


常用代码模板1——基础算法

作者:  yxc (/user/myspace/index/1/), 2019-07-31 21:18:31, 阅读 78589

▲ 算法基础课相关代码模板

681 • 活动链接 —— 算法基础课 (<https://www.acwing.com/activity/content/11/>)

▼ 快速排序算法模板 —— 模板题 AcWing 785. 快速排序 (<https://www.acwing.com/problem/content/787/>)

★
839

```
void quick_sort(int q[], int l, int r)
{
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = q[l + r >> 1];
    while (i < j)
    {
        do i ++ ; while (q[i] < x);
        do j -- ; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j), quick_sort(q, j + 1, r);
}
```

归并排序算法模板 —— 模板题 AcWing 787. 归并排序 (<https://www.acwing.com/problem/content/789/>)

```
void merge_sort(int q[], int l, int r)
{
    if (l >= r) return;

    int mid = l + r >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);

    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (q[i] <= q[j]) tmp[k ++ ] = q[i ++ ];
        else tmp[k ++ ] = q[j ++ ];

    while (i <= mid) tmp[k ++ ] = q[i ++ ];
    while (j <= r) tmp[k ++ ] = q[j ++ ];

    for (i = l, j = 0; i <= r; i ++, j ++ ) q[i] = tmp[j];
}
```

整数二分算法模板 —— 模板题 AcWing 789. 数的范围 (<https://www.acwing.com/problem/content/791/>)

```
bool check(int x) { /* ... */ } // 检查x是否满足某种性质

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check()判断mid是否满足性质
        else l = mid + 1;
    }
    return l;
}

// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

浮点数二分算法模板 —— 模板题 AcWing 790. 数的三次方根 (<https://www.acwing.com/problem/content/792/>)

```
bool check(double x) { /* ... */ } // 检查x是否满足某种性质

double bsearch_3(double l, double r)
{
    const double eps = 1e-6;    // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

高精度加法 —— 模板题 AcWing 791. 高精度加法 (<https://www.acwing.com/problem/content/793/>)

```
// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &A, vector<int> &B)
{
    if (A.size() < B.size()) return add(B, A);

    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++)
    {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }

    if (t) C.push_back(t);
    return C;
}
```

高精度减法 —— 模板题 AcWing 792. 高精度减法 (<https://www.acwing.com/problem/content/794/>)

```
// C = A - B, 满足A >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); i ++ )
    {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度乘低精度 —— 模板题 AcWing 793. 高精度乘法 (<https://www.acwing.com/problem/content/795/>)

```
// C = A * b, A >= 0, b > 0
vector<int> mul(vector<int> &A, int b)
{
    vector<int> C;

    int t = 0;
    for (int i = 0; i < A.size() || t; i ++ )
    {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}
```

高精度除以低精度 —— 模板题 AcWing 794. 高精度除法 (<https://www.acwing.com/problem/content/796/>)

```
// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r)
{
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i -- )
    {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

一维前缀和 —— 模板题 AcWing 795. 前缀和 (<https://www.acwing.com/problem/content/797/>)

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
```

二维前缀和 —— 模板题 AcWing 796. 子矩阵的和 (<https://www.acwing.com/problem/content/798/>)

$S[i, j]$ = 第*i*行*j*列格子左上部分所有元素的和
以(x_1, y_1)为左上角, (x_2, y_2)为右下角的子矩阵的和为:
 $S[x_2, y_2] - S[x_1 - 1, y_2] - S[x_2, y_1 - 1] + S[x_1 - 1, y_1 - 1]$

一维差分 —— 模板题 AcWing 797. 差分 (<https://www.acwing.com/problem/content/799/>)

给区间 $[l, r]$ 中的每个数加上 c : $B[l] += c, B[r + 1] -= c$

二维差分 —— 模板题 AcWing 798. 差分矩阵 (<https://www.acwing.com/problem/content/800/>)

给以(x_1, y_1)为左上角, (x_2, y_2)为右下角的子矩阵中的所有元素加上 c :
 $S[x_1, y_1] += c, S[x_2 + 1, y_1] -= c, S[x_1, y_2 + 1] -= c, S[x_2 + 1, y_2 + 1] += c$

位运算 —— 模板题 AcWing 801. 二进制中1的个数 (<https://www.acwing.com/problem/content/803/>)

求*n*的第*k*位数字: $n \gg k \& 1$
返回*n*的最后一位1: $\text{lowbit}(n) = n \& -n$

双指针算法 —— 模板题 AcWing 799. 最长连续不重复子序列 (<https://www.acwing.com/problem/content/801/>), AcWing 800. 数组元素的目标和 (<https://www.acwing.com/problem/content/802/>)

```
for (int i = 0, j = 0; i < n; i ++ )
{
    while (j < i && check(i, j)) j ++ ;

    // 具体问题的逻辑
}
```

常见问题分类：

- (1) 对于一个序列，用两个指针维护一段区间
- (2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作

离散化 —— 模板题 AcWing 802. 区间和 (<https://www.acwing.com/problem/content/804/>)

```
vector<int> alls; // 存储所有待离散化的值
sort(alls.begin(), alls.end()); // 将所有值排序
alls.erase(unique(alls.begin(), alls.end()), alls.end()); // 去掉重复元素

// 二分求出x对应的离散化的值
int find(int x) // 找到第一个大于等于x的位置
{
    int l = 0, r = alls.size() - 1;
    while (l < r)
    {
        int mid = l + r >> 1;
        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1; // 映射到1, 2, ...n
}
```

区间合并 —— 模板题 AcWing 803. 区间合并 (<https://www.acwing.com/problem/content/805/>)

```
// 将所有存在交集的区间合并
void merge(vector<PII> &segs)
{
    vector<PII> res;

    sort(segs.begin(), segs.end());

    int st = -2e9, ed = -2e9;
    for (auto seg : segs)
        if (ed < seg.first)
        {
            if (st != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);

    if (st != -2e9) res.push_back({st, ed});

    segs = res;
}
```

125 评论

在这里写评论... (支持Markdown和Latex语法)

提交评论



(/user
/myspace
/index
/82142/)

Waken (/user/myspace/index/82142/) 20小时前 回复

快排quick_sort(q, l, j), quick_sort(q, j + 1, r);
可以改为quick_sort(q, l, j - 1), quick_sort(q, j + 1, r);吗



(/user
/myspace
/index
/82142/)

Waken (/user/myspace/index/82142/) 20小时前 回复

快速排序能从 $i = l, j = r$ 开始吗？循环部分不先自增




(/user
/myspace)

小强d (/user/myspace/index/80510/) 18天前 回复

我是刷吐了，一道题4h，脑瓜子嗡嗡嗡

常用代码模板2——数据结构

作者:  yxc (/user/myspace/index/1/), 2019-07-31 21:34:51, 阅读 43642

▲ 算法基础课相关代码模板

250 • 活动链接 —— 算法基础课 (<https://www.acwing.com/activity/content/11/>)

▼ 单链表 —— 模板题 AcWing 826. 单链表 (<https://www.acwing.com/problem/content/828/>)

★
404

```
// head存储链表头，e[]存储节点的值，ne[]存储节点的next指针，idx表示当前用到了哪个节点
int head, e[N], ne[N], idx;
```

```
// 初始化
void init()
{
```

```
    head = -1;
    idx = 0;
}
```

```
// 在链表头插入一个数a
```

```
void insert(int a)
{
    e[idx] = a, ne[idx] = head, head = idx ++ ;
}
```

```
// 将头结点删除，需要保证头结点存在
```

```
void remove()
{
    head = ne[head];
}
```

双链表 —— 模板题 AcWing 827. 双链表 (<https://www.acwing.com/problem/content/829/>)


```
// e[]表示节点的值，l[]表示节点的左指针，r[]表示节点的右指针，idx表示当前用到了哪个
int e[N], l[N], r[N], idx;

// 初始化
void init()
{
    //0是左端点，1是右端点
    r[0] = 1, l[1] = 0;
    idx = 2;
}

// 在节点a的右边插入一个数x
void insert(int a, int x)
{
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx ++ ;
}

// 删除节点a
void remove(int a)
{
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
```

栈 —— 模板题 AcWing 828. 模拟栈 (<https://www.acwing.com/problem/content/830/>)

```
// tt表示栈顶
int stk[N], tt = 0;

// 向栈顶插入一个数
stk[ ++ tt] = x;

// 从栈顶弹出一个数
tt -- ;

// 栈顶的值
stk[tt];

// 判断栈是否为空
if (tt > 0)
{
}
}
```

队列 —— 模板题 AcWing 829. 模拟队列 (<https://www.acwing.com/problem/content/831/>)

1. 普通队列:

```
// hh 表示队头, tt表示队尾
int q[N], hh = 0, tt = -1;

// 向队尾插入一个数
q[ ++ tt] = x;

// 从队头弹出一个数
hh ++ ;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh <= tt)
{
}
```

2. 循环队列

```
// hh 表示队头, tt表示队尾的后一个位置
int q[N], hh = 0, tt = 0;

// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;

// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh != tt)
{
}
```

单调栈 —— 模板题 AcWing 830. 单调栈 (<https://www.acwing.com/problem/content/832/>)

常见模型：找出每个数左边离它最近的比它大/小的数

```
int tt = 0;
for (int i = 1; i <= n; i ++ )
{
    while (tt && check(stk[tt], i)) tt -- ;
    stk[ ++ tt] = i;
}
```

单调队列 —— 模板题 AcWing 154. 滑动窗口 (<https://www.acwing.com/problem/content/156/>)

常见模型：找出滑动窗口中的最大值/最小值

```
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ )
{
    while (hh <= tt && check_out(q[hh])) hh ++ ; // 判断队头是否滑出窗口
    while (hh <= tt && check(q[tt], i)) tt -- ;
    q[ ++ tt] = i;
}
```

KMP —— 模板题 AcWing 831. KMP字符串 (<https://www.acwing.com/problem/content/833/>)

// s[]是长文本，p[]是模式串，n是s的长度，m是p的长度

求模式串的Next数组：

```
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;
}
```

// 匹配

```
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)
    {
        j = ne[j];
        // 匹配成功后的逻辑
    }
}
```

Trie树 —— 模板题 AcWing 835. Trie字符串统计 (<https://www.acwing.com/problem/content/837/>)

```
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];
}
```

并查集 —— 模板题 AcWing 836. 合并集合 (<https://www.acwing.com/problem/content/838/>), **AcWing 837. 连通块中点的数量** (<https://www.acwing.com/problem/content/839/>)

(1)朴素并查集:

```
int p[N]; //存储每个点的祖宗节点

// 返回x的祖宗节点
int find(int x)
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

// 初始化, 假定节点编号是1~n
for (int i = 1; i <= n; i ++ ) p[i] = i;

// 合并a和b所在的两个集合:
p[find(a)] = find(b);
```

(2)维护size的并查集:

```
int p[N], size[N];
//p[]存储每个点的祖宗节点, size[]只有祖宗节点的有意义, 表示祖宗节点所在集合中的

// 返回x的祖宗节点
int find(int x)
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

// 初始化, 假定节点编号是1~n
for (int i = 1; i <= n; i ++ )
{
    p[i] = i;
    size[i] = 1;
}

// 合并a和b所在的两个集合:
size[find(b)] += size[find(a)];
p[find(a)] = find(b);
```

(3)维护到祖宗节点距离的并查集:

```
int p[N], d[N];
//p[]存储每个点的祖宗节点, d[x]存储x到p[x]的距离

// 返回x的祖宗节点
int find(int x)
```

```
{
    if (p[x] != x)
    {
        int u = find(p[x]);
        d[x] += d[p[x]];
        p[x] = u;
    }
    return p[x];
}

// 初始化, 假定节点编号是1~n
for (int i = 1; i <= n; i ++ )
{
    p[i] = i;
    d[i] = 0;
}

// 合并a和b所在的两个集合:
p[find(a)] = find(b);
d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量
```

堆 —— 模板题 AcWing 838. 堆排序 (<https://www.acwing.com/problem/content/840/>), AcWing 839. 模拟堆 (<https://www.acwing.com/problem/content/841/>)

```
// h[N]存储堆中的值, h[1]是堆顶, x的左儿子是2x, 右儿子是2x + 1
// ph[k]存储第k个插入的点在堆中的位置
// hp[k]存储堆中下标是k的点是第几个插入的
int h[N], ph[N], hp[N], size;

// 交换两个点, 及其映射关系
void heap_swap(int a, int b)
{
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}

void down(int u)
{
    int t = u;
    if (u * 2 <= size && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= size && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u != t)
    {
        heap_swap(u, t);
        down(t);
    }
}

void up(int u)
{
    while (u / 2 && h[u] < h[u / 2])
    {
        heap_swap(u, u / 2);
        u >>= 1;
    }
}

// O(n)建堆
for (int i = n / 2; i; i -- ) down(i);
```

一般哈希 —— 模板题 AcWing 840. 模拟散列表 (<https://www.acwing.com/problem/content/842/>)

(1) 拉链法

```
int h[N], e[N], ne[N], idx;

// 向哈希表中插入一个数
void insert(int x)
{
    int k = (x % N + N) % N;
    e[idx] = x;
    ne[idx] = h[k];
    h[k] = idx ++ ;
}

// 在哈希表中查询某个数是否存在
bool find(int x)
{
    int k = (x % N + N) % N;
    for (int i = h[k]; i != -1; i = ne[i])
        if (e[i] == x)
            return true;

    return false;
}
```

(2) 开放寻址法

```
int h[N];

// 如果x在哈希表中，返回x的下标；如果x不在哈希表中，返回x应该插入的位置
int find(int x)
{
    int t = (x % N + N) % N;
    while (h[t] != null && h[t] != x)
    {
        t ++ ;
        if (t == N) t = 0;
    }
    return t;
}
```

字符串哈希 —— 模板题 AcWing 841. 字符串哈希 (<https://www.acwing.com/problem/content/843/>)

核心思想：将字符串看成 P 进制数， P 的经验值是131或13331，取这两个值的冲突概率低

小技巧：取模的数用 2^{64} ，这样直接用unsigned long long存储，溢出的结果就是取模的结果

```
typedef unsigned long long ULL;
ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值, p[k]存储  $P^k \bmod 2^{64}$ 

// 初始化
p[0] = 1;
for (int i = 1; i <= n; i++)
{
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
}

// 计算子串 str[l ~ r] 的哈希值
ULL get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}
```

C++ STL简介

vector, 变长数组, 倍增的思想

- `size()` 返回元素个数
- `empty()` 返回是否为空
- `clear()` 清空
- `front()/back()`
- `push_back()/pop_back()`
- `begin()/end()`
- `[]`
- 支持比较运算, 按字典序

pair<int, int>

- `first`, 第一个元素
- `second`, 第二个元素
- 支持比较运算, 以`first`为第一关键字, 以`second`为第二关键字 (字典序)

string, 字符串

- `size()/length()` 返回字符串长度
- `empty()`
- `clear()`
- `substr(起始下标, (子串长度))` 返回子串
- `c_str()` 返回字符串所在字符数组的起始地址

queue, 队列

- `size()`
- `empty()`
- `push()` 向队尾插入一个元素
- `front()` 返回队头元素
- `back()` 返回队尾元素
- `pop()` 弹出队头元素

priority_queue, 优先队列, 默认是大根堆

- `size()`
- `empty()`
- `push()` 插入一个元素
- `top()` 返回堆顶元素
- `pop()` 弹出堆顶元素
- 定义成小根堆的方式: `priority_queue<int, vector<int>, greater<int>> q;`

stack, 栈

- `size()`
- `empty()`
- `push()` 向栈顶插入一个元素
- `top()` 返回栈顶元素
- `pop()` 弹出栈顶元素

deque, 双端队列

- `size()`
- `empty()`
- `clear()`

```

front()/back()
push_back()/pop_back()
push_front()/pop_front()
begin()/end()
[]

```

set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列

```

size()
empty()
clear()
begin()/end()
++, -- 返回前驱和后继，时间复杂度  $O(\log n)$ 

```

set/multiset

```

insert() 插入一个数
find() 查找一个数
count() 返回某一个数的个数
erase()
    (1) 输入是一个数x，删除所有x  $O(k + \log n)$ 
    (2) 输入一个迭代器，删除这个迭代器
lower_bound()/upper_bound()
    lower_bound(x) 返回大于等于x的最小的数的迭代器
    upper_bound(x) 返回大于x的最小的数的迭代器

```

map/multimap

```

insert() 插入的数是一个pair
erase() 输入的参数是pair或者迭代器
find()
[] 注意multimap不支持此操作。 时间复杂度是  $O(\log n)$ 
lower_bound()/upper_bound()

```

unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
和上面类似，增删改查的时间复杂度是 $O(1)$

不支持 lower_bound()/upper_bound(), 迭代器的++, --

bitset, 压位

```

bitset<10000> s;
~, &, |, ^
>>, <<
==, !=
[]

```

count() 返回有多少个1

any() 判断是否至少有一个1

none() 判断是否全为0

set() 把所有位置成1

set(k, v) 将第k位变成v

reset() 把所有位变成0

flip() 等价于~

flip(k) 把第k位取反

74 评论

在这里写评论... (支持Markdown和Latex语法)

提交评论



(/user
/myspace
/index
/65605/)

3624 (/user/myspace/index/65605/) 15天前 回复
(3)维护到祖宗节点距离的并查集:

```
// 合并a和b所在的两个集合:  
p[find(a)] = find(b);  
d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量
```

感觉顺序应该反过来? 不然先“p[find(a)] = find(b)”的话, “d[find(a)] = distance”就变成了“d[pb] = distance”了:

```
// 合并a和b所在的两个集合:  
d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量  
p[find(a)] = find(b);
```



(/user
/myspace
/index
/3846/)

AC_渣龙 (/user/myspace/index/3846/) 1个月前 回复
Y总可以稍微介绍下, java刷题常用集合么? 类似于你写的stl这里😁😁



小A小A (/user/myspace/index/70747/) 29天前 回复
(/user +1!
/myspace
/index
/70747/)



(/user
/myspace
/index
/70532/)


夏小天 (/user/myspace/index/70532/) 2个月前 回复
p是131的时候能存下str[i]是因为可显字符少于131个吗?



(/user
/myspace
/index)

dwr (/user/myspace/index/41941/) 3个月前 回复
y总, 字符串哈希就是Rabin-Karp嘛, 至少思想上是类似的叭?

常用代码模板3——搜索与图论

作者:  yxc (/user/myspace/index/1/), 2019-07-31 21:51:16, 阅读 35212

▲ 算法基础课相关代码模板

194 • 活动链接 —— 算法基础课 (<https://www.acwing.com/activity/content/11/>)

▼ 树与图的存储



树是一种特殊的图，与图的存储方式相同。

对于无向图中的边 ab ，存储两条有向边 $a \rightarrow b$, $b \rightarrow a$ 。

408 因此我们可以只考虑有向图的存储。

(1) 邻接矩阵: $g[a][b]$ 存储边 $a \rightarrow b$

(2) 邻接表:

```
// 对于每个点k，开一个单链表，存储k所有可以走到的点。h[k]存储这个单链表的头结点
int h[N], e[N], ne[N], idx;

// 添加一条边a->b
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// 初始化
idx = 0;
memset(h, -1, sizeof h);
```

树与图的遍历

时间复杂度 $O(n + m)$, n 表示点数, m 表示边数

(1) 深度优先遍历 —— 模板题 AcWing 846. 树的重心 (<https://www.acwing.com/problem/content/848/>)

```
int dfs(int u)
{
    st[u] = true; // st[u] 表示点u已经被遍历过

    for (int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j]) dfs(j);
    }
}
```

(2) 宽度优先遍历 —— 模板题 AcWing 847. 图中点的层次 (<https://www.acwing.com/problem/content/849/>)

```
queue<int> q;
st[1] = true; // 表示1号点已经被遍历过
q.push(1);

while (q.size())
{
    int t = q.front();
    q.pop();

    for (int i = h[t]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j])
        {
            st[j] = true; // 表示点j已经被遍历过
            q.push(j);
        }
    }
}
```

拓扑排序 —— 模板题 AcWing 848. 有向图的拓扑序列 (<https://www.acwing.com/problem/content/850/>)

时间复杂度 $O(n + m)$, n 表示点数, m 表示边数

```
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;
}
```

朴素dijkstra算法 —— 模板题 AcWing 849. Dijkstra求最短路 I (<https://www.acwing.com/problem/content/851/>)

时间复杂度是 $O(n^2 + m)$, n 表示点数, m 表示边数

```
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];
}
```

堆优化版dijkstra —— 模板题 AcWing 850. Dijkstra求最短路 II
(<https://www.acwing.com/problem/content/852/>)

时间复杂度 $O(m \log n)$, n 表示点数, m 表示边数


```
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];
}
```

Bellman-Ford算法 —— 模板题 AcWing 853. 有边数限制的最短路 (<https://www.acwing.com/problem/content/855/>)

时间复杂度 $O(nm)$, n 表示点数, m 表示边数

注意在模板题中需要对下面的模板稍作修改，加上备份数组，详情见模板题。

```
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
    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];
}
```

spfa 算法 (队列优化的Bellman-Ford算法) —— 模板题 AcWing 851. spfa求最短路 (<https://www.acwing.com/problem/content/853/>)

时间复杂度 平均情况下 $O(m)$, 最坏情况下 $O(nm)$, n 表示点数, m 表示边数

```
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];
}
```

spfa判断图中是否存在负环 —— 模板题 AcWing 852. spfa判断负环
(<https://www.acwing.com/problem/content/854/>)

时间复杂度是 $O(nm)$, n 表示点数, m 表示边数

```
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], cnt[N];        // dist[x]存储1号点到x的最短距离, cnt[x]存储1到x的最
bool st[N];        // 存储每个点是否在队列中

// 如果存在负环, 则返回true, 否则返回false。
bool spfa()
{
    // 不需要初始化dist数组
    // 原理: 如果某条最短路径上有n个点(除了自己), 那么加上自己之后一共有n+1个点,

    queue<int> q;
    for (int i = 1; i <= n; i ++ )
    {
        q.push(i);
        st[i] = 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];
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n) return true;        // 如果从1号点到x的最短路中
                if (!st[j])
                {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }

    return false;
}
```

floyd算法 —— 模板题 AcWing 854. Floyd求最短路 (<https://www.acwing.com/problem/content/856/>)

时间复杂度是 $O(n^3)$, n 表示点数

初始化:

```
for (int i = 1; i <= n; i ++ )
    for (int j = 1; j <= n; j ++ )
        if (i == j) d[i][j] = 0;
        else d[i][j] = INF;
```

// 算法结束后, $d[a][b]$ 表示 a 到 b 的最短距离

```
void floyd()
{
    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]);
}
```

朴素版prim算法 —— 模板题 AcWing 858. Prim算法求最小生成树
(<https://www.acwing.com/problem/content/860/>)

时间复杂度是 $O(n^2 + m)$, n 表示点数, m 表示边数

```
int n;        // n表示点数
int g[N][N];   // 邻接矩阵, 存储所有边
int dist[N];   // 存储其他点到当前最小生成树的距离
bool st[N];    // 存储每个点是否已经在生成树中

// 如果图不连通, 则返回INF(值是0x3f3f3f3f), 否则返回最小生成树的树边权重之和
int prim()
{
    memset(dist, 0x3f, sizeof dist);

    int res = 0;
    for (int i = 0; i < n; i ++ )
    {
        int t = -1;
        for (int j = 1; j <= n; j ++ )
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;

        if (i && dist[t] == INF) return INF;

        if (i) res += dist[t];
        st[t] = true;

        for (int j = 1; j <= n; j ++ ) dist[j] = min(dist[j], g[t][j]);
    }

    return res;
}
```

Kruskal算法 —— 模板题 AcWing 859. Kruskal算法求最小生成树
(<https://www.acwing.com/problem/content/861/>)

时间复杂度是 $O(m \log m)$, n 表示点数, m 表示边数

```
int n, m;          // n是点数, m是边数
int p[N];          // 并查集的父节点数组

struct Edge        // 存储边
{
    int a, b, w;

    bool operator< (const Edge &W) const
    {
        return w < W.w;
    }
} edges[M];

int find(int x)     // 并查集核心操作
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal()
{
    sort(edges, edges + m);

    for (int i = 1; i <= n; i++) p[i] = i;    // 初始化并查集

    int res = 0, cnt = 0;
    for (int i = 0; i < m; i++)
    {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b);
        if (a != b)    // 如果两个连通块不连通, 则将这两个连通块合并
        {
            p[a] = b;
            res += w;
            cnt++;
        }
    }

    if (cnt < n - 1) return INF;
    return res;
}
```

染色法判别二分图 —— 模板题 AcWing 860. 染色法判定二分图
(<https://www.acwing.com/problem/content/862/>)

时间复杂度是 $O(n + m)$, n 表示点数, m 表示边数

```
int n;        // n表示点数
int h[N], e[M], ne[M], idx;    // 邻接表存储图
int color[N];    // 表示每个点的颜色, -1表示未染色, 0表示白色, 1表示黑色

// 参数: u表示当前节点, c表示当前点的颜色
bool dfs(int u, int c)
{
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (color[j] == -1)
        {
            if (!dfs(j, !c)) return false;
        }
        else if (color[j] == c) return false;
    }

    return true;
}

bool check()
{
    memset(color, -1, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i ++ )
        if (color[i] == -1)
            if (!dfs(i, 0))
            {
                flag = false;
                break;
            }
    return flag;
}
```

匈牙利算法 —— 模板题 AcWing 861. 二分图的最大匹配 (<https://www.acwing.com/problem/content/863/>)

时间复杂度是 $O(nm)$, n 表示点数, m 表示边数


```

int n1, n2;    // n1表示第一个集合中的点数, n2表示第二个集合中的点数
int h[N], e[M], ne[M], idx;    // 邻接表存储所有边, 匈牙利算法中只会用到从第一个
int match[N];    // 存储第二个集合中的每个点当前匹配的的第一个集合中的点是哪个
bool st[N];    // 表示第二个集合中的每个点是否已经被遍历过

bool find(int x)
{
    for (int i = h[x]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (!st[j])
        {
            st[j] = true;
            if (match[j] == 0 || find(match[j]))
            {
                match[j] = x;
                return true;
            }
        }
    }

    return false;
}

// 求最大匹配数, 依次枚举第一个集合中的每个点能否匹配第二个集合中的点
int res = 0;
for (int i = 1; i <= n1; i ++ )
{
    memset(st, false, sizeof st);
    if (find(i)) res ++ ;
}

```

81 评论

在这里写评论... (支持Markdown和Latex语法)

提交评论



(/user

/myspace


/index

/80510/)

小强d (/user/myspace/index/80510/) 9天前 回复

谢谢y总让我感受到算法之美。

常用代码模板4——数学知识

作者:  yxc (/user/myspace/index/1/), 2019-07-31 22:10:03, 阅读 25783

▲ 算法基础课相关代码模板

183 • 活动链接 —— 算法基础课 (<https://www.acwing.com/activity/content/11/>)

▼ 试除法判定质数 —— 模板题 AcWing 866. 试除法判定质数 (<https://www.acwing.com/problem/content/868/>)



333

```
bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
            return false;
    return true;
}
```

试除法分解质因数 —— 模板题 AcWing 867. 分解质因数 (<https://www.acwing.com/problem/content/869/>)

```
void divide(int x)
{
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            int s = 0;
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}
```

朴素筛法求素数 —— 模板题 AcWing 868. 筛质数 (<https://www.acwing.com/problem/content/870/>)

```
int primes[N], cnt;    // primes[]存储所有素数
bool st[N];           // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (st[i]) continue;
        primes[cnt ++ ] = i;
        for (int j = i + i; j <= n; j += i)
            st[j] = true;
    }
}
```

线性筛法求素数 —— 模板题 AcWing 868. 筛质数 (<https://www.acwing.com/problem/content/870/>)

```
int primes[N], cnt;    // primes[]存储所有素数
bool st[N];           // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

试除法求所有约数 —— 模板题 AcWing 869. 试除法求约数 (<https://www.acwing.com/problem/content/871/>)

```
vector<int> get_divisors(int x)
{
    vector<int> res;
    for (int i = 1; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res.push_back(i);
            if (i != x / i) res.push_back(x / i);
        }
    sort(res.begin(), res.end());
    return res;
}
```

约数个数和约数之和 —— 模板题 AcWing 870. 约数个数
(<https://www.acwing.com/problem/content/872/>), AcWing 871. 约数之和
(<https://www.acwing.com/problem/content/873/>)

如果 $N = p_1^{c_1} * p_2^{c_2} * \dots * p_k^{c_k}$
约数个数: $(c_1 + 1) * (c_2 + 1) * \dots * (c_k + 1)$
约数之和: $(p_1^0 + p_1^1 + \dots + p_1^{c_1}) * \dots * (p_k^0 + p_k^1 + \dots + p_k^{c_k})$

欧几里得算法 —— 模板题 AcWing 872. 最大公约数 (<https://www.acwing.com/problem/content/874/>)

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```

求欧拉函数 —— 模板题 AcWing 873. 欧拉函数 (<https://www.acwing.com/problem/content/875/>)

```
int phi(int x)
{
    int res = x;
    for (int i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);

    return res;
}
```

筛法求欧拉函数 —— 模板题 AcWing 874. 筛法求欧拉函数 (<https://www.acwing.com/problem/content/876/>)

```
int primes[N], cnt;    // primes[]存储所有素数
int euler[N];          // 存储每个数的欧拉函数
bool st[N];            // st[x]存储x是否被筛掉

void get_eulers(int n)
{
    euler[1] = 1;
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i])
        {
            primes[cnt ++ ] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            int t = primes[j] * i;
            st[t] = true;
            if (i % primes[j] == 0)
            {
                euler[t] = euler[i] * primes[j];
                break;
            }
            euler[t] = euler[i] * (primes[j] - 1);
        }
    }
}
```

快速幂 —— 模板题 AcWing 875. 快速幂 (<https://www.acwing.com/problem/content/877/>)

求 $m^k \bmod p$ ，时间复杂度 $O(\log k)$ 。

```
int qmi(int m, int k, int p)
{
    int res = 1 % p, t = m;
    while (k)
    {
        if (k & 1) res = res * t % p;
        t = t * t % p;
        k >>= 1;
    }
    return res;
}
```

扩展欧几里得算法 —— 模板题 AcWing 877. 扩展欧几里得算法 (<https://www.acwing.com/problem/content/879/>)

```
// 求x, y, 使得ax + by = gcd(a, b)
int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1; y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a/b) * x;
    return d;
}
```

高斯消元 —— 模板题 AcWing 883. 高斯消元解线性方程组 (<https://www.acwing.com/problem/content/885/>)

```

// a[N][N]是增广矩阵
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ )    // 找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i <= n; i ++ ) swap(a[t][i], a[r][i]);    // 将绝对
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c];    // 将当前上的
        for (int i = r + 1; i < n; i ++ )    // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1; // 有无穷多组解
    }

    for (int i = n - 1; i >= 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];

    return 0; // 有唯一解
}

```

递归法求组合数 —— 模板题 AcWing 885. 求组合数 I (<https://www.acwing.com/problem/content/887/>)

```

// c[a][b] 表示从a个苹果中选b个的方案数
for (int i = 0; i < N; i ++ )
    for (int j = 0; j <= i; j ++ )
        if (!j) c[i][j] = 1;
        else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;

```

通过预处理逆元的方式求组合数 —— 模板题 AcWing 886. 求组合数 II (<https://www.acwing.com/problem/content/888/>)

首先预处理出所有阶乘取模的余数`fact[N]`，以及所有阶乘取模的逆元`infact[N]`

如果取模的数是质数，可以用费马小定理求逆元

`int qmi(int a, int k, int p)` // 快速幂模板

```
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}
```

// 预处理阶乘的余数和阶乘逆元的余数

```
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i++)
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
```

Lucas定理 —— 模板题 AcWing 887. 求组合数 III (<https://www.acwing.com/problem/content/889/>)

若 p 是质数，则对于任意整数 $1 \leq m \leq n$ ，有：

$$C(n, m) = C(n \% p, m \% p) * C(n / p, m / p) \pmod{p}$$

```
int qmi(int a, int k)        // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

int C(int a, int b)        // 通过定理求组合数C(a, b)
{
    int res = 1;
    for (int i = 1, j = a; i <= b; i ++, j -- )
    {
        res = (LL)res * j % p;
        res = (LL)res * qmi(i, p - 2) % p;
    }
    return res;
}

int lucas(LL a, LL b)
{
    if (a < p && b < p) return C(a, b);
    return (LL)C(a % p, b % p) * lucas(a / p, b / p) % p;
}
```

分解质因数法求组合数 —— 模板题 AcWing 888. 求组合数 IV
(<https://www.acwing.com/problem/content/890/>)

当我们要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：

1. 筛法求出范围内的所有质数
2. 通过 $C(a, b) = a! / b! / (a - b)!$ 这个公式求出每个质因子的次数。 $n!$ 中 p 的
3. 用高精度乘法将所有质因子相乘

```
int primes[N], cnt;    // 存储所有质数
int sum[N];           // 存储每个质数的次数
bool st[N];           // 存储每个数是否已被筛掉

void get_primes(int n)    // 线性筛法求素数
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++)
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int get(int n, int p)    // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b)    // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i++)
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
```

```

        c.push_back(t % 10);
        t /= 10;
    }

    return c;
}

get_primes(a); // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )    // 求每个质因数的次数
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )    // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);

```

卡特兰数 —— 模板题 AcWing 889. 满足条件的01序列 (<https://www.acwing.com/problem/content/891/>)

给定 n 个0和 n 个1，它们按照某种顺序排成长度为 $2n$ 的序列，满足任意前缀中0的个数都不少于1的

NIM游戏 —— 模板题 AcWing 891. Nim游戏 (<https://www.acwing.com/problem/content/893/>)

给定 N 堆物品，第 i 堆物品有 A_i 个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。取走最后一件物品者获胜。两人都采取最优策略，问先手是否必胜。

我们把这种游戏称为NIM博弈。把游戏过程中面临的状态称为局面。整局游戏第一个行动的称为先手，第二个行动的称为后手。若在某一局面下无论采取何种行动，都会输掉游戏，则称该局面必败。

所谓采取最优策略是指，若在某一局面下存在某种行动，使得行动后对面面临必败局面，则优先采取该行动。同时，这样的局面被称为必胜。我们讨论的博弈问题一般都只考虑理想情况，即两人都无失误，都采取最优策略行动时游戏的结果。

NIM博弈不存在平局，只有先手必胜和先手必败两种情况。

定理：NIM博弈先手必胜，当且仅当 $A_1 \oplus A_2 \oplus \dots \oplus A_n \neq 0$

公平组合游戏ICG

若一个游戏满足：

1. 由两名玩家交替行动；
2. 在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关；
3. 不能行动的玩家判负；

则称该游戏为一个公平组合游戏。

NIM博弈属于公平组合游戏，但城建的棋类游戏，比如围棋，就不是公平组合游戏。因为围棋交战双方分别只能落黑子和白子，胜负判定也比较复杂，不满足条件2和条件3。

有向图游戏

给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子。两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。该游戏被称为有向图游戏。

任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个节点，并且从每个局面向沿着合法行动能够到达的下一个局面连有向边。

Mex运算

设 S 表示一个非负整数集合。定义 $\text{mex}(S)$ 为求出不属于集合 S 的最小非负整数的运算，即：

$\text{mex}(S) = \min\{x\}$, x 属于自然数，且 x 不属于 S

SG函数

在有向图游戏中，对于每个节点 x ，设从 x 出发共有 k 条有向边，分别到达节点 y_1, y_2, \dots, y_k ，定义 $\text{SG}(x)$ 为 x 的后继节点 y_1, y_2, \dots, y_k 的SG函数值构成的集合再执行 $\text{mex}(S)$ 运算的结果，即：

$\text{SG}(x) = \text{mex}(\{\text{SG}(y_1), \text{SG}(y_2), \dots, \text{SG}(y_k)\})$

特别地，整个有向图游戏 G 的SG函数值被定义为有向图游戏起点 s 的SG函数值，即 $\text{SG}(G) = \text{SG}(s)$ 。

有向图游戏的和 —— 模板题 AcWing 893. 集合-Nim游戏 (<https://www.acwing.com/problem/content/895/>)

设 G_1, G_2, \dots, G_m 是 m 个有向图游戏。定义有向图游戏 G ，它的行动规则是任选某个有向图游戏 G_i ，并在 G_i 上行动一步。 G 被称为有向图游戏 G_1, G_2, \dots, G_m 的和。

有向图游戏的和的SG函数值等于它包含的各个子游戏SG函数值的异或和，即：

$\text{SG}(G) = \text{SG}(G_1) \oplus \text{SG}(G_2) \oplus \dots \oplus \text{SG}(G_m)$

定理

有向图游戏的某个局面必胜，当且仅当该局面对应节点的SG函数值大于0。

有向图游戏的某个局面必败，当且仅当该局面对应节点的SG函数值等于0。

40 评论

在这里写评论... (支持Markdown和Latex语法)

提交评论



(/user

/myspace

/index

/73618/)

SQ_0 (/user/myspace/index/73618/) 10天前 回复

这个数学的东西蓝桥杯里多吗



(/user

/myspace

/index

/74126/)

Accept-ed (/user/myspace/index/74126/) 1个月前 回复

代码模板还有更新吗...很迫切的需要...



yxc (/user/myspace/index/1/) 11天前 回复

(/user 算法基础课新增了两道模板题。代码模板类似于数学公式，每年考的
/myspace 都一样，不需要经常更新呀hh

/index/1/)

Accept-ed (/user/myspace/index/74126/) 5天前 回复了 yxc 的评

(/user 论 回复

/myspace 好的谢谢y总，现已被清华大学计算机系录取，谢谢Y总的课程！

/index

/74126/)

SherlockOuO (/user/myspace/index/7646/) 1天前 回复了 Accept-

(/user ed 的评论 回复

/myspace 啊 这！牛啊

/index

/7646/)



(/user

/myspace

/index

/52033/)

Rambo (/user/myspace/index/52033/) 5个月前 回复

y总