <= j) {
        while (A[i] < mid) i++;
        while (A[j] > mid) j--;
        if (i <= j) {
            swap(A[i], A[j]);
            ++i; --j;
        }
    }
    if (i < r) quicksort(A, i, r);
    if (l < j) quicksort(A, l, j);
    return ;
}
```

- Heap Sort（见堆的内容）

DP

LIS

```
int A[MAX_N];
long lis(int n) {
    int dp[MAX_N];
    fill(dp, dp + n, INF);
    for (int i = 0; i < n; ++i)
        *lower_bound(dp, dp + n, A[i]) = A[i];// lds: -A[i]; ln: upper_bound
    return lower_bound(dp, dp + n, INF) - dp;
}
```

Knapsack Problem

- 0/1 背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]] + v[i])$$

```
for (int i = 0; i < N; ++i)
    for (int j = W; j >= w[i]; --j)
        f[j] = max(f[j - w[i]] + c[i], f[j]);
```

- 完全背包

$$f[i,j] = max(f[i-1,j], f[i-1,j-w[i]] + j[i])$$

```
for (int i = 0; i < N; ++i)
    for (int j = w[i]; j <= W; ++j)
        f[j] = max(f[j - w[i]] + c[i], f[v]);
```

注意循环顺序的不同背后思路。

- 一个简单的优化：若两件物品 i、j 满足 $w[i] \leq w[j]$ 且 $c[i] \geq c[j]$，则讲物品 j 去掉，不用考虑。
- 转化为 01 背包问题求解：
  - 第 i 种物品转化为 $\frac{V}{w[i]}$ 件费用于价值均不变的物品。
  - 第 i 种物品拆成费用为 $w[i] * 2^k$，价值为 $c[i] * 2^k$ 的若干件物品其中 k 满足 $w[i] * 2^k < V$

- 多重背包

$$f[i,j] = max(f[i-1, j-w[i]*k] + v[i]*k | 0 <= k <= m[i])$$

- 优化：转化为 01 背包问题
  - 将第 i 件物品分成若干件物品，每件物品的系数分别为：$1, 2, 4, \ldots, 2^{(k-1)}, n[i] - 2^k$
  - **根据 w，v 范围改变 DP 对象，可以考虑针对不同价值计算最小的重量。（$f[i][j]$，其中 j 代表价值总和）**

```
for (int i = 0; i < N; ++i) {
    int num = m[i];
    for (int k = 1; num > 0; k <<= 1) {
        int mul = min(k, num);
        for (int j = W; j >= w[i] * mul; --j) {
            f[j] = max(f[j - w[i] * mul] + v[i] * mul, f[j]);
        }
        num -= mul;
    }
}
```

- 混合三种背包

  弄清楚上面三种背包后分情况就好

- 二维费用背包

$$f[i,j,k] = max(f[i-1,j,k], f[i-1,j-a[i],k-b[i]] + c[i])$$

二维费用可由最多取 m 件等方式隐蔽给出。

- 分组背包

$$f[k,j] = max f[k-1,j], f[k-1,j-w[i]] + v[i] | i \in K)$$

```
for (int k = 0; k < K; ++k)
    for (j = W; j >= 0; --j)
        for (int i = 0; i <= m[k]; ++i)
            f[j] = max(f[j - w[i]] + v[i], f[j]);
```

显然可以对每组中物品应用完全背包中“一个简单有效的优化”

- 有依赖背包

由 NOIP2006 金明的预算方案引申，对每个附件先做一个 01 背包，再与组件得到一个 $V - w[i] + 1$ 个物品组。
更一般问题，依赖关系由「森林」形式给出，涉及到树形 DP 以及泛化物品，这里不表。

- 背包问题方案总数

$$f[i,j] = sum f[i-1,j], f[i-1,j-w[i]] + v[i], f[0,0] = 0$$

> 更多内容详见「背包九讲」

Maximum Subarray Sum

```
int max_subarray_sum(int A[], int n) {
    int res, cur;
    if (!A || n <= 0) return 0;
    res = cur = a[0];
    for (int i = 0; i < n; ++i) {
        if (cur < 0) cur = a[i];
        else cur += a[i];
        res = max(cur, res);
    }
    return res;
}
```

Set

```cpp
// 子集枚举
int sub = sup;
do {
    sub = (sub - 1) & sup;
} while (sub != sup); // -1 & sup = sup;

// 势为 k 的集合枚举
int comb = (1 << k) - 1;
while (comb < 1 << n) {
    int x = comb & -comb, y = comb + x;
    comb = ((comb & ~y) / x >> 1) | y;
}

// 排列组合
do {

} while (next_permutation(A, A + N)); // prev_permutation
```

Data Structure

```cpp
// Heap
int heap[MAX_N], sz = 0;
void push(int x) {
    int i = sz++;

    while (i > 0) {
        int p = (i - 1) / 2;
        if (heap[p] <= x) break;
        heap[p] = heap[i];
        i = p;
    }
    heap[i] = x;
}
int pop() {
    int ret = heap[0];
    int x = heap[--sz];
    int i = 0;
    while (i * 2 + 1 < sz) {
        int a = i * 2 + 1, b = i * 2 + 2;
        if (b < sz && heap[b] < heap[a]) a = b;
        if (heap[a] >= x) break;
        heap[i] = heap[a];
        i = a;
    }
    heap[i] = x;
    return ret;
}
```

```cpp
// Binary Search Tree
struct node {
    int val;
    node *lch, rch;
};

node *insert(node *p, int x) {
    if (p == NULL) {
        node *q = new node;
        q->val = x;
        q->lch = q->rch = NULL;
        return q;
    } else {
        if (x < p->val) p->lch = insert(p->lch, x);
```

```
        else p->rch = insert(p->rch, x);
        return p;
    }
}
bool find(node *p, int x) {
    if (p == NULL) return false;
    else if (x == p->val) return true;
    else if (x < p->val) return find(p->lch, x);
    else return find(p->rch, x);
}
node *remove(node *p, int x) {
    if (p == NULL) return NULL;
    else if (x < p->val) p->lch = remove(p->lch, x);
    else if (x > p->val) p->rch = remove(p->rch, x);
    else if (p->lch == NULL) {
        node *q = p->rch;
        delete p;
        return q;
    } else if (p->lch->rch == NULL) {
        node *q = p->lch;
        q->rch = p->rch;
        delete p;
        return q;
    }  else {
        // 把左儿子子孙中最大的节点提到需要删除的节点上
        node *q;
        for (q = p->lch; q->rch->rch != NULL; q = q->rch);
        node *r = q->rch;
        q->rch = r->lch;
        r->lch = p->lch;
        r->rch = p->rch;
        delete p;
        return r;
    }
    return p;
}
```

```
// Union-find Set
int par[MAX_N];
int rnk[MAX_N];
void init(int n) {
    for (int i = 0; i < n; ++i) {
        par[i] = i;
        rnk[i] = 0;
    }
}
int find(int x) {
    return par[x] == x? x : par[x] = find(par[x]);
}
bool same(int x, int y) {
    return find(x) == find(y);
}
void unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (rnk[x] < rnk[y]) {
        par[x] = y;
    } else {
        par[y] = x;
        if (rnk[x] == rnk[y]) rnk[x]++;
    }
}
```

当然，更快捷简单的做法，是使用 C++ 的 container。

Graph

```cpp
struct edge {
    int from;
    int to, dis;
};
vector<edge> G[MAX_V];
vector<edge> es;
bool vis[MAX_V];
int V, E, pre[MAX_V], dist[MAX_V];
// int cost[MAX_V][MAX_V];
```

```cpp
// Shortest Way
void dijkstra(int s) {
    priority_queue<Pii, vector<Pii>, greater<Pii> > que;// fisrt 是最短距离，second 是顶点编号
    fill(dist, dist + V, INF);
    dist[s] = 0; que.push(Pii(0, s));
    while (!que.empty()) {
        Pii p = que.top(); que.pop();
        int v = p.second;
        if (dist[v] < p.first) continue;
        for (int i = 0; i < G[v].size(); i++) {
            edge e = G[v][i];
            if (dist[e.to] > dist[v] + e.dis) {
                dist[e.to] = dist[v] + e.dis;
                que.push(Pii(dist[e.to], e.to));
            }
        }
    }
}
void bellman_ford(int s) {
    fill(dist, dist + V, INF);
    dist[s] = 0;
    while (true) {
        bool update = false;
        for (int i = 0; i < E; ++i) {
            edge e = es[i];
            if (dist[e.from] != INF && dist[e.from] + e.dis < dist[e.to]) {
                update = true;
                dist[e.to] = dist[e.from] + e.dis;
            }
        }
        if (!update) break;
    }
}
bool find_negative_loop() {
    memset(dist, 0, sizeof dist);
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < E; ++j) {
            edge e = es[j];
            if (d[e.to] > d[e.from] + e.dis) {
                d[e.to] = d[e.from] + e.dis;
                if (i == V - 1) return true;
            }
        }
    return false;
}
void spfa(int s) {
    queue<int> que;
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[s] = 0; que.push(s); vis[s] = true;
    while (!que.empty()) {
        int v = que.front(); que.pop();
        vis[v] = false;
        for (int i = 0; i < G[v].size(); ++i) {
            int u = G[v][i].to;
            if (dist[u] > dist[v] + G[v][i].dis) {
                dist[u] = dist[v] + G[v][i].dis;
                if (!vis[u]) {
                    que.push(u);
```

```
                    vis[u] = true;
                }
            }
        }
    }
}
```

```
// Spanning Tree
int prime() {
    /*
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[0] = 0;
    int res = 0;
    while (true) {
        int v = -1;
        for (int u = 0; u < V; ++u) {
            if(!vis[u] && (v == -1 || dist[u] < dist[v])) v = u;
        }
        if (v == -1) break;
        vis[v] = true;
        res += dist[v];
        for (int u = 0; u < V; u++)
            dist[u] = min(dist[u], cost[v][u]);
    }
    //*/
    priority_queue<Pii, vector<Pii>, greater<Pii> > que;
    int res = 0;
    fill(dist, dist + V, INF);
    fill(vis, vis + V, false);
    dist[0] = 0;
    que.push(Pii(0, 0));
    while (!que.empty()) {
        Pii p = que.top(); que.pop();
        int v = p.second;
        if (vis[v] || dist[v] < p.first) continue;
        res += dist[v]; vis[v] = true;
        for (int i = 0; i < G[v].size(); ++i) {
            edge e = G[v][i];
            if (dist[e.to] > e.dis) {
                dist[e.to] = e.dis;
                que.push(Pii(dist[e.to], e.to));
            }
        }
    }
    return res;
}

bool cmp(const edge e1, const edge e2) {
    return e1.dis < e2.dis;
}
int kruskal() {
    sort(es.begin(), es.end(), cmp);
    init(V);
    int res = 0;
    for (int i = 0; i < E; ++i) {
        edge e = es[i];
        if (!same(e.from, e.to)) {
            unite(e.from, e.to);
            res += e.dis;
        }
    }
    return res;
}
```

```
// Network Flow(Dinic)
void add_edge(int from, int to, int cap) {
    G[from].push_back((edge){to, cap, static_cast<int>(G[to].size()), 1});
```

```
    G[to].push_back((edge){from, 0, static_cast<int>(G[from].size() - 1), 1});
}

void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while (!que.empty()) {
        int v = que.front(); que.pop();
        for (int i = 0; i < G[v].size(); ++i) {
            edge &e = G[v][i];
            // cout << v << "->" << e.to << " : " << e.cap << endl;
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }
}

int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int &i = iter[v]; i < G[v].size(); ++i) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof iter);
        int f;
        while ((f = dfs(s, t, INF)) > 0) {
            flow += f;
        }
    }
}
```

Math Problem

```
// returning count of nk in range [l, r], from Infinity
template<typename T> T mps(T l, T r, T k) {
    return ((r - (r % k + k) % k) - (l + (k - l % k) % k)) / k + 1;
}
template<typename T> T gcd(T a, T b) {
    //return (b)? gcd(b, a  % b) : a;
    while (b) { T t = a % b; a = b; b = t; } return a;
}
template<typename T> T lcm(T a, T b) {
    return a / gcd(a, b) * b;
}
// find (x, y) s.t. a x + b y = gcd(a, b) = d
template<typename T> T exgcd(T a, T b, T &x, T &y) {
    T d = a;
    if (b) {
```

```
        d = exgcd(b, a % b, y, x);
        y -= a / b * x;
    } else {
        x = 1; y = 0;
    }
    return d;
}
template<typename T> T mod_mult(T a, T b, T mod) {
  T res = 0;
  while (b) {
    if (b & 1) {
        res = (res + a) % mod;
        // res += a;
        // if (res >= mod) res -= mod;
    }
    a = (a + a) % mod;
    // a <<= 1;
    // if (a >= mod) a -= mod;
    b >>= 1;
  }
  return res;
}
template<typename T> T mod_pow(T x, T n, T mod) {
    T res = 1;
    while (n) {
        if (n & 1) res = mod_mult(res, x, mod);
        x = mod_mult(x, x, mod);
        n >>= 1;
    }
    return res;
    // return b ? mod_pow(a * a % mod, b >> 1, mod) * (b & 1 ? a : 1) % mod : 1;
}
template<typename T> T mod_inverse(T a, T m) {
    T x, y;
    exgcd(a, m, x, y);
    return (m + x % m) % m;
}
ll CRT(vector<ll> &a, vector<ll> &m) {
    ll M = 1LL, res = 0;
    for (int i = 0; i < m.size(); ++i)
        M *= m[i];
    for (int i = 0; i < m.size(); ++i) {
        ll Mi, Ti;
        Mi = M / m[i]; Ti = mod_inverse(Mi, mi);
        res = (res + a[i] * (Mi * Ti % M) % M) % M;
    }
    return res;
}
// only for MOD < 1e6;
ll fact[MOD + 10];
void init() {
    fact[0] = 1;
    for (int i = 1; i <= MOD; ++i)
        fact[i] = fact[i - 1] * i % MOD;
}
int mod_fact(int n, int p, int &e) {
    e = 0;
    if (n == 0) return 1;
    int res = mod_fact(n / p, p, e);
    e += n / p;
    if (n / p % 2 != 0) return res * (p - fact[n % p]) % p;
    return res * fact[n % p] % p;
}
int mod_comb(int n, int k, int p) {
    if (n < 0 || k < 0 || n < k) return 0;
    if (n == 0) return 1;
    int e1, e2, e3;
    int a1 = mod_fact(n, p, e1), a2 = mod_fact(k, p, e2), a3 = mod_fact(n - k, p, e3);
    if (e1 > e2 + e3) return 0;
    return a1 * mod_inverse(a2 * a3 % p, p) % p;
}
```

```cpp
ll lucas(ll n, ll k, const ll &p) {
    if (n < 0 || k < 0 || n < k) return 0;
    if (n == 0) return 1;
    return lucas(n / p, k / p, p) * mod_comb(n % p, k % p, p) % p;
}
```

```cpp
// 矩阵快速幂
typedef vector<int> vec;
typedef vector<vec> mat;
mat G(MAX_N);

mat mat_mul(mat &A, mat &B) {
    mat C(A.size(), vec(B[0].size()));
    for (int i = 0; i < A.size(); ++i)
        for (int k = 0; k < B.size(); ++k)
            for (int j = 0; j < B[0].size(); ++j)
                C[i][j] = (C[i][j] + A[i][k] % MOD * B[k][j] % MOD + MOD) % MOD;
    return C;
}
mat pow(mat A, ll n) {
    mat B(A.size(), vec(A.size()));
    for (int i = 0; i < A.size(); ++i)
        B[i][i] = 1;
    while (n > 0) {
        if (n & 1) B = mat_mul(B, A);
        A = mat_mul(A, A);
        n >>= 1;
    }
    return B;
}
void vec_pow_mod() {
    mat A(2, vec(2));
    A[0][0] = ; A[0][1] = ;
    A[1][0] = ; A[1][1] = ;
    pow(A, n);
}
```

```cpp
// prime number
bool is_prime(int n) {
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
    return n != 1;
}
vector<int> divisor(int n) {
    vector<int> res;
    for (int i = 1; i * i <= n; ++i) {
        if (n % i == 0) {
            res.push_back(i);
            if (i != n / i) res.push_back(n / i);
        }
    }
    return res;
}
map<int, int> prime_factor(int n) {
    map<int, int> res;
    for (int i = 2; i * i <= n; ++i) {
        while (n % i == 0) {
            ++res[i];
            n /= i;
        }
    }
    if (n != 1) res[n] = 1;
    return res;
}
int prime[MAX_N];
bool isPrime[MAX_N + 1];
int seive(int n) {
    int p = 0;
```

```
    fill(isPrime, isPrime + n + 1, true);
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i <= n; ++i)
        if (isPrime[i]) {
            prime[p++] = i;
            for (int j = 2 * i; j <= n; j += i) isPrime[j] = false;
        }
    return p;
}
// the number of prime in [L, r)
// 对区间 [l, r) 内的整数执行筛法, prime[i - l] = true <=> i 是素数
bool segPrimeSmall[MAX_L];
bool segPrime[MAX_SQRT_R];
void segment_sieve(ll l, ll r) {
    for (int i = 0; (ll)i * i < r; ++i) segPrimeSmall[i] = true;
    for (int i = 0; i < r - l; ++i) segPrime[i] = true;
    for (int i = 2; (ll)i * i < r; ++i) {
        if (segPrimeSmall[i]) {
            for (int j = 2 * i; (ll)j * j <= r; j += i) segPrimeSmall[j] = false;
            for (ll j = max(2ll, (l + i - 1) / i) * i; j < r; j += i) segPrime[j - l] = false;
        }
    }
}
// Miller_Rabin
bool check(ll a, ll n, ll x, ll t) {
    ll res = mod_pow(a, x, n);
    ll last = res;
    for (int i = 1; i <= t; ++i) {
        res = mod_mult(res, res, n);
        if (res == 1 && last != 1 && last != n - 1) return true;
        last = res;
    }
    if (res != 1) return true;
    return false;
}
bool Miller_Rabin(ll n) {
    if (n < MAX_N) return isPrime[n]; // small number may get wrong answer?!
    if (n < 2) return false;
    if (n == 2) return true;
    if ((n & 1) == 0) return false;
    ll x = n - 1, t = 0;
    while ((x & 1) == 0) {
        x >>= 1;
        ++t;
    }
    for (int i = 0; i < S; ++i) {
        ll a = rand() % (n - 1) + 1;
        if (check(a, n, x, t))
            return false;
    }
    return true;
}
// find factors
vector<ll> factor;
ll Pollard_rho(ll x, ll c) {
    ll i = 1, k = 2;
    ll x0 = rand() % x;
    ll y = x0;
    while (true) {
        ++i;
        x0 = (mod_mult(x0, x0, x) + c) % x;
        ll d;
        if (y == x0) d = 1;
        else
            if (y > x0)
                d = gcd(y - x0, x);
            else d = gcd(x0 - y, x);
        if (d != 1 && d != x) return d;
        if (y == x0) return x;
        if (i == k) {
            y = x0;
```

```
            k += k;
        }
    }
}
void find_factor(ll n) {
    if (n == 1) return ;
    if (Miller_Rabin(n)) {
        factor.push_back(n);
        return ;
    }
    ll p = n;
    while (p >= n) p = Pollard_rho(p, rand() % (n - 1) + 1);
    find_factor(p);
    find_factor(n / p);
}
```

- Meisell-Lehmer
- Moebius
  如果

$$F(n) = \sum_{d|n} f(d)$$

，则

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

对于 $\mu(d)$ 函数，有如下性质：

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{n} = 1 \\ 0 & \text{n} > 1 \end{cases}$$

$$\sum_{d|n} \frac{\mu(d)}{d} = \frac{\phi(n)}{n}$$

```
int mu[MAX_N];
void moebius() {
    int cnt = 0; mu[1] = 1;
    memset(vis, 0, sizeof vis);
    for (int i = 2; i < MAX_N; ++i) {
        if (!vis[i]) {
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < cnt && i * prime[j] < MAX_N; ++j) {
            vis[i * prime[j]] = true;
            if (i % prime[j])
                mu[i * prime[j]] = -mu[i];
            else
                mu[i * prime[j]] = 0, break;
        }
    }
}
```

```
// Guass_jordan
const double eps = 1e-8;
typedef vector<double> vec;
typedef vector<vec> mat;

vec gauss_joedan(const mat &A, const vec& b) {
    int n = A.size();
    mat B(n, vec(n + 1));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
    for (int i = 0; i < n; ++i) B[i][n] = b[i];
```

```
    for (int i = 0; i < n; ++i) {
        int pivot = i;
        for (int j = i; j < n; ++j) {
            if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
        }
        if (i != pivot) swap(B[i], B[pivot]);

        if (abs(B[i][i]) < eps) return vec();

        for (int j = i + 1; j <= n; ++j) B[i][j] /= B[i][i];
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                for (int k = i + 1; k <= n; ++k) B[j][k] -= B[j][i] * B[i][k];
            }
        }
    }
    vec x(n);
    for (int i = 0; i < n; ++i) x[i] = B[i][n];
    return x;
}

vec gauss_joedan_xor(const mat& A, const vec& b) {
    int n = A.size();
    mat B(n, vec(n + 1));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) B[i][j] = A[i][j];
    for (int i = 0; i < n; ++i) B[i][n] = b[i];

    for (int i = 0; i < n; ++i) {
        int pivot = i;
        for (int j = i; j < n; ++j)
            if (B[j][i]) {
                pivot = j;
                break;
            }
        if (pivot != i) swap(B[i], B[pivot]);

        for (int j = 0; j < n; ++j) if (i != j && B[j][i]) {
                for (int k = i + 1; k <= n; ++k) B[j][k] ^= B[i][k];
            }
    }

    vec x(n);
    for (int i = 0; i < n; ++i) x[i] = B[i][n];
    return x;
}
```

String

1. Hash
2. KMP
3. Extend KMP
4. trie树 poj2001 2503 3630 1056 hdu 1075 1251 1247 1298 1671
5. Manacher算法
6. AC自动机
7. 后缀数组
8. 后缀树
9. 后缀自动机
10. 回文自动机

```
// 最小最大表示法:
int getMinString(const string &s) {
    int len = (int)s.length();
```

```
    int i = 0, j = 1, k = 0;
    while(i < len && j < len && k < len) {
        int t = s[(i+k)%len] - s[(j+k)%len];
        if(t == 0) k++;
        else {
            if(t > 0) i += k + 1;//getMaxString: t < 0
            else j += k + 1;
            if(i==j) j++;
            k = 0;
        }
    }
    return min(i,j);
}
```

```
// KMP
int nxt[MAX_N];
void getNext(const string &str) {
    int len = str.length();
    int j = 0, k;
    k = nxt[0] = -1;
    while (j < len) {
        if (k == -1 || str[j] == str[k])
            nxt[++j] = ++k;
        else k = nxt[k];
    }
}
int kmp(const string &tar, const string &pat) {
    getNext(pat);
    int num, j, k;
    int lenT = tar.length(), lenP = pat.length();
    num = j = k = 0;
    while (j < lenT) {
        if(k == -1 || tar[j] == pat[k])
            j++, k++;
        else k = nxt[k];
        if(k == lenP) {
            // res = max(res, j - lenP);
            k = nxt[k];
            ++num;
        }
    }
    return num;//lenP - res - 1;
}
```

```
// AC 自动机
int ans[MAX_N], d[MAX_N];

struct Trie {
    int nxt[MAX_N][26], fail[MAX_N], end[MAX_N];
    int root, L;
    int newnode() {
        for(int i = 0; i < 26; i++)
            nxt[L][i] = -1;
        end[L++] = 0;
        return L-1;
    }
    void init() {
        L = 0;
        root = newnode();
    }
    void insert(char buf[]) {
        int len = strlen(buf);
        int now = root;
        for(int i = 0; i < len; i++) {
            if(nxt[now][buf[i]-'a'] == -1)
                nxt[now][buf[i]-'a'] = newnode();
            now = nxt[now][buf[i]-'a'];
        }
```

```
        end[now] = 1;
        d[now] = len;
    }
    void build() {
        queue<int> Q;
        fail[root] = root;
        for(int i = 0; i < 26; i++)
            if(nxt[root][i] == -1)
                nxt[root][i] = root;
            else {
                fail[nxt[root][i]] = root;
                Q.push(nxt[root][i]);
            }
        while( !Q.empty() ) {
            int now = Q.front(); Q.pop();
            for(int i = 0; i < 26; i++)
                if(nxt[now][i] == -1)
                    nxt[now][i] = nxt[fail[now]][i];
                else {
                    fail[nxt[now][i]] = nxt[fail[now]][i];
                    Q.push(nxt[now][i]);
                }
        }
    }
    void solve(char buf[]) {
        int cur = root;
        int len = strlen(buf);
        int index;
        for(int i = 0; i < len; ++i) {
            if(buf[i] >= 'A' && buf[i] <= 'Z')
                index = buf[i] - 'A';
            else if(buf[i] >= 'a' && buf[i] <= 'z')
                index = buf[i] - 'a';
            else continue;
            cur = nxt[cur][index];
            int x = cur;
            while(x != root) {
                if(end[x]) {
                    ans[i + 1] -= 1;
                    ans[i - d[x] + 1] += 1;
                    break;
                }
                x = fail[x];
            }
        }
    }
};

Trie ac;
```