LeetCode高階算法綜合解析：從圖論到狀態壓縮的完全指南

一、圖論問題的綜合解法

1.1 最短路徑算法變形

1.1.1 網絡延遲時間（LeetCode 743）

應用Dijkstra算法實現分佈式系統延時計算：

```
import heapq

def networkDelayTime(times, n, k):
    graph = [[] for _ in range(n+1)]
    for u, v, w in times:
        graph[u].append((v, w))

    dist = [float('inf')] * (n+1)
    dist[k] = 0
    heap = [(0, k)]

    while heap:
        d, u = heapq.heappop(heap)
        if d > dist[u]:
            continue
        for v, w in graph[u]:
            if dist[v] > dist[u] + w:
                dist[v] = dist[u] + w
                heapq.heappush(heap, (dist[v], v))

    max_dist = max(dist[1:])
    return max_dist if max_dist < float('inf') else -1
```

關鍵點在於優先隊列維護當前最短路徑，時間複雜度O(E + V log V)[1] [2]

1.1.2 課程安排III（LeetCode 630）

使用貪心策略與最大堆維護最優課程組合：

```
import heapq

def scheduleCourse(courses):
    courses.sort(key=lambda x: x[^1])
    heap = []
    time = 0
    for duration, end in courses:
        if time + duration <= end:
            heapq.heappush(heap, -duration)
            time += duration
        elif heap and -heap[^0] > duration:
            time += duration + heapq.heappop(heap)
            heapq.heappush(heap, -duration)
    return len(heap)
```

通過動態調整已選課程，保證總用時最小化[2:1]

1.2 最小生成樹應用

1.2.1 連接所有城市的最低成本（LeetCode 1135）

Kruskal算法實現城市連通最優解：

```
class UnionFind:
    def __init__(self, size):
        self.parent = list(range(size))
        self.rank = [^0]*size

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        xroot = self.find(x)
        yroot = self.find(y)
        if xroot == yroot:
            return False
        if self.rank[xroot] < self.rank[yroot]:
            self.parent[xroot] = yroot
        else:
            self.parent[yroot] = xroot
            if self.rank[xroot] == self.rank[yroot]:
                self.rank[xroot] += 1
        return True

def minimumCost(n, connections):
    uf = UnionFind(n)
    connections.sort(key=lambda x: x[^2])
    total = 0
    count = 0
    for u, v, cost in connections:
        if uf.union(u-1, v-1):
            total += cost
            count += 1
            if count == n-1:
                return total
    return -1
```

並查集結構高效處理集合合併操作，時間複雜度O(E log E) [2:2]

二、樹形DP的進階應用

2.1 二叉樹監控佈局（LeetCode 968）

狀態機模型實現最小攝像頭覆蓋：

```
def minCameraCover(root):
    result = 0
    def dfs(node):
        nonlocal result
        if not node:
            return 2
        left = dfs(node.left)
        right = dfs(node.right)
        if left == 0 or right == 0:
            result += 1
            return 1
        return 2 if left == 1 or right == 1 else 0
    return (dfs(root) == 0) + result
```

定義三種節點狀態：0未監控、1攝像頭、2已監控，自底向上決策 [3]

2.2 二叉樹中最長交錯路徑（LeetCode 1372）

雙向遞歸追蹤左右轉向次數：

```
def longestZigZag(root):
    max_len = 0
    def dfs(node, is_left, length):
        nonlocal max_len
        if not node:
            return
        max_len = max(max_len, length)
        if is_left:
            dfs(node.right, False, length+1)
            dfs(node.left, True, 1)
        else:
            dfs(node.left, True, length+1)
            dfs(node.right, False, 1)
    dfs(root, True, 0)
    dfs(root, False, 0)
    return max_len
```

通過方向標記記錄當前路徑形態，時間複雜度O(n) [3:1]

三、回溯法的剪枝藝術

3.1 數獨求解器（LeetCode 37）

位運算加速候選數篩選：

```
def solveSudoku(board):
    rows = [^0]*9
    cols = [^0]*9
    boxes = [^0]*9
    to_fill = []

    for i in range(9):
        for j in range(9):
            if board[i][j] == '.':
                to_fill.append((i,j))
            else:
                num = int(board[i][j])
                mask = 1 << (num-1)
                rows[i] |= mask
                cols[j] |= mask
                boxes[(i//3)*3 + j//3] |= mask

    def backtrack(index):
        if index == len(to_fill):
            return True
        i,j = to_fill[index]
        b = (i//3)*3 + j//3
        used = rows[i] | cols[j] | boxes[b]
        for num in range(1,10):
            mask = 1 << (num-1)
            if not (used & mask):
                rows[i] |= mask
                cols[j] |= mask
                boxes[b] |= mask
                board[i][j] = str(num)
                if backtrack(index+1):
                    return True
                rows[i] ^= mask
                cols[j] ^= mask
                boxes[b] ^= mask
        return False

    backtrack(0)
```

利用位掩碼快速判斷可用數字，減少重複檢查[2:3]

3.2 排列序列（LeetCode 60）

階乘數系統定位第k個排列：

```
def getPermutation(n, k):
    factorials = [^1]
    for i in range(1, n):
        factorials.append(factorials[-1] * i)
    k -= 1
    numbers = list(range(1, n+1))
    result = []
    for i in range(n-1, -1, -1):
        index = k // factorials[i]
        result.append(str(numbers.pop(index)))
        k %= factorials[i]
    return ''.join(result)
```

通過數學分析跳過無效排列，時間複雜度O(n) [2:4]

四、動態規劃的狀態壓縮

4.1 最大矩形（LeetCode 85）

柱狀圖擴展法結合單調棧：

```
def maximalRectangle(matrix):
    if not matrix:
        return 0
    m, n = len(matrix), len(matrix[^0])
    heights = [^0]*n
    max_area = 0
    for i in range(m):
        for j in range(n):
            heights[j] = heights[j]+1 if matrix[i][j] == '1' else 0
        stack = [-1]
        for k in range(n):
            while stack[-1] != -1 and heights[k] < heights[stack[-1]]:
                h = heights[stack.pop()]
                w = k - stack[-1] - 1
                max_area = max(max_area, h*w)
            stack.append(k)
        while stack[-1] != -1:
            h = heights[stack.pop()]
            w = n - stack[-1] - 1
            max_area = max(max_area, h*w)
    return max_area
```

逐行構建高度數組，利用單調棧快速計算最大面積 [1:1] [4]

4.2 股票買賣最佳時機IV（LeetCode 188）

滾動數組優化空間複雜度：

```
def maxProfit(k, prices):
    if not prices:
        return 0
    n = len(prices)
    if k >= n//2:
        return sum(max(0, prices[i]-prices[i-1]) for i in range(1,n))

    buy = [-float('inf')]*(k+1)
    sell = [^0]*(k+1)
    for price in prices:
        for j in range(1, k+1):
            buy[j] = max(buy[j], sell[j-1]-price)
            sell[j] = max(sell[j], buy[j]+price)
    return sell[k]
```

通過狀態分離實現O(k)空間複雜度，時間複雜度O(nk) [1:2] [4:1]

五、排序算法的變形應用

5.1 逆序對計數（LeetCode 315）

歸併排序變形實現高效計數：

```
def countSmaller(nums):
    def sort(enum):
        half = len(enum) // 2
        if half:
            left, right = sort(enum[:half]), sort(enum[half:])
            for i in reversed(range(len(enum))):
                if not right or (left and left[-1][^1] > right[-1][^1]):
                    res[left[-1][^0]] += len(right)
                    enum[i] = left.pop()
                else:
                    enum[i] = right.pop()
        return enum
    res = [^0]*len(nums)
    sort(list(enumerate(nums)))
    return res
```

在合併過程中統計右側小於當前元素的數量，時間複雜度O(n log n) [2:5]

5.2 最大間距（LeetCode 164）

桶排序思想實現線性時間複雜度：

```
def maximumGap(nums):
    if len(nums) < 2:
        return 0
    min_val, max_val = min(nums), max(nums)
    bucket_size = max(1, (max_val - min_val) // (len(nums)-1))
    bucket_num = (max_val - min_val) // bucket_size + 1
    buckets = [[None, None] for _ in range(bucket_num)]

    for num in nums:
        idx = (num - min_val) // bucket_size
        if not buckets[idx][^0]:
            buckets[idx][^0] = buckets[idx][^1] = num
        else:
            buckets[idx][^0] = min(buckets[idx][^0], num)
            buckets[idx][^1] = max(buckets[idx][^1], num)

    max_gap = 0
    prev_max = min_val
    for bucket in buckets:
        if not bucket[^0]:
            continue
        max_gap = max(max_gap, bucket[^0] - prev_max)
        prev_max = bucket[^1]
    return max_gap
```

利用桶的間隔性質保證最大間距出現在跨桶元素間[2:6]

六、狀態壓縮DP的精妙設計

6.1 旅行商問題（LeetCode 943）

位掩碼表示訪問狀態：

```
def shortestSuperstring(words):
    n = len(words)
    overlap = [[^0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            a, b = words[i], words[j]
            for k in range(min(len(a), len(b)), -1, -1):
                if a.endswith(b[:k]):
                    overlap[i][j] = k
                    break

    dp = [[^0]*n for _ in range(1<<n)]
    parent = [[None]*n for _ in range(1<<n)]

    for mask in range(1, 1<<n):
        for bit in range(n):
            if not (mask & (1<<bit)):
                continue
            prev_mask = mask ^ (1<<bit)
            if prev_mask == 0:
                continue
            for j in range(n):
                if prev_mask & (1<<j):
                    val = dp[prev_mask][j] + overlap[j][bit]
                    if val > dp[mask][bit]:
                        dp[mask][bit] = val
                        parent[mask][bit] = j

    mask = (1<<n)-1
    max_overlap = max(dp[mask])
    for i in range(n):
        if dp[mask][i] == max_overlap:
            result = words[i]
            parent_idx = parent[mask][i]
            while parent_idx is not None:
                overlap_len = overlap[parent_idx][i]
                result = words[parent_idx][:-overlap_len] + result
                i = parent_idx
                mask ^= 1<<i
                parent_idx = parent[mask][i] if mask else None
            return result
    return ""
```

動態規劃狀態轉移中利用位掩碼記錄節點訪問情況 [1:3] [4:2]

結論

從圖論到狀態壓縮，掌握這些核心算法模式需把握三個關鍵：1) 建立清晰的狀態定義 2) 設計高效的狀態轉移方程 3) 選擇合適的數據結構進行優化。建議在實戰中通過以下步驟提升：

1. 優先理解問題的數學本質，識別潛在的算法模式
2. 手動推導小規模案例，驗證狀態轉移邏輯
3. 逐步加入空間和時間優化手段
4. 對比不同解法的效率差異，分析取捨條件

針對LeetCode高頻難題，可重點研究「問題變形模式」與「算法組合應用」這兩大方向，例如將Dijkstra算法與二分查找結合解決帶限制的最短路徑問題，或將動態規劃與貪心策略混合使用。持續進行系統化專題訓練，可快速建立解題直覺。

⁂

1. https://hackmd.io/@bangyewu/H1PYYF3Z6
2. https://www.secondlife.tw/algorithms-backtracking/
3. https://hackmd.io/@bangyewu/B1jBD5WQp
4. https://vocus.cc/article/650d8ffbfd89780001ab0197