# GZHU I WANT TO EAT KFC

## c++11

```
#pragma GCC diagnostic error "-std=c++11"
```

## C++IO

```
// 简版
template <class I> void read(I& x) {
    char c;
    while ((c = getchar()) < '0' && c > '9');
    for (x = c - '0'; (c = getchar()) >= '0' && c <= '9'; x = x * 10 + c - '0');
}
// io完全体
namespace io {
    const int BUFLEN = (1 << 21) + 1;
    bool EOFError;
    inline char gc() {
        static char buf[BUFLEN], *st = nullptr, *ed = nullptr;
        if (st == ed) (ed = (st = buf) + fread(buf, 1, BUFLEN, stdin));
        return (st == ed) ? -1 : (*st++);
    }
    inline bool check(char x) {
        return x == '-' || x == '\n' || x == ' ' || x == '\r' || x == '\t';
    }
    template <class I> inline void read(I& x) {
        char c; int f = 1;
        while (check(c = gc())) if (c == '-') f = -1;
        if (c == -1)  { EOFError = 1; return; }
        for (x = c - '0'; (c = gc()) >= '0' && c <= '9'; x = x * 10 + (c & 15)); x *= f;
    }
    inline void gstr(char *s, int len) {
        char c; for (c = gc(); c < 'a' || c > 'z'; c = gc());
        if (c == -1) return;
        for (len = 0; c >= 'a' && c <= 'z'; c = gc()) s[len++] = c; s[len] = 0;
    }
    char obuf[BUFLEN], *ost = obuf, *oed = obuf + BUFLEN - 1, Stack[55], Top;
    inline void flush() { fwrite(obuf, 1, ost - obuf, stdout); ost = obuf; }
    inline void pc(char x) { *ost++ = x; if (ost == oed) flush(); }
    template <class I> inline void print(I x) {
        if (!x) pc('0');
        if (x < 0) pc('-'), x = -x;
        while (x) Stack[++Top] = x % 10 + '0', x /= 10;
        while (Top) pc(Stack[Top--]);
```

```
    }
    template <class I> inline void println(I x) { print(x), pc('\n'); }
    inline void pstr(char *s) { for (int i = 0, n = strlen(s); i < n; i++) pc(s[i]); }
    struct IOFLUSHER { ~IOFLUSHER() { flush(); } } _ioflusher_;
}
using namespace io;
```

# Java template

- BigInteger

| Modifier and Type | Method and Description |
|---|---|
| `BigInteger` | `abs()`<br>Returns a BigInteger whose value is the absolute value of this BigInteger. |
| `BigInteger` | `add(BigInteger val)`<br>Returns a BigInteger whose value is `(this + val)`. |
| `BigInteger` | `and(BigInteger val)`<br>Returns a BigInteger whose value is `(this & val)`. |
| `BigInteger` | `andNot(BigInteger val)`<br>Returns a BigInteger whose value is `(this & ~val)`. |
| `int` | `bitCount()`<br>Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit. |
| `int` | `bitLength()`<br>Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit. |
| `byte` | `byteValueExact()`<br>Converts this `BigInteger` to a `byte`, checking for lost information. |
| `BigInteger` | `clearBit(int n)`<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared. |
| `int` | `compareTo(BigInteger val)`<br>Compares this BigInteger with the specified BigInteger. |
| `BigInteger` | `divide(BigInteger val)`<br>Returns a BigInteger whose value is `(this / val)`. |
| `BigInteger[]` | `divideAndRemainder(BigInteger val)`<br>Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`. |
| `double` | `doubleValue()`<br>Converts this BigInteger to a `double`. |
| `boolean` | `equals(Object x)`<br>Compares this BigInteger with the specified Object for equality. |
| `BigInteger` | `shiftRight(int n)`<br>Returns a BigInteger whose value is `(this >> n)`. |
| `short` | `shortValueExact()`<br>Converts this `BigInteger` to a `short`, checking for lost information. |
| `int` | `signum()`<br>Returns the signum function of this BigInteger. |
| `BigInteger` | `subtract(BigInteger val)`<br>Returns a BigInteger whose value is `(this - val)`. |
| `boolean` | `testBit(int n)`<br>Returns `true` if and only if the designated bit is set. |
| `byte[]` | `toByteArray()`<br>Returns a byte array containing the two's-complement representation of this BigInteger. |
| `String` | `toString()`<br>Returns the decimal String representation of this BigInteger. |
| `String` | `toString(int radix)`<br>Returns the String representation of this BigInteger in the given radix. |
| `BigInteger` | `xor(BigInteger val)`<br>Returns a BigInteger whose value is `(this ^ val)`. |

| | |
|---|---|
| BigInteger | **mod**(BigInteger m)<br>Returns a BigInteger whose value is (this mod m). |
| BigInteger | **modInverse**(BigInteger m)<br>Returns a BigInteger whose value is ($this^{-1}$ mod m). |
| BigInteger | **modPow**(BigInteger exponent, BigInteger m)<br>Returns a BigInteger whose value is ($this^{exponent}$ mod m). |
| BigInteger | **multiply**(BigInteger val)<br>Returns a BigInteger whose value is (this * val). |
| BigInteger | **negate**()<br>Returns a BigInteger whose value is (-this). |
| BigInteger | **nextProbablePrime**()<br>Returns the first integer greater than this BigInteger that is probably prime. |
| BigInteger | **not**()<br>Returns a BigInteger whose value is (~this). |
| BigInteger | **or**(BigInteger val)<br>Returns a BigInteger whose value is (this \| val). |
| BigInteger | **pow**(int exponent)<br>Returns a BigInteger whose value is ($this^{exponent}$). |
| BigInteger | **remainder**(BigInteger val)<br>Returns a BigInteger whose value is (this % val). |
| BigInteger | **setBit**(int n)<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set. |
| BigInteger | **shiftLeft**(int n)<br>Returns a BigInteger whose value is (this << n). |

| | |
|---|---|
| BigInteger | **flipBit**(int n)<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped. |
| float | **floatValue**()<br>Converts this BigInteger to a float. |
| BigInteger | **gcd**(BigInteger val)<br>Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val). |
| int | **getLowestSetBit**()<br>Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit). |
| int | **hashCode**()<br>Returns the hash code for this BigInteger. |
| int | **intValue**()<br>Converts this BigInteger to an int. |
| int | **intValueExact**()<br>Converts this BigInteger to an int, checking for lost information. |
| boolean | **isProbablePrime**(int certainty)<br>Returns true if this BigInteger is probably prime, false if it's definitely composite. |
| long | **longValue**()<br>Converts this BigInteger to a long. |
| long | **longValueExact**()<br>Converts this BigInteger to a long, checking for lost information. |
| BigInteger | **max**(BigInteger val)<br>Returns the maximum of this BigInteger and val. |
| BigInteger | **min**(BigInteger val)<br>Returns the minimum of this BigInteger and val. |

---

## *Java*

```java
// min
import java.io.BufferedInputStream;
import java.io.PrintWriter;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(new BufferedInputStream(System.in));
        PrintWriter out = new PrintWriter(System.out);
        Task solver = new Task();
        try {
```

```
            solver.solve(in, out);
        } catch (NoSuchElementException e) {
            out.close();
        }
    }
    private static BigInteger[] a = new BigInteger[1005];
    public static class Task {
        public static void solve(Scanner in, PrintWriter out) {
            // Your code start here.

        }
    }
}
```

```
// max
import java.io.*;
import java.util.StringTokenizer;

public class Main {
    public static void main(String[] args) {
        InputReader in = new InputReader(System.in);
        PrintWriter out = new PrintWriter(System.out);
        Task solver = new Task();
        try {
            while (true) {
                solver.solve(in, out);
            }
        } catch(RuntimeException e) {
            out.close();
        }
    }

    public static class Task {
        void solve(InputReader in, PrintWriter out) {
            // Code start here

        }
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
```

```
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }
        public long nextLong() {
            return Long.parseLong(next());
        }
        public double nextDouble() {
            return Double.parseDouble(next());
        }
        public BigInteger nextBigInteger() {
            return new BigInteger(next());
        }
        public BigDecimal nextBigDecimal() {
            return new BigDecimal(next());
        }
    }
}
```

---

# 数据结构

## *Fenwic_Tree(树状数组)*

- 维护&求值均为$O(\log n)$

```
struct Fenwick {
    static const int maxn = 1e5 + 5;
    int a[maxn];
    void init() { memset(a, 0, sizeof a); }
    inline int lowbit(int i) { return (i & (-i)); }
    void upd(int i, int x) {
        for (; i < maxn; i += lowbit(i)) {
            a[i] += x;
        }
    }
    int sum(int i) {
        int res = 0;
        for (; i > 0; i -= lowbit(i)) {
            res += a[i];
        }
        return res;
    }
    inline int query(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    int upper_bound(int x) {
        int res = 0, ptr = 0;
        while ((1 << (ptr + 1)) <= n) ++ptr;
        for (; ptr >= 0; ptr--) {
```

```
                int p = res + (1 << ptr);
                if (p <= n && a[p] <= x) {
                    x -= a[p];
                    res += 1 << ptr;
                }
            }
            return res;
        }
    };
```

---

## *spare_table_rmq*

- build $O(n \log n)$, query $O(1)$

```
    const int MAX = 1e6 + 5;

    int dp[MAX][25];

    void rmq(int n) {
        int len = (int)(log(n) / log(2.0));
        for (int j = 1; j <= len; j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int l, int r) {
        int p = (int)(log(r - l + 1) / log(2.0));
        return min(dp[l][p], dp[r - (1 << p) + 1][p]);
    }
```

---

## *Disjoin_set_union(并查集)*

```
    struct Dsu {
        static const int maxn = 1e5 + 5;
        int fa[maxn], sz[maxn];
        void init(int n) {
            for (int i = 0; i <= n; i++) {
                fa[i] = i, sz[i] = 0;
            }
        }
        int find(int x) {
            return x == fa[x] ? x : fa[x] = find(fa[x]);
        }
```

```
    bool unite(int u, int v) {
        int a = find(u), b = find(v);
        if (a == b) return false;
        if (sz[a] < sz[b]) fa[a] = b;
        else fa[b] = a, sz[a] += (sz[a] == sz[b]);
        return true;
    }
};
```

---

## *Segment_Tree(线段树)*

- build $O(n \log n)$, update&modify $O(\log n)$, query $O(\log n)$

```
struct Segment_Tree {
    static const int maxn = 1e3 + 5;
    int a[maxn << 2], lazy[maxn << 2], cover[maxn << 2];
    bool covered[maxn << 2];

#define lson i << 1, l, mid
#define rson i << 1 | 1, mid + 1, r
#define Lson i << 1
#define Rson i << 1 | 1

    void build(int i, int l, int r, int *arr) {
        a[i] = lazy[i] = 0;
        if (l == r) {
            a[i] = arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(lson, arr), build(rson, arr);
    }

    inline void pushdown(int i) {
        if (covered[i]) {
            a[Lson] = a[Rson] = cover[Lson] = cover[Rson] = cover[i];
            covered[Lson] = covered[Rson] = true;
            lazy[Lson] = lazy[Rson] = 0;
            covered[i] = false;
        }
        if (lazy[i]) {
            a[Lson] += lazy[i];
            a[Rson] += lazy[i];
            lazy[Lson] += lazy[i];
            lazy[Rson] += lazy[i];
            lazy[i] = 0;
        }
    }

    inline void pushup(int i) {
        a[i] = min(a[Lson], a[Rson]);
```

```
        }

        void update(int L, int R, int i, int l, int r, int x) {
            if (L <= l && r <= R) {
                a[i] += x, lazy[i] += x;
                return;
            }
            pushdown(i);
            int mid = (l + r) >> 1;
            if (R <= mid) update(L, R, lson, x);
            else if (mid < L) update(L, R, rson, x);
            else update(L, R, lson, x), update(L, R, rson, x);
            pushup(i);
        }

        void modify(int L, int R, int i, int l, int r, int x) {
            if (L <= l && r <= R) {
                a[i] = x; lazy[i] = 0; cover[i] = x; covered[i] = true;
                return;
            }
            pushdown(i);
            int mid = (l + r) >> 1;
            if (R <= mid) modify(L, R, lson, x);
            else if (mid < L) modify(L, R, rson, x);
            else modify(L, R, lson, x), modify(L, R, rson, x);
            pushup(i);
        }

        int query(int L, int R, int i, int l, int r) {
            if (L <= l && r <= R) {
                return a[i];
            }
            pushdown(i);
            int mid = (l + r) >> 1;
            if (R <= mid) return query(L, R, lson);
            else if (mid < L) return query(L, R, rson);
            else return min(query(L, R, lson), query(L, R, rson));
        }
    };
```

---

## *leftist_heap(左偏堆)*

- 均摊$O(\log n)$

```
    struct leftist_heap {
        static const int maxn = 2e5 + 5;
        static const int maxm = 1e3 + 5;

        int root[maxm], tot;
        int v[maxn], ch[maxn][2], size[maxn], stk[maxn], tp;
```

```
        inline void init(int &n) {
            tot = tp = 0;
            memset(root, 0, (n + 1) * sizeof(int));
        }

        inline int newnode(int x) {
            int rt = (tp > 0) ? stk[--tp] : (tot++);
            v[rt] = x, size[rt] = 1, ch[rt][0] = ch[rt][1] = 0;
            return rt;
        }

        int merge(int a, int b) {
            if (!a || !b) {
                return a ? a : b;
            }
            if (v[a] > v[b]) {
                swap(a, b);
            }
            ch[a][1] = merge(ch[a][1], b);
            if (size[ch[a][0]] < size[ch[a][1]]) {
                swap(ch[a][0], ch[a][1]);
            }
            size[a] = size[ch[a][1]] + 1;
            return a;
        }

        void Merge(int a, int b) {
            root[a] = merge(root[a], root[b]);
            root[b] = 0;
        }

        void insert(int i, int val) {
            int tmp = newnode(val);
            root[i] = merge(root[i], tmp);
        }

        inline int top(int i) {
            return (root[i] > 0) ? v[root[i]] : -1;
        }

        inline void pop(int i) {
            (root[i] > 0) ? (stk[tp++] = root[i], root[i] = merge(ch[root[i]][0], ch[root[i]]
[1])) : 0;
        }
    };
```

---

## *Trie(字典树) $O(len)$*

```
    struct Trie {
        static const int maxn = 1e5 + 5;
        int tot, ch[maxn][30], ed[maxn], cnt[maxn];
```

```
    void init() { tot = 0; cnt[0] = 0; ed[0] = 0; memset(ch[0], 0, sizeof ch[0]); }

    int newnode() {
        cnt[++tot] = 0; ed[tot] = 0;
        memset(ch[tot], 0, sizeof ch[tot]);
        return tot;
    }

    void insert(char *s, int len) {
        int now = 0;
        for (int i = 0; i < len; i++) {
            if (ch[now][s[i] - 'a'] == 0) {
                ch[now][s[i] - 'a'] = newnode();
            }
            cnt[now]++;
            now = ch[now][s[i] - 'a'];
        }
        cnt[now]++;
        ed[now] = 1;
    }

    int count(char *s, int len) {
        int now = 0;
        for (int i = 0; i < len; i++) {
            if (ch[now][s[i] - 'a'] == 0) {
                return 0;
            }
            now = ch[now][s[i] - 'a'];
        }
        return cnt[now];
    }

    bool check(char *s, int len) {
        int now = 0;
        for (int i = 0; i < len; i++) {
            if (ch[now][s[i] - 'a'] == 0) {
                return false;
            }
            now = ch[now][s[i] - 'a'];
        }
        return ed[now];
    }
};
```

# BM推公式大法

```
#include <bits/stdc++.h>
using namespace std;
struct BM {
    static const int MAXN = 10005;
```

```
    int n;
    vector<double> ps[MAXN];
    int pn, fail[MAXN];
    double delta[MAXN];

    void Solve(double x[], const int& n)
    {
        pn = 0;
        memset(fail, 0, sizeof fail);
        memset(delta, 0, sizeof delta);

        ps[0].clear();
        for (int i = 1; i <= n; i++) {
            double dt = -x[i];
            for (int j = 0; j < ps[pn].size(); j++) {
                dt += x[i - j - 1] * ps[pn][j];
            }
            delta[i] = dt;

            if (fabs(dt) <= 1e-8) continue;
            fail[pn] = i;
            if (!pn) {
                ps[++pn].resize(1);
                continue;
            }

            vector<double>& ls = ps[pn - 1];
            double k = -dt / delta[fail[pn - 1]];
            vector<double> cur(i - fail[pn - 1] - 1);
            cur.push_back(-k);

            for (int j = 0; j < ls.size(); j++) {
                cur.push_back(ls[j] * k);
            }
            if (cur.size() < ps[pn].size()) {
                cur.resize(ps[pn].size());
            }
            for (int j = 0; j < ps[pn].size(); j++) {
                cur[j] += ps[pn][j];
            }

            ps[++pn] = cur;
        }
    }

    void print()
    {
        for (int i = 0; i < ps[pn].size(); i++) {
            printf("%lf ", ps[pn][i]);
        }
        printf("\n");
    }
}B;

double x[BM::MAXN];
```

```
int main()
{
    int n;
    while (scanf("%d", &n) == 1) {
        for (int i = 1; i <= n; i++) {
            scanf("%lf", &x[i]);
        }
        B.Solve(x, n);
        B.print();
    }
}
```

## 最长递增子序列$O(n \log n)$

```
const int MAX = 500005;
typedef long long ll;

ll v[MAX], stack[MAX];

int upb(int l, int r, int k) {
    int mid;
    while (l < r) {
        mid = (l + r) >> 1;
        if (stack[mid] > k) r = mid;
        else l = mid + 1;
    }
    return r;
}

void solve() {
    int n; cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i];

    int top = 0;
    for (int i = 1; i <= n; i++) {
        if (top == 0) stack[++top] = v[i];
        else if (stack[top] <= v[i]) stack[++top] = v[i];
        else {
            int pos = upb(1, top, v[i]);
            stack[pos] = v[i];
        }
    }
    cout << top << endl;
}
```

## 最长回文子序列(记忆化搜索)$O(n^2)$

```
int len;
char v[1300];
int dp[1300][1300];

int dfs(int l, int r) {
    if (l == r) return 1;
    if (l > r) return 0;
    if (dp[l][r] != -1) return dp[l][r];

    int res = max(dfs(l, r - 1), dfs(l + 1, r));
    if (v[l] == v[r]) {
        res = max(res, dfs(l + 1, r - 1) + 2);
    }
    else {
        return dp[l][r] = res;
    }
}

int main() {
    while (scanf("%s", v) != EOF) {
        len = strlen(v);
        for (int i = 0; i < len; i++) {
            v[i] = tolower(v[i]);
        }
        memset(dp, -1, sizeof dp);
        printf("%d\n", len - dfs(0, len - 1));
    }
}
```

---

# FastTransform

## $FFT\ O(n \log n)$

```
/*
    Example:
        Complex[] a; int len1 = length(a);
        Complex[] b; int len2 = length(b);
        int Len = FFT::trans(len1 + len2 - 1);
        FFT::DFT(a, Len, 1);
        FFT::DFT(b, Len, 1);
        for i in range(0, l):
            a[i] *= b[i]
        FFT::DFT(a, Len, -1);
*/
namespace FFT {
    const double PI = acos(-1.0);
    // Complex
    struct Complex {
        double r, i;
        Complex(double x = 0, double y = 0) : r(x), i(y) {}
```

```
        Complex(int n) : r(cos(2 * PI / n)), i(sin(2 * PI / n)) {}
        Complex operator + (const Complex& b)const {
            return Complex(r + b.r, i + b.i);
        }
        Complex operator - (const Complex& b)const {
            return Complex(r - b.r, i - b.i);
        }
        Complex operator * (const Complex& b)const {
            return Complex(r * b.r - i * b.i, r * b.i + i * b.r);
        }
        friend Complex& operator *= (Complex& a, const Complex& b) {
            a = a * b;
            return a;
        }
    };

    // bit reverse
    void rev(Complex *a, int n) {
        for (int i = 1, j = n >> 1, k; i < n - 1; i++) {
            if (i < j) swap(a[i], a[j]);
            for (k = n >> 1; j >= k; j -= k, k >>= 1);
            j += k;
        }
    }

    // Discrete Fourier transform
    // t ->  1, DFT
    // t -> -1, IDFT
    void DFT(Complex *a, int n, int t) {
        rev(a, n);
        for (int i = 2; i <= n; i <<= 1) {
            Complex wi(i * t);
            for (int j = 0; j < n; j += i) {
                Complex w(1, 0);
                for (int k = j, h = i >> 1; k < j + h; k++) {
                    Complex t = w * a[k + h], u = a[k];
                    a[k] = u + t;
                    a[k + h] = u - t;
                    w *= wi;
                }
            }
        }
        if (t == -1) for (int i = 0; i < n; i++) a[i].r /= n;
    }

    // Get FFT_Len
    // min(2 ^ p) which (2 ^ p) > x
    int trans(int x) {
        int i = 0;
        for (; x > (1 << i); i++);
        return 1 << i;
    }
}
```

# $FWT\ O(n \log n)$

```cpp
const int MOD = 1e9 + 7;

int qpow(int a, int t) {
    int b = 1;
    while (t > 1) {
        if (t & 1) b = b * a % MOD;
        a = a * a % MOD;
        t >>= 1;
    }
    return b;
}

const int inv2 = qpow(2, MOD - 2); // (x/2) % MOD == x*inv2 % MOD

inline int trans(int n) {
    int k = 1;
    for (; k < n; k <<= 1);
    return k;
}

void fwt(int a[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int i = 0, k = d << 1; i < n; i += k) {
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % MOD;
                a[i + j + d] = (x - y + MOD) % MOD;
                // xor : a[i + j] = x + y, a[i + j + d] = x - y
                // and : a[i + j] = x + y
                // or : a[i + j + d] = x + y
            }
        }
    }
}

void ifwt(int a[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int i = 0, k = d << 1; i < n; i += k) {
            for (int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = 1ll * (x + y) * inv2 % MOD;
                a[i + j + d] = (1ll * (x - y) * inv2 % MOD + MOD) % MOD;
                // xor : a[i + j] = (x + y) / 2, a[i + j + d] = (x - y) / 2
                // and : a[i + j] = x - y
                // or : a[i + j + d] = y - x;
            }
        }
    }
}
```

# Graph(图相关)

## *匈牙利二分图匹配 $O(n(n+m))$ ~~(real?)~~*

```cpp
const int maxn = 1e5 + 5;

vector<int> e[maxn];
int link[maxn], vis[maxn];

void init() {
    for (int i = 0; i < maxn; i++) {
        e[i].clear();
    }
}

void addedge(int u, int v) {
    e[u].push_back(v);
    // e[v].push_back(u);
}

bool find(int u) {
    for (int i = 0; i < (int)e[u].size(); i++) {
        int v = e[u][i];
        if (!vis[v]) {
            vis[v] = 1;
            if (link[v] == -1 || find(link[v])) {
                link[v] = u;
                // link[u] = v;
                return true;
            }
        }
    }
    return false;
}

int solve(int n) {
    int res = 0;
    memset(link, -1, sizeof link);
    for (int i = 1; i <= n; i++) {
        if (link[i] == -1) {
            memset(vis, 0, sizeof vis);
            res += find(i);
        }
    }
    return res;
}
```

- 最小点覆盖:

  点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。最小点覆盖(minimum vertex covering)就是点最少的点覆盖。

- 最小边覆盖:

边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边” 覆盖了所有“点”。最小边覆盖(minimum edge covering)就是边最少的边覆盖。

- 最大点独立集：

  独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。或者说是导出的子图是零图（没有边）的点集。最大独立集(maximum independent set)就是点最多的独立集。

- 最大边独立集：

  边独立集又称匹配

  边独立集即一个边集，满足边集中的任两边不邻接。最大边独立集(maximum edge independent set)就是边最多的边独立集

- 最大团：

  团即一个点集，集合中任两个结点相邻。最大团(maximum clique)就是点最多的团。 最小点支配集：

  支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。最小支配集(minimum dominating set)就是点最少的支配集。 最小边支配集：

  边支配集即一个边集，使得所有边至少有一条邻接边在集合里。最小边支配集(minimum edge dominating set)就是边最少的边支配集。 最小路径覆盖：

- 最小路径覆盖(path covering)：

  是“路径” 覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图G的所有顶点，即每个顶点严格属于一条路径。路径的长度可能为0(单个点)。

- 二分图性质

  二分图的 最小点覆盖数 == 最大边匹配数

  【即求最少的点使得每条边都至少和其中的一个点相关联，很显然直接取最大匹配的一段节点即可】 二分图的 最小边覆盖数 == 总顶点数 — 最大边匹配数 == 最大点独立集

  二分图的 最大点独立集 == 总顶点数 — 最大边匹配数

  【很显然的把最大匹配两端的点都从顶点集中去掉这个时候剩余的点是独立集，这是|V|-2*|M|，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持其独立集性质】 二分图的 最大边独立集 == 最大匹配数

  DAG的最小路径覆盖 == 将每个点拆点后作最大匹配，结果为n-m

  结论

  6个及以上的点集合，一定含有团或者独立集（≥3个点的大小）

---

## $KM\ O(n^3)$

```
const int INF = 0x3f3f3f3f;
const int maxn = 205;
const int N = 205;
int nx, ny; // point num
int g[maxn][maxn]; // graph
int linker[maxn], lx[maxn], ly[maxn];
int slack[N];
bool visx[N], visy[N];
bool dfs(int x) {
    visx[x] = 1;
```

```
    for (int y = 0; y < ny; y++) {
        if (visy[y]) continue;
        int tmp = lx[x] + ly[y] - g[x][y];
        if (tmp == 0) {
            visy[y] = 1;
            if (linker[y] == -1 || dfs(linker[y])) {
                linker[y] = x;
                return 1;
            }
        }
        else if (slack[y] > tmp) slack[y] = tmp;
    }
    return false;
}

int KM() {
    memset(linker, -1, sizeof linker);
    memset(ly, 0, sizeof ly);
    for (int i = 0; i < nx; i++) {
        lx[i] = -INF;
        for (int j = 0; j < ny; j++) {
            if (g[i][j] > lx[i]) {
                lx[i] = g[i][j];
            }
        }
    }
    for (int x = 0; x < nx; x++) {
        memset(slack, 0x3f, sizeof slack);
        while (1) {
            memset(visx, 0, sizeof visx);
            memset(visy, 0, sizeof visy);
            if (dfs(x)) break;
            int d = INF;
            for (int i = 0; i < ny; i++) {
                if (!visy[i] && d > slack[i]) {
                    d = slack[i];
                }
            }
            for (int i = 0; i < nx; i++) {
                if (visx[i]) {
                    lx[i] -= d;
                }
            }
            for (int i = 0; i < ny; i++) {
                if (visy[i]) {
                    ly[i] += d;
                }
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for (int i = 0; i < ny; i++) {
        if (~linker[i]) {
            res += g[linker[i]][i];
        }
    }
```

```
    return res;
}
```

## *Kruskal* $O(n \log n)$

```
const int MAX = 1e5 + 5;

struct edge {
    int u, v, w;
    edge(int uu = 0, int vv = 0, int ww = 0) : u(uu), v(vv), w(ww) {}
    bool operator< (const edge& b) const { return w < b.w; }
}e[MAX * 2];

int tot, cnt, ans, fa[MAX];

void init(int n) {
    ans = tot = cnt = 0;
    for (int i = 0; i <= n; i++) fa[i] = i;
}

inline void addedge(int u, int v, int w) { e[tot++] = edge(u, v, w); }

int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

void kruskal(int n) {
    sort(e, e + tot);
    for (int i = 0; i < tot && cnt < n - 1; i++) {
        int u = find(e[i].u), v = find(e[i].v);
        if (u != v) {
            cnt++, ans += e[i].w;
            fa[fa[v]] = fa[u];
        }
    }
}
```

## *Dijkstra* $O(n \log n)$

```
const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct edge { int v, w; edge(int vv = 0, int ww = 0) : v(vv), w(ww) {} };

vector<edge> G[MAX];
int dis[MAX], vis[MAX];
```

```
void init(int n) {
    for (int i = 0; i <= n; i++) {
        G[i].clear();
        dis[i] = INF, vis[i] = 0;
    }
}

void addedge(int u, int v, int w) {
    G[u].emplace_back(v, w);
    G[v].emplace_back(u, w);
}

inline bool chkmin(int& x, int y) { return (x > y) ? (x = y, 1) : 0; }

void Dijkstra(int st, int ed) {
    priority_queue<pair<int, int>> Q;
    Q.emplace(0, st); dis[st] = 0;
    while (!Q.empty()) {
        int u = Q.top().second; Q.pop();
        if (u == ed) return;
        if (vis[u]) continue; else vis[u] = 1;
        for (edge& e : G[u]) {
            if (!vis[e.v] && chkmin(dis[e.v], dis[u] + e.w)) {
                Q.emplace(-dis[e.v], e.v);
            }
        }
    }
}
```

## *SPFA*

```
const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct edge { int v, w; edge(int vv = 0, int ww = 0) : v(vv), w(ww) {} };

vector<edge> e[MAX];
int dis[MAX], inq[MAX];
queue<int> Q;

void init(int n) {
    for (; !Q.empty(); Q.pop());
    for (int i = 0; i <= n; i++) {
        e[i].clear();
        dis[i] = INF, inq[i] = 0;
    }
}

void addedge(int u, int v, int w) {
    e[u].emplace_back(v, w);
    e[v].emplace_back(u, w);
```

```
}

void spfa(int st, int ed) {
    Q.push(st); dis[st] = 0; inq[st] = 1;
    while (!Q.empty()) {
        int u = Q.front(); Q.pop(); inq[u] = 0;
        for (edge v : e[u]) {
            if (dis[v.v] > dis[u] + v.w) {
                dis[v.v] = dis[u] + v.w;
                if (!inq[v.v]) Q.push(v.v), inq[v.v] = 1;
            }
        }
    }
}
```

---

## *Dinic* $O(V^2E)$

```
const int MAX = 1e5 + 5;
const int INF = 0x3f3f3f3f;
struct edge {
    int v, w, next;
    edge(int vv = 0, int ww = 0, int nn = 0) : v(vv), w(ww), next(nn) {}
}e[MAX << 2];
int n, tot, ans, head[MAX], level[MAX];

void init() {
    tot = ans = 0;
    for (int i = 0; i <= n; i++) {
        head[i] = -1;
    }
}

void addedge(int u, int v, int w) {
    e[tot] = edge(v, w, head[u]); head[u] = tot++;
    e[tot] = edge(u, 0, head[v]); head[v] = tot++;
}

queue<int> Q;
bool bfs(int st, int ed) {
    for (int i = 0; i <= n; i++) level[i] = 0;
    while (!Q.empty()) Q.pop();
    Q.push(st); level[st] = 1;
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        for (int i = head[u]; ~i; i = e[i].next) {
            int v = e[i].v, w = e[i].w;
            if (level[v] == 0 && w > 0) {
                level[v] = level[u] + 1;
                Q.push(v);
            }
        }
```

```
    }
    return level[ed] != 0;
}

int dfs(int u, int ed, int flow) {
    if (u == ed) return flow;
    int ret = 0;
    for (int i = head[u]; flow > 0 && (~i); i = e[i].next) {
        int v = e[i].v;
        if (level[v] == level[u] + 1 && e[i].w > 0) {
            int tmp = dfs(v, ed, min(flow, e[i].w));
            if (tmp == 0) continue;
            e[i].w -= tmp; e[i ^ 1].w += tmp;
            flow -= tmp; ret += tmp;
        }
    }
    return ret;
}

void Dinic(int st, int ed) {
    while (bfs(st, ed)) {
        ans += dfs(st, ed, INF);
    }
}
```

---

## *Tarjan Sccno(强连通)* $O(n+m)$

```
const int MAX = 1e5 + 5;
vector<int> e[MAX];
stack<int> S;
int index, tot, def[MAX], low[MAX], ins[MAX], scc[MAX];

void init(int n) {
    for (int i = 0; i <= n; i++) {
        e[i].clear();
        scc[i] = -1;
    }
}

inline void addedge(int u, int v) {
    e[u].push_back(v);
}

void dfs(int u) {
    def[u] = low[u] = ++index;
    S.push(u); ins[u] = 1;
    int v;
    for (int i = 0; i < (int)e[u].size(); i++) {
        v = e[u][i];
        if (!def[v]) {
            dfs(v);
```

```
            low[u] = min(low[u], low[v]);
        }
        else if (ins[v]) {
            low[u] = min(low[u], def[v]);
        }
    }
    if (def[u] == low[u]) {
        ++tot;
        do {
            v = S.top();
            S.pop(); ins[v] = 0;
            scc[v] = tot;
        } while (u != v);
    }
}

void solve(int n) {
    index = tot = 0;
    for (int i = 0; i <= n; i++) {
        def[i] = low[i] = 0;
    }
    while (!S.empty()) S.pop();
    for (int i = 1; i <= n; i++) {
        (!def[i]) && (dfs(i), 1);
    }
}
```

## 倍增法LCA $O(\log n)$

```
const int maxn = 1e4 + 5;

vector<int> e[maxn];
int dep[maxn], dp[maxn][15], maxb;

void init(int n) {
    for (int i = 0; i < maxn; i++) {
        e[i].clear(), dep[i] = 0;
        memset(dp[i], -1, sizeof dp[i]);
    }
    for (maxb = 0; (1 << maxb) <= n; ++maxb);
}

void DFS(int u, int d, int pre) {
    dp[u][0] = pre;
    dep[u] = d;
    for (int j = 1; j < maxb; j++) {
        if (~dp[i][j - 1]) {
            dp[i][j] = dp[dp[i][j - 1]][j - 1];
        }
    }
    for (int i = 0; i < (int)e[u].size(); i++) {
```

```
        if (e[u][i] != pre) {
            DFS(e[u][i], d + 1, u);
        }
    }
}

int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    for (int j = maxb - 1; ~j; j--) {
        if (dep[dp[u][j]] >= dep[v]) {
            u = dp[u][j];
        }
    }
    if (u == v) return u;
    for (int j = maxb - 1; ~j; j--) {
        if (dp[u][j] != dp[v][j]) {
            u = dp[u][j], v = dp[v][j];
        }
    }
    return dp[u][0];
}
```

## *Tarjan LCA $O(n+q)$(离线)*

```
const int maxn = 1e5 + 5;

struct query {
    int v, i;
    query(int a = 0, int b = 0) : v(a), i(b) {}
};

int fa[maxn];
int ans[maxn], color[maxn];
vector<int> e[maxn];
vector<query> Q[maxn];

void init() {
    for (int i = 0; i < maxn; i++) {
        fa[i] = i, ans[i] = -1, color[i] = 0;
        e[i].clear(), Q[i].clear();
    }
}

int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

bool unite(int u, int v) {
    u = find(u), v = find(v);
    if (u != v) {
        fa[v] = u;
        return true;
    }
```

```
    return false;
}

void addedge(int u, int v) {
    e[u].push_back(v);
    e[v].push_back(u);
}

void addquery(int u, int v, int i) {
    Q[u].push_back(query(v, i));
    Q[v].push_back(query(u, i));
}

void DFS(int u, int pre) {
    for (int i = 0; i < (int)e[u].size(); i++) {
        if (!color[e[u][i]] && e[u][i] != pre) {
            DFS(e[u][i], u);
        }
    }
    for (int i = 0; i < (int)Q[u].size(); i++) {
        if (color[Q[u][i].v]) {
            ans[Q[u][i].i] = find(Q[u][i].v);
        }
    }
    color[u] = 1;
    if (pre != -1) unite(pre, u);
}

void solve() { DFS(1, -1); }
```

## 拓扑排序$O(n+m)$

```
const int maxn = 1e5 + 5;
vector<int> e[maxn];
int seq[maxn], vis[maxn], tot, circle;

void init(int n) {
    tot = 0; cirlce = 0;
    for (int i = 0; i <= n; i++) {
        e[i].clear(), vis[i] = 0;
    }
}

void addedge(int u, int v) {
    e[u].push_back(v);
}

void DFS(int u) {
    vis[u] = 1;
    for (int i = 0; i < (int)e[u].size(); i++) {
        int& v = e[u][i];
        if (vis[v] == 1) circle = true;
```

```
        else if (vis[v] == 0) DFS(v);
    }
    vis[u] = 2;
    seq[tot++] = u;
}

void solve(int n) {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            DFS(i);
        }
    }
    reverse(seq, seq + tot);
}
```

---

# String

## *AC_automaton*

```
struct Aho_Corasick {
    static const int maxn = 5e5 + 5000;
    static const int sigma = 26;

    int tot, son[maxn][sigma], cnt[maxn], fail[maxn];

    inline void init() {
        tot = cnt[0] = fail[0] = 0;
        memset(son[0], 0, sizeof son[0]);
    }

    inline int trans(int x) {
        return x - 'a';
    }

    void insert(char *s) {
        int now = 0, n = strlen(s);
        for (int i = 0; i < n; i++) {
            int c = trans(s[i]);
            if (!son[now][c]) {
                cnt[++tot] = 0; fail[tot] = 0;
                memset(son[tot], 0, sizeof son[tot]);
                son[now][c] = tot;
            }
            now = son[now][c];
        }
        cnt[now]++;
    }

    std::queue<int> Q;

    void build() {
```

```
        while (!Q.empty()) Q.pop();
        for (int i = 0; i < sigma; i++) {
            if (son[0][i]) {
                Q.push(son[0][i]);
            }
        }
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int i = 0; i < sigma; i++) {
                if (son[u][i]) {
                    fail[son[u][i]] = son[fail[u]][i];
                    Q.push(son[u][i]);
                }
                else son[u][i] = son[fail[u]][i];
            }
        }
    }

    int solve(char *s) {
        int ret = 0, n = strlen(s);
        for (int i = 0, now = 0, c; i < n; i++, now = son[now][c]) {
            c = trans(s[i]);
            for (int u = son[now][c]; u; u = fail[u]) {
                ret += cnt[u]; cnt[u] = 0;
            }
        }
        return ret;
    }
}Accepted;
```

## 回文树

```
struct Palindromic_Tree {
    static const int maxn = 2e6 + 5;
    static const int char_db = 10;

    int tot, len[maxn], fail[maxn], ch[maxn][char_db];
    long long sum[maxn];

    // even root -> 0, odd root -> 1

    inline int newnode(int val) {
        sum[tot] = 0, len[tot] = val;
        memset(ch[tot], 0, sizeof ch[tot]);
        return tot++;
    }

    void init() {
        tot = 0;
        newnode(0); newnode(-1);
        fail[0] = 1, fail[1] = 0;
```

```
    }

    int getfail(char *s, int cur, int i) {
        while (i - len[cur] - 1 < 0 || s[i] != s[i - len[cur] - 1]) {
            cur = fail[cur];
        }
        return cur;
    }

    void build(char *s, int n) {
        int cur = 1;
        for (int i = 0; i < n; i++) {
            cur = getfail(s, cur, i);
            if (!ch[cur][s[i] - '0']) {
                int nxt = newnode(len[cur] + 2);
                fail[nxt] = ch[getfail(s, fail[cur], i)][s[i] - '0'];
                ch[cur][s[i] - '0'] = nxt;
            }
            cur = ch[cur][s[i] - '0'];
        }
    }
}pt;
```

---

## *strHash*

```
struct strhash {
    vector<ull> h, p;
    strhash(int n = 0) : h(n + 5, 0), p(n + 5, 0) {}
    void init(char *s) {
        for (int i = 0; s[i]; i++) {
            if (i) h[i] = h[i - 1] * 131, p[i] = p[i - 1] * 131;
            else p[i] = 1;
            h[i] += s[i] - 'a' + 1;
        }
    }
    inline ull gethash(int l, int r) {
        ull ret = h[r];
        if (l) ret -= h[l - 1] * p[r - l + 1];
        return ret;
    }
};
```

---

## *后缀数组*

```
namespace SuffixArray {
```

```
        const int maxn = "edit";

        int wa[maxn], wb[maxn], c[maxn], d[maxn];

        inline bool cmp(int *r, int a, int b, int k) {
            return (r[a] == r[b]) && (r[a + k] == r[b + k]);
        }

        void da(int *r, int *sa, int n, int m) {
            int i, j, p, *x = wa, *y = wb, *t;

            for (i = 0; i < m; i++) d[i] = 0;
            for (i = 0; i < n; i++) d[x[i] = r[i]]++;

            for (i = 1; i < m; i++) d[i] += d[i - 1];
            for (i = n - 1; i >= 0; i--) sa[--d[x[i]]] = i;

            for (j = 1, p = 1; j <= n; j <<= 1, m = p) {
                for (p = 0, i = n - j; i < n; i++) y[p++] = i;
                for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;

                for (i = 0; i < n; i++) c[i] = x[y[i]];
                for (i = 0; i < m; i++) d[i] = 0;

                for (i = 0; i < n; i++) d[c[i]]++;
                for (i = 1; i < m; i++) d[i] += d[i - 1];

                for (i = n - 1; i >= 0; i--) sa[--d[c[i]]] = y[i];
                for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++) {
                    x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? (p - 1) : (p++);
                }
            }
        }
        int rank[maxn], height[maxn];
        void calheight(int *r, int *sa, int n) {
            int i, j, k = 0;
            for (i = 1; i <= n; i++) rank[sa[i]] = i;
            for (i = 0; i < n; i++) {
                if (k) --k;
                for (j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
                // blank
                height[rank[i]] = k;
            }
        }
    }
```

## *Suffix Array*

- usage:

```
// s原数组, sa, n数组长度, m值域[0, m)
SuffixArray::da(s, sa, n, m);
// SuffixArray::rank, SuffixArray::height
calheight(r, sa, n);
```

- sa数组: $sa_i$为$Suffix_i$在字符串中第一次出现的位置
- height数组: $height_i = LongestPrefix(suffix(sa_{i-1}), suffix(sa_i))$
- $suffix_j$和$suffix_k$的最长公共前缀为$min(height_{rank_j+1}, height_{rank_j+2}, \ldots, height_{rank_k})$
- 求字符串中某两个后缀的最长公共前缀: 可转化为求height中某区间的最小值, 即RMQ问题
- 可重叠最长重复子串: $ans = \max height_i$
- 不可重叠最长重复子串: 二分答案, 扫height数组, 将连续的满足$height_i \geq k$的后缀分为一组, 若同一组中存在$sa_{max} - sa_{min} \geq k$, 说明存在一组不重叠的重复子串
- 可重叠的k次最长重复子串: 二分答案, 扫height数组, 分组, 判断是否存在个数不小于k的组

---

## *KMP*

```
int n, m;
char s[1005], t[1005];
int fail[1005];

void getfail() {
    int i, j;
    n = strlen(s), m = strlen(t);
    for (i = 0, j = fail[0] = -1; i < m; i++) {
        while (j >= 0 && t[j] != t[i]) j = fail[j];
        fail[i + 1] = j + 1;
    }
}

int solve() {
    getfail();
    int i, j, res;
    for (i = j = res = 0; i < n; ) {
        while (j >= 0 && s[i] != t[j]) {
            j = fail[j];
        }
        ++i, ++j;
        if (j >= m) {
            ++res, j = 0;
        }
    }
    return res;
}
```

---

## *exKMP*

```
    // nxt[i]: t[i...m-1]与t[0...m-1]的最长公共前缀
    // extend[i]: s[i...n-1]与t[0...m-1]的最长公共前缀

    const int maxn = 1e6 + 5;
    int nxt[maxn], extend[maxn];
    void exkmp(char *s, int n, char *t, int m) {
        int j = 0, k = 1;
        for (; j + 1 < m && t[j] == t[j + 1]; ++j);
        nxt[0] = m, nxt[1] = j;

        for (int i = 2; i < m; i++) {
            int p = nxt[k] + k - 1, L = nxt[i - k];
            if (p + 1 - i - L > 0) {
                nxt[i] = L;
            }
            else {
                for (j = (p - i + 1 > 0) ? (p - i + 1) : 0; i + j < m && t[i + j] == t[j]; ++j);
                // blank
                nxt[i] = j, k = i;
            }
        }

        j = k = 0;
        for (; j < n && j < m && t[j] == s[j]; ++j);
        extend[0] = j;

        for (int i = 1; i < n; i++) {
            int p = extend[k] + k - 1, L = nxt[i - k];
            if (p + 1 - i - L > 0) {
                extend[i] = L;
            }
            else {
                for (j = (p - i + 1 > 0) ? (p - i + 1) : 0; i + j < n && j < m && s[i + j] ==
t[j]; ++j);
                // blank
                extend[i] = j, k = i;
            }
        }
    }
```

---

## *manacher*

```
    const int maxn = 110005;

    int p[maxn << 1];
    char str[maxn << 1];

    int manacher(char *s, int n) {
```

```
        str[0] = '$'; str[1] = '#';

        for (int i = 0; i < n; i++) {
            str[(i << 1) + 2] = s[i];
            str[(i << 1) + 3] = '#';
        }
        n = (n + 1) << 1;
        str[n] = 0;

        int ret = 0, mx = 0, pos;
        for (int i = 1; i < n; i++) {
            p[i] = mx > i ? min(p[(pos << 1) - i], mx - i) : 1;

            while (str[i - p[i]] == str[i + p[i]]) p[i]++;

            if (p[i] + i > mx) mx = p[i] + i, pos = i;

            ret = max(ret, p[i]);
        }
        return ret - 1;
    }

    // ************************************************

    // index start from one
    int solve(char *s, int n) {
        int mx = 1; s[0] = '$'; s[++n] = 0;
        for (int i = 0, p, q; i < n; i++) {
            for (q = i; s[i + 1] == s[i]; ++i);
            for (p = i; s[q - 1] == s[p + 1]; --q, ++p);
            mx = max(mx, p - q + 1);
        }
        return mx;
    }
```

---

# 数论

## *九余数定理*

- 一个数各位数字之和等于这个数对9取模所得的数
- 每次将指数进行一次log(N)级别的变换
- 矩阵快速幂: 在O(log(N))级别的时间求第n个斐波納契数列f(n)=a*f(n-1)*+*b*f(n-2)
- 快速乘: 利用二进制实现ab(mod p), 防止溢出

## *母函数(组合数学)*

- 详见模板和实例

# 五边形数定理

- 五边形数: 1, 5, 12, 22, ……
- 第(n-1)个三角数+n^2为第n个五边形数
- $S_n = S_{n-1} + 3n - 2$

# zeckendorf定理

- 任何正整数可以表示为若干个不连续的Fibonacci数之和(斐波納契博弈)

# 错排公式

- $d(n) = (n-1)(d[n-2] + d[n-1])$

# 不定方程

- 二元一次不定方程ax+by=c有解的充要条件是(a,b)|c

# 欧拉定理

- 欧拉函数: φ(n)是小于等于n的正整数中与n互质的数的数目
- 若n, a为正整数且n与a互质, 则a^φ(n)≡1(mod n)
- 费马小定理(Fermat小定理): 对任意a和任意质数p: a^p≡a(mod p), 若a不能被p整除, a^(p-1)≡1(mod p)
- 欧拉降幂: $x^y \mod p = x^{y \mod \phi(p) + \phi(p)}, y \geq \phi(p)$

```
# (x ^ y) % p
def calc(x, y, p):
    if y < p:
        return qpow(x, y, p)
    else:
        # (x ^ y) % p = (x ^ (y % ϕ(p) + ϕ(p))) % p
        return qpow(x, y % phi[p] + phi[p], p)
```

# 费马大定理 && 费马方程

- $x^n + y^n = z^n$, 由费马大定理可知: 若$n \geq 2$且$n$为整数, 则不存在整数解$(x, y, z)(xyz \neq 0)$

# SG(Sprague-Grundy)函数

- 对于任意状态x, 它的SG函数值g(x)=mex{g(y)|y是x的后续状态}, mex是一个对于非负整数集合S的运算, mex(S)为S中没有出现的最小非负整数
- 终止状态的SG函数值为0
- 博弈打表(待更新)

# pick定理(实际上属于计算几何)

- 给定正方形格子点的简单多边形, i为其内部格点数目, b为其边上格点数, 则其面积$A = i + b/2 - 1$

# 逆元(费马小定理)

- 当p为素数时， $a/b \mod p = ab_1 \mod p, b_1 = b^{p-2} \mod p$

# 威尔逊定理

- 当且仅当p为素数时: $(p-1)! \equiv -1( \mod p)$

# 杨辉三角(应用于排列组合)

- 第n行的元素个数有n个
- 第n行的元素之和为2^(n-1)
- 第n行第m个数的值为C(n-1,m-1),C为组合数
- (a+b)^n展开后的各项系数等于第n+1行的值
- 第n行第m个数的奇偶判断(m-1)&(n-1)==(m-1)?odd:even

# 第一类斯特林数

- $S(p,k)$表示把p个人分成k组作环排列的方案数
- $S(p,k) = (p-1) * S(p-1,k) + S(p-1,k-1), 1 \le k \le p-1$
- $S(p,0) = 0, p \ge 1$
- $S(p,p) = 1, p \ge 0$
- 使第p个人单独构成一个环排列，前p-1人构成k-1个环排列，方案数$S(p-1,k-1)$。
- 使前p-1个人构成k个环排列，第p个人插入到第i人左边，方案数$S(p-1,k)$

# 第二类斯特林数

- 将p个物体分成k个非空的不可辨别的集合的方案数
- $S(p,k)=k*S(p-1,k)+S(p-1,k-1),1\leq k\leq p-1$
- $S(p,0)=0,p\geq 1$
- $S(p,p)=1,p\geq 0$
- 考虑第p个物品，p可以单独构成一个非空集合，此时前p-1个物品构成k-1个非空的不可辨别的集合，方法数为$S(p-1,k-1)$
- 也可以前p-1种物品构成k个非空的不可辨别的集合，第p个物品放入任意一个中，这样有$kS(p-1,k)$种方法。

# Lucas定理(大组合数取模)

- $C(n,m)\mod p=C(n/p,m/p)C(n\mod p,m\mod p)\mod p$,其中p为质数

```
// C(n, m) % p = C(n % p, m % p) * C(n / p, m / p) % p
// p is a prime

typedef long long ll;
const int MOD = 1e9 + 7;

// calc C(n, m) % MOD
inline ll C(ll n, ll m);

ll lucas(ll n, ll m) {
    if (n < MOD && m < MOD) {
        return C(n, m);
    }
    return C(n % MOD, m % MOD) * lucas(n / MOD, m / MOD) % MOD;
}
```

# 欧拉&素数线性筛

```
struct Seive {
    int maxn;
    vector<int>phi;
    Seive(int n) : maxn(n + 5), phi(n + 5) {
        phi[0] = 0;
        phi[1] = 1;
        for(int i = 2; i < maxn; i++) {
            if(!phi[i]) {
                for(int j = i; j < maxn; j += i) {
                    if(!phi[j]) {
                        phi[j] = j;
                    }
                    phi[j] -= phi[j] / i;
                }
            }
```

```
            }
        }
        inline bool chkpri(int x) { return phi[x] == x - 1; }
        inline int getphi(int x) { return phi[x]; }
    };
```

## 组合数打表

```
    const int MOD = 1e9 + 7;
    long long C[1005][1005];
    void init() {
        for (int i = C[0][0] = 1; i < 1005; i++) {
            for (int j = C[i][0] = 1; j <= i; j++) {
                C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % MOD;
            }
        }
    }
```

## Simpson求积分

```
    double simpson(const double& a, const double& b)
    {
        double c = (a + b) / 2;
        return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
    }

    double asr(double a, double b, double eps, double A)
    {
        double c = (a + b) / 2;
        double L = simpson(a, c), R = simpson(c, b);
        if (fabs(L + R - A) <= 15 * eps)
            return L + R + (L + R - A) / 15.0;
        return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
    }
```

## Meissel-Lehmer(求[1, n]之间的素数个数)$O(n^{2/3})$

```
namespace pcf{
        typedef long long ll;
        const int N = 5e6 + 2;
        bool np[N];
        int prime[N], pi[N];

        int getprime() {
```

```
    int cnt = 0;
    np[0] = np[1] = 1;
    pi[0] = pi[1] = 0;
    for (int i = 2; i < N; i++) {
        if (!np[i]) prime[++cnt] = i;
        pi[i] = cnt;
        for (int j = 1; j <= cnt && i * prime[j] < N; ++j) {
            np[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    return cnt;
}

const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;

int phi[PM + 1][M + 1], sz[M + 1];

void init() {
    getprime();
    sz[0] = 1;
    for (int i = 0; i <= PM; i++) {
        phi[i][0] = i;
    }
    for (int i = 1; i <= M; i++) {
        sz[i] = prime[i] * sz[i - 1];
        for (int j = 1; j <= PM; j++) {
            phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];
        }
    }
}

int sqrt2(ll x) {
    ll r = ll(sqrt(x - 0.1));
    while (r * r <= x) ++r;
    return int(r - 1);
}

int sqrt3(ll x) {
    ll r = ll(cbrt(x - 0.1));
    while (r * r * r <= x) ++r;
    return int(r - 1);
}

ll getphi(ll x, int s) {
    if (s == 0) {
        return x;
    }
    if (s <= M) {
        return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    }
    if (x <= prime[s] * prime[s]) {
        return pi[x] - s + 1;
    }
    if (x <= prime[s] * prime[s] * prime[s] && x < N) {
        int s2x = pi[sqrt2(x)];
```

```
            ll ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
            for (int i = s + 1; i <= s2x; i++) {
                ans += pi[x / prime[i]];
            }
            return ans;
        }
        return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
    }

    ll getpi(ll x) {
        if (x < N) {
            return pi[x];
        }
        ll ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
        for (int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) {
            ans -= getpi(x / prime[i]) - i + 1;
        }
        return ans;
    }

    ll lehmer(ll x) {
        if (x < N) {
            return pi[x];
        }
        int a = int(lehmer(sqrt2(sqrt2(x))));
        int b = int(lehmer(sqrt2(x)));
        int c = int(lehmer(sqrt3(x)));
        ll sum = getphi(x, a) + ll(b + a - 2) * (b - a + 1) / 2;
        for (int i = a + 1; i <= b; i++) {
            ll w = x / prime[i];
            sum -= lehmer(w);
            if (i > c) {
                continue;
            }
            ll lim = lehmer(sqrt2(w));
            for (int j = i; j <= lim; j++) {
                sum -= lehmer(w / prime[j]) - (j - 1);
            }
        }
        return sum;
    }
}
```

## 扩展欧几里得与中国剩余定理

```
pll exgcd(const ll x, const ll y)
{
    if(!y)
    {
        return make_pair(1, 0);
    }
    pll cur = exgcd(y, x % y);
```

```
    return make_pair(cur.second, cur.first - (x / y) * cur.second);
}
pll crt(const vector<pll> & v)
{
    //v里每个pll中first为被模数，second为模数
    ll a = 1, r = 0;
    const int len = v.size();
    for(int i = 0; i < len; i++) {
        pll cur = exgcd(a, v[i].first);
        ll gcd = a * cur.first + v[i].first * cur.second;
        if((v[i].second - r) % gcd != 0){
            return make_pair(-1, -1);
        }
        const ll p = v[i].first / gcd;
        r += mod(cur.first * ((v[i].second - r) / gcd), p) * a;
        a *= p;
    }
    return make_pair(a, r);
}
```

## 大随机数生成与素性测试

```
ull randull()
{
    static random_device rd;
    static mt19937_64 eng(rd());
    static uniform_int_distribution<ull>distr;
    return distr(eng);
}
ull randint(ull const& min = 0, ull const& max = 0)
{
    return double(randull()) / ULLONG_MAX * (max - min + 1) + min;
}
bool is_prime(ll n)
{
    if (n == 2) return true;
    if (n < 2 || (~n & 1)) return false;
    ll m = n - 1, k = 0;
    while (!(m & 1))
    {
        k++;
        m >>= 1;
    }
    for (int i = 1; i <= 30; i++)
    {
        ll a = randint((ull)1, (ull)(n - 1));
        ll x = fpow(a, m, n);
        ll y;
        for (int j = 1; j <= k; j++)
        {
            y = fmul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1)
```

```
            {
                return false;
            }
            x = y;
        }
        if (y != 1)
        {
            return false;
        }
    }
    return true;
}
```

## 蔡勒公式

```
int zeller(int y, int m, int d)
{
    //星期日为0
    if (m == 1 || m == 2)
    {
        m += 12;
        y--;
    }
    int c = y / 100;
    y = y % 100;
    int w = y + y / 4 + c / 4 - 2 * c + (26 * (m + 1)) / 10 + d - 1;
    w = ((w % 7) + 7) % 7;
    return w;
}
```

## 后缀表达式

```
const int MXLEN = 1000 + 5;
int fst[MXLEN];
char str[MXLEN];

typedef double CSS;

CSS jud(int begin, int end)
{
    int i;
    CSS k;
    //越往后优先级越高
    for (i = begin; i <= end; i++)
    {
        if (str[i] == '+' && fst[i] == fst[begin])
        {
            k = jud(begin, i - 1) + jud(i + 1, end);
```

```
                return k;
            }
        }
        for (i = end; i >= begin; i--)
        {
            if (str[i] == '-' && fst[i] == fst[begin])
            {
                k = jud(begin, i - 1) - jud(i + 1, end);
                return k;
            }
        }
        if (str[begin] == '(')
        {
            for (i = begin + 1; fst[i] >= fst[begin + 1]; i++);

            k = jud(begin + 1, i - 1);
        }
        else
        {
            char *p = str;
            sscanf(p + begin, "%lf", &k);
        }
        return k;
    }

    CSS solve()
    {
        const int len = strlen(str);
        for (int i = 1; i <= len - 1; i++)
        {
            if (str[i - 1] == '(')
            {
                fst[i] = fst[i - 1] + 1;
            }
            else
            {
                if (str[i] == ')')
                {
                    fst[i] = fst[i - 1] - 1;
                }
                else
                {
                    fst[i] = fst[i - 1];
                }
            }
        }
        return jud(0, len);
    }
```

## *java大数牛顿迭代法开方*

```
    public static BigInteger sqrt(BigInteger n) {
```

```
        String a = n.toString();
        final int len = a.length();
        if((~len & 1) == 1) {
            a = a.substring(0, len / 2 + 1);
        } else {
            a = a.substring(0, len / 2);
        }
        BigInteger x = new BigInteger(a);
        final BigInteger two = BigInteger.valueOf(2);
        if(a == "0" || a == "1") {
            return n;
        } else {
            while(n.compareTo(x.multiply(x)) < 0) {
                x = (x.add(n.divide(x))).divide(two);
            }
            return x;
        }
    }
```

# Convex凸包

```
const int MAX = 10005;
const double eps = 1e-8;

/*
    O(nlogn)
    p->原点集，s->凸包
    Convex::Graham(n); // 返回值为凸包中点数
*/

namespace Convex {
    struct Point {
        double x, y;
        Point(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
    }p[MAX], s[MAX];
    template <typename T> inline T sqr(T x) { return x * x; }
    inline int dcmp(double x) {
        if (fabs(x) <= eps) return 0;
        return x < 0 ? -1 : 1;
    }
    inline double dist(Point a, Point b) {
        return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
    }
    inline double cross(Point a, Point b, Point c) {
        return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
    }
    bool cmp(const Point& a, const Point& b) {
        int c = dcmp(cross(p[0], a, b));
        if (c < 0) return false;
        if (c > 0) return true;
        return dist(p[0], a) < dist(p[0], b);
    }
    int Graham(int n) {
```

```
        for (int i = 0; i < n; i++) {
            if (dcmp(p[i].y - p[0].y) < 0) {
                swap(p[i], p[0]);
            }
            else if (!dcmp(p[i].y - p[0].y) && dcmp(p[i].x - p[0].x) < 0) {
                swap(p[i], p[0]);
            }
        }
        sort(p + 1, p + n, cmp);
        s[0] = p[0], s[1] = p[1];
        int tot = 1;
        for (int i = 2; i < n; i++) {
            while (tot > 0 && dcmp(cross(s[tot - 1], s[tot], p[i])) <= 0) {
                --tot;
            }
            s[++tot] = p[i];
        }
        return tot + 1;
    }
}
```