acwing算法学习

第一章

课上：学思想

课下：背代码

题目，一道题写好几遍

记忆力 毅力/自制力

沉下心背东西

快速排序算法模板 —— 模板题 AcWing 785. 快速排序

分治

1、确定分界点，l、r、 (l+r)/2 随机

2、调整区间，分为两边，左边小于等于x,右边大于等于x

3、递归处理左右两段

```
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);
}
-std=c++11
```

归并排序算法模板 —— 模板题 AcWing 787. 归并排序

排序稳定：序列中相同的值排序后的相对位置是否发生改变

时间复杂度有 （nlogn)

1）确定分界点mid

2）递归排序两边

2）归并，合并为一个有序数组

```
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

边界问题

本质：区间内一半满足一半不满足

l=mid时加一

```
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

浮点数二分，比较对应整数二分

```
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

```cpp
// 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

```cpp
// 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

```cpp
// 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;
    }
    return C;
}
```

高精度除以低精度 —— 模板题 AcWing 794. 高精度除法

```
// 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

快速求区间和

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1] 从1开始，便于处理边界
```

二维前缀和 —— 模板题 AcWing 796. 子矩阵的和

```
S[i, j] = 第i行j列格子左上部分所有元素的和
以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：
S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]
```

一维差分 —— 模板题 AcWing 797. 差分

```
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/二维差分

```
给以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：
S[x1, y1] += c,
S[x2 + 1, y1] -= c,
S[x1, y2 + 1] -= c,
S[x2 + 1, y2 + 1] += c
```

双指针算法 —— 模板题 AcWIng 799. 最长连续不重复子序列, AcWing 800. 数组元素的目标和 _滑动窗口?

核心：把O(n^2)算法优化为O(n)

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

位运算 —— 模板题 AcWing 801. 二进制中1的个数

```
原码，反码，补码
求n二进制表示中第k位数字：n >> k & 1
返回n的最后一位1：lowbit(n) = n & -n 树状数组基本操作
```

整数离散化 —— 模板题 AcWing 802. 区间和

```
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())返回去重后数组末尾端点, 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

```
// 将所有存在交集的区间合并 贪心
void merge(vector<PII> &segs)
{
    vector<PII> res;
    sort(segs.begin(), segs.end()); //区间左端点排序
    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;
}
```

作者：yxc
链接：https://www.acwing.com/blog/content/277/
来源：AcWing

第二章

数据结构，以数组模拟的形式

指针+结构体： 面试题

单链表 —— 模板题 AcWing 826. 单链表

邻接表-存储树和图 静态链表

下标从0开始

```
// 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 ++ ;
}
// 插入下标k后面
void add(int k,int x)
{
    e[idx]= x, ne[idx] = ne[k], ne[k] = idx++ ;
}
// 将头结点删除，需要保证头结点存在
void remove()
{
    head = ne[head];
}
// 将k后面的点删掉
void remove(int k)
{
    ne[k] = ne[ne[k]];
}
```

双链表 —— 模板题 AcWing 827. 双链表

优化某些问题

```
// 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节点
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

```
// 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

```
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)
{
    //不空
}

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

```
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

```
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

用**模板**串来匹配**模式**串，找到模式串

s的真前缀以及真后缀是指不等于s的前缀以及后缀，即至少是**s[1~n-2]**或**s[0~n-1]**

ne[i] : 以**i**结尾的串中 **最长真前缀与真后缀相等的串** 的长度，如果没有则为0。

```cpp
//前缀h
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
```

实际使用范例（下面这个例子字符串从1开始）

```cpp
// 求Next数组：
// ne[i] 存储真前缀和真后缀相等的长度，所以至少从2开始：
abcab,从b开始才有真前缀
// s[]是模式串，p[]是模板串，n是s的长度，m是p的长度
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

```cpp
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, AcWing 837. 连通块中点的数量

按秩合并

字符按字符串读入

(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点的数量
// 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：
```

```
p[find(a)] = find(b);
size[b] += size[a];
```

(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, AcWing 839. 模拟堆

维护集合的数据结构 （大根堆，父节点值不小于子节点值）

```
// 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]]); 根据题意
    //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

```
(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

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;
```

```cpp
const int p = 131 or 13331 ;
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 mod 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

```
所有容器都有size() empty()

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
    vector<int> a(10,2);
    a.empty();
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）
    pair<int,int> p;
    p=make_pair(1,1);
    p={1,1};
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
        strcpy(c,s.c_str());//c_str()的使用方法

queue, 队列
    没有clear函数
    清空用

    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
    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(logn)
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 + logn)
        (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(logn)
    lower_bound()/upper_bound()
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 + logn)
        (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(logn)
    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
    none()  判断是否全为0      any()  判断是否至少有一个1
    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
```

作者：yxc
链接：https://www.acwing.com/blog/content/404/
来源：AcWing

第三章

树与图的存储

树是一种特殊的图: 无环连通图，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->b, b->a。
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->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)O(n+m), nn 表示点数，mm 表示边数

(1) 深度优先遍历 —— 模板题 AcWing 846. 树的重心

```
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

```
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

时间复杂度 O(n+m)O(n+m), nn 表示点数，mm 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;
}
```

最短路

单源最短路

所有边权均为正：朴素Dijkstra O(n^2)；堆优化版的Dijkstra(mlogn)

存在负权变：Bellman-Ford O(nm)； SPFA（队列优化Bellman-Ford) 一般：O(m) 最坏 O(nm)

多元汇最短路：Floyed算法 O(n^3)

朴素dijkstra算法 —— 模板题 AcWing 849. Dijkstra求最短路 I 基于贪心

时间复杂是 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

时间复杂度 O(mlogn), 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

```

```cpp
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

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。

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

```
    }
    if (dist[n] > 0x3f3f3f3f / 2) return -1;
    return dist[n];
}
```

spfa 算法（队列优化的Bellman-Ford算法） —— 模板题 AcWing 851. spfa求最短路

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

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的最短路中经过的点数
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，由抽屉原理一定有两个点相同，所以存在环。
    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的最短路中包含至少n个点（不包括自己），则说明存在环
                if (!st[j])
                {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
    return false;
}
```

floyd算法 —— 模板题 AcWing 854. Floyd求最短路

```
时间复杂度是 O(n3)O(n3)，nn 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]);
```

```
}
```

最小生成树（无向图）

Prim : 朴素版Prim O(n^2)(稠密图) ; 堆优化版Prim O(mlogn)(一般不会用)

Kruskal ：O(mlogm)(稀疏图)

朴素版prim算法 —— 模板题 AcWing 858. Prim算法求最小生成树

时间复杂度是 O(n^2+m), n 表示点数，m 表示边数

初始化距离为正无穷，迭代所有点，找到集合中最近的点，更新它到**集合**的距离，把t加到集合中。

```
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

时间复杂度是 O(mlogm), 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;
    // 重载小于号运算符
    // 用于比较两个Edge对象的大小关系
    bool operator < (const Edge &W) const
    {
        return w < W.w; // 如果当前对象的边权小于W对象的边权，返回true，否则返回
false
    }
}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（本质dfs)判别二分图 —— 模板题 AcWing 860. 染色法判定二分图

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表示为染色，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

时间复杂度最坏是 O(nm)，实际运行时间一般远小于O(nm)， n 表示点数，m 表示边数

做错一件事，错过一件事

```
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二个集合指向第一个集合的边，所以这里只用存一个方向的边
int match[N];       // 存储第二个集合中的每个点当前匹配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 ++ ;
}
```

作者：yxc
链接：https://www.acwing.com/blog/content/405/
来源：AcWing

第四章

试除法判定质数 —— 模板题 AcWing 866. 试除法判定质数

质数：大于1的整数中，如果只包含1和本身两个约束，称为质数（素数）

（1）判定，试除法

（2）分解质因数

```
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

```
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

```
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; j <= n; j += i)
            st[j] = true;
    }
}
```

线性筛法求素数 —— 模板题 AcWing 868. 筛质数

被最小质因子筛掉

```
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

int范围内约数个数最多为1500个左右

```
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, AcWing 871. 约数之和

```
如果 N = p1^c1 * p2^c2 * ... *pk^ck
约数个数：  (c1 + 1) * (c2 + 1) * ... * (ck + 1)
约数之和：  (p1^0 + p1^1 + ... + p1^c1) * ... * (pk^0 + pk^1 + ... + pk^ck)
```

欧几里得算法 —— 模板题 AcWing 872. 最大公约数

```
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

欧拉函数公式（容斥原理）：n*(1-p1)*(1-p2)...(1-pk)

```
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

O(n) 线性求所有数的欧拉函数

应用：欧拉定理：a与n互质 a^φ(n)=1(mod n)

```
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

求 m^k mod p，时间复杂度 O(logk)。

```
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

```
// 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

O(n^3)时间内解n个方程n个未知数的解。

解：无解； 无穷多组解； 唯一解；

- 完美阶梯型：唯一解
- 0=非零： 无解
- 0=0： 无穷多组解

高斯消元：

枚举每一列c：

1. 找到绝对值最大的一行
2. 将该行换到最上面
3. 将该行第一个数变成1
4. 将下面所有行的第c列消成0

```
// 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]);      // 将绝对值最大的行换到最顶端
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c];      // 将当前上的首位变成1
        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

十万 1<b<a<2000 O(n^2)

```
// 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

一万 1<b<a<10^5 O(logn)

```
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;
}
ll C(ll n,ll m){
    if(m>n) return 0ll;
    return fact[n]*infact[m]%mod*infact[n-m]%mod;
}
```

*Lucas*定理 —— 模板题 AcWing 887. 求组合数 III

组合数，1<b<a

```
若p是质数，则对于任意整数 1 <= m <= n，有：
    C(n, m) = C(n % p, m % p) * C(n / p, m / p) (mod 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

当我们需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的次数是
n / p + n / p^2 + n / p^3 + ...
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

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的个数的序列的数量为：

$$Cat(n) = C(2n, n)/(n+1) \tag{1}$$

容斥原理

找1~n中能至少被素数p1,p2,...,pn一个整除的整数有多少个。

位运算对应容斥原理集合，1~n中能被x整除的个数为n/x，奇数加上，偶数减去

NIM(尼姆)游戏 —— 模板题 AcWing 891. Nim游戏

给定**N**堆物品，第**i**堆物品有**Ai**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，即两人均无失误，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 A1 ^ A2 ^ ... ^ An != 0

公平组合游戏ICG
若一个游戏满足：

由两名玩家交替行动；
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；
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mex(S)为求出不属于集合S的最小非负整数的运算，即：
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y1, y2, ..., yk，定义SG(x)为x的后继节点y1, y2, ..., yk 的SG函数值构成的集合再执行**mex(S)**运算的结果，即：
SG(x) = mex({SG(y1), SG(y2), ..., SG(yk)})
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SG(G) = SG(s)。

```
int sg(int x){
    if (f[x] != -1) return f[x];
    unordered_set<int> s;
    for(int i = 0; i < m; i++ ) {
        int sum = s[i];
        if (x >= sum) s.insert(sg(x - sum));
    }
    for (int i=0; ; i++ )
        if (!s.count(i))
                return f[x] = i;
}
```

有向图游戏的和 —— 模板题 AcWing 893. 集合-Nim游戏

设G1, G2, ..., Gm 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Gi，并在Gi上行动一步。G被称为有向图游戏G1, G2, ..., Gm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：
SG(G) = SG(G1) ^ SG(G2) ^ ... ^ SG(Gm)

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。

作者：yxc
链接：https://www.acwing.com/blog/content/406/
来源：AcWing

动态规划

常见模型

背包 九讲

01背包

每件物品最多可以用一次

体积从大到小的原因： 如果从小到大，则 `f[j-w[i]]+v[i]` 实际是 `f[i-1][j-w[i]]+v[i]`，需要的是上一维度f[i-1]维度，所以从大到小可以使用上一维度的，因为这一维度的还没有被计算到。

```
    for(int i=1;i<=N;i++){
        for(int j=V;j>=w[i];j--){
            f[j]=max(f[j],f[j-w[i]]+v[i]);
        }
    }
```

完全背包

每件物品可以用无限次

```
    for(int i=1;i<=N;i++){
        for(int j=w[i];j<=V;j++){
            f[j]=max(f[j],f[j-w[i]]+v[i]);
        }
    }
```

多重背包

每件物品特定数量

二进制优化

```
    for(int i=1;i<=N;i++){
        cin>>a>>b>>c; //权重a,价值b，数量c
        int k=1;
        while(k<=c){
            cnt++; c-=k;
            w[cnt]=k*a;
            v[cnt]=k*b;
            k*=2;
        }
        if(c){
            cnt++;
            w[cnt]=c*a;
            v[cnt]=c*b;
        }
    } //之后用01背包
```

单调队列优化

```
    for(int i=1;i<=N;i++){ // 遍历每件物品
        memcpy(g, f, sizeof g); // 将上一轮的最优解拷贝给g数组
        for(int r=0;r<v[i];r++){ // 遍历余数r（用于优化循环）
            int h=0,t=-1; // 滑动窗口的起始索引和结束索引
            for(int l=r;l<=V;l+=v[i]){ // 遍历背包容量，间隔为当前物品体积
                while(h<=t && l-q[h]>s[i]*v[i]) h++; // 如果窗口左边界超出限制，则左边界右移
                while(h<=t && g[q[t]] + (l-q[t])/v[i]*w[i]<=g[l]) t--; // 保持窗口单调性，将窗口内不符合条件的解移除
                q[++t] = l; // 将当前状态的背包容量加入窗口
                f[l] = g[q[h]] + (l-q[h])/v[i]*w[i]; // 更新当前状态的最优解
            }
        }
    }
```

分组背包

若干组，一组只能选一个

```
for(int i=1;i<=N;i++){
    for(int j=V;j>=0;j--){
        for(int k=1;k<=s[i];k++){
            if(w[i][k]<=j){
                f[j]=max(f[j],f[j- w[i][k] ]+v[i][k]);
            }
        }
    }
}
```

混合背包

转换成01背包问题

```
for(int i=1;i<=N;i++){
    cin>>a>>b>>s;
    if(s==-1) s=1;
    else if(s==0) s=V/a;
    k=1;
    while(k<=s){
        s-=k; cnt++;
        v[cnt] = k*a; w[cnt] = k*b;
        k*=2;
    }
    if(s){
        cnt++;
        v[cnt] = s*a; w[cnt] = s*b;
    }
}
for(int i=1;i<=cnt;i++){
    for(int j=V;j>=v[i];j--){
        f[j] = max(f[j], f[j-v[i]]+w[i]);
    }
}
```

二维费用的背包问题

采用两层循环

```
cin>>N>>V>>W;
for(int i=1;i<=N;i++){
    cin>>v>>m>>w;
    for(int j=V;j>=v;j--){
        for(int k=W;k>=m;k--){
            f[j][k] = max(f[j][k], f[j-v][k-m] + w);
        }
    }
}
cout<<f[V][W]<<endl;
```

线性dp

数字三角形

```
    for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++){
            a[i][j]+=max(a[i-1][j],a[i-1][j-1]);
        }
    }
```

LIS

```
    for(int i=1;i<=n;i++){
        for(int j=1;j<i;j++){
            if(a[j]<a[i]) f[i]=max(f[j]+1,f[i]);
        }
    }
    for(int i=1;i<=n;i++){
        ma=max(f[i],ma);
    }
```

LCS

```
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            f[i][j]=max(f[i-1][j],f[i][j-1]);
            if(a[i]==b[j]) f[i][j]=f[i-1][j-1]+1;
        }
    }
```

区间dp

石子合并

```
     memset(f, 0x3f, sizeof f);
     for(int len=2;len<=n;len++){
         for(int i=1;i+len-1<=n;i++){
             int j=i+len-1;
             if (len == 1) {
                 f[i][j] = 0;  // 边界初始化
                 continue;
             }
             for(int k=i;k<j;k++){
                 f[l][r]=min(f[l][r],f[l][k]+f[k+1][r]+s[r]-s[l-1]);
             }
         }
     }
     cout<<f[1][n];
```

数位统计dp

状态表示

分情况讨论

状态压缩 dp

蒙德里安的梦想

最短Hamilton距离

树形dp

没有上司的舞会

记忆化

滑雪

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=1e4;
ll f[maxn][maxn],g[maxn][maxn]; //distance
struct node{
    ll i,j,num;
}a[maxn];
struct cmp
{
    bool operator () (node x,node y){
        return x.num>y.num;
    }

};
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    ll n,m,ma=-1;
    cin>>n>>m;
    priority_queue<node, vector<node>,cmp>pq;
    for(ll i=1;i<=n;i++){
        for(ll j=1;j<=m;j++){
            f[i][j]=1;
            node a;
            a.i=i;
            a.j=j;
            cin>>a.num;
            g[i][j]=a.num;
            pq.push(a);
        }
    }
    while(!pq.empty()){
        node t=pq.top();
        pq.pop();
        ll i=t.i;
```

```
        ll j=t.j;
        ll nu=t.num;
        if(g[i-1][j]<nu) f[i][j]=max(f[i][j],f[i-1][j]+1);
        if(g[i+1][j]<nu) f[i][j]=max(f[i][j],f[i+1][j]+1);
        if(g[i][j-1]<nu) f[i][j]=max(f[i][j],f[i][j-1]+1);
        if(g[i][j+1]<nu) f[i][j]=max(f[i][j],f[i][j+1]+1);
        ma=max(ma,f[i][j]);
    }
    cout<<ma<<endl;
    return 0;
}

```

搜索