链表、栈、队列

调度场算法

```python
prec = {"(": 1, "+": 2, "-": 2, "*": 3, "/": 3}
def infixToPostfix(infix):
    global prec
    op_stack = []
    ans = []
    for token in infix:
        try:
            float(token)
            ans.append(token)
        except ValueError:
            if token == "(":
                op_stack.append(token)
            elif token == ")":
                while (op := op_stack.pop()) != "(":
                    ans.append(op)
            else:
                while op_stack and prec[op_stack[-1]] >= prec[token]:
                    ans.append(op_stack.pop())
                op_stack.append(token)
    while op_stack:
        ans.append(op_stack.pop())
    return " ".join(map(str, ans))

def expToList(exp):
    global prec
    infix = []
    last = 0
    for i in range(len(exp)):
        if exp[i] in prec or exp[i] == ')':
            if exp[last:i]:
                infix.append(exp[last:i])
            infix.append(exp[i])
            last = i + 1
    if exp[last:]:
        infix.append(exp[last:])
    return infix


for i in range(int(input())):
    print(infixToPostfix(expToList(input().strip())))
```

快慢指针

树

前中序构建树

```python
def construct(i, left, right):
    if left <= right:
        node = TreeNode(preorder[root])
        i = dic[preorder[root]]
        node.left = recur(root + 1, left, i - 1)
        node.right = recur(i - left + root + 1, i + 1, right)
        return node
dic = {}
for i in range(len(inorder)):
    dic[inorder[i]] = i
```

前缀树

图

最短路

迪杰斯特拉算法

```
while heap:
    d, node = heappop(heap)
    if node == N - 1:
        dist[-1] = d
        break
    for i, w in adj_mat[node]:
        if d + w < dist[i]:
            dist[i] = d + w
            heappush(heap, (dist[i], i))
```

变形：双变量

```
while heap:
    distance, cost, node = heappop(heap)
    if node == N:
        ans = distance
        break
    if distance > length[node][cost]:
        continue
    for l, c, n in dist[node]:
        if c + cost <= K and l + distance < length[n][c + cost]:
            length[n][c + cost] = l + distance
            heappush(heap, (length[n][c + cost], c + cost, n))
```

弗洛伊德-华沙算法

```
for k in range(n):
    for i in range(n):
        for j in range(n):
            f[i][j] = f[i][k] + f[k][j]
```

SPFA

```
while queue:
    node = queue.popleft()
    vis[node] = False
    for i in range(N):
        if edge[node][i] < 1e5 and edge[0][i] > edge[0][node] + edge[node][i]:
            edge[0][i] = edge[0][node] + edge[node][i]
            if not vis[i]:
                queue.append(i)
                vis[i] = True
```

最小生成树

```
while heap:
    d, node = heappop(heap)
    if node in vis:
        continue
    vis.add(node)
    ans += d
    if len(vis) == n:
        break
    for r, star in adj_tab[node]:
        if star not in vis:
            heappush(heap, (r, star))

```

拓扑排序

```
from heapq import heappush, heappop
class Vertex:
    def __init__(self, n):
        self.num = n
        self.name = f"v{n + 1}"
        self.ind = 0
v, a = map(int, input().split())
ans = []
heap = []
adj_mat = [[False for i in range(v)] for j in range(v)]
nodes = [Vertex(i) for i in range(v)]
for i in range(a):
    n1, n2 = map(int, input().split())
    if not adj_mat[n1 - 1][n2 - 1]:
        nodes[n2 - 1].ind += 1
        adj_mat[n1 - 1][n2 - 1] = True
for i in nodes:
    if not i.ind:
        heappush(heap, i.num)
while heap:
    node = heappop(heap)
    for i in range(v):
        if adj_mat[node][i]:
            nodes[i].ind -= 1
            if nodes[i].ind == 0:
                heappush(heap, i)
    ans.append(nodes[node].name)

print(*ans)
```

如果不要求节点从小到大，`heap` 可以改用 `queue`，`heappop` 改为 `popleft`

有向图判环

- 染色法，遇到正在访问的节点说明有环

其他

KMP算法

```python
""""
compute_lps 函数用于计算模式字符串的LPS表。LPS表是一个数组，
其中的每个元素表示模式字符串中当前位置之前的子串的最长前缀后缀的长度。
该函数使用了两个指针 length 和 i，从模式字符串的第二个字符开始遍历。
"""
def compute_lps(pattern):
    """
    计算pattern字符串的最长前缀后缀（Longest Proper Prefix which is also Suffix）表
    :param pattern: 模式字符串
    :return: lps表
    """
    m = len(pattern)
    lps = [0] * m  # 初始化lps数组
    length = 0  # 当前最长前后缀长度
    for i in range(1, m):  # 注意i从1开始，lps[0]永远是0
        while length > 0 and pattern[i] != pattern[length]:
            length = lps[length - 1]  # 回退到上一个有效前后缀长度
        if pattern[i] == pattern[length]:
            length += 1
        lps[i] = length

    return lps

def kmp_search(text, pattern):
    n = len(text)
    m = len(pattern)
    if m == 0:
        return 0
    lps = compute_lps(pattern)
    matches = []

    # 在 text 中查找 pattern
    j = 0  # 模式串指针
    for i in range(n):  # 主串指针
        while j > 0 and text[i] != pattern[j]:
            j = lps[j - 1]  # 模式串回退
        if text[i] == pattern[j]:
            j += 1
        if j == m:
            matches.append(i - j + 1)  # 匹配成功
            j = lps[j - 1]  # 查找下一个匹配

    return matches

text = "ABABABABCABABABABCABABABABC"
pattern = "ABABCABAB"
index = kmp_search(text, pattern)
print("pos matched: ", index)
# pos matched:  [4, 13]
```

注意点

1. 读入的如果是数字不能完全依靠`isdigit`，考虑负数（`-`），小数（`.`）
2. 只有`node.left`和`node.right`均为`None`的节点才是叶结点