Lời giải chi tiết các bài tập Quay lui (Backtracking)

1. Liệt kê các dãy nhị phân độ dài N

python

```python
def binary_strings(n):
    result = []
    current = [0] * n

    def backtrack(pos):
        if pos == n:
            result.append(''.join(map(str, current)))
            return

        # Thử đặt số 0 tại vị trí pos
        current[pos] = 0
        backtrack(pos + 1)

        # Thử đặt số 1 tại vị trí pos
        current[pos] = 1
        backtrack(pos + 1)

    backtrack(0)
    return result

# Ví dụ
print(binary_strings(3))
# Output: ['000', '001', '010', '011', '100', '101', '110', '111']
```

Giải thích:

- Tại mỗi vị trí, ta có 2 lựa chọn: đặt số 0 hoặc 1
- Sau khi đặt xong n số, ta thêm dãy vào kết quả
- Độ phức tạp thời gian: O(2^n) vì có 2^n dãy nhị phân độ dài n

2. Liệt kê hoán vị

python

```python
def permutations(n):
    result = []
    current = []
    used = [False] * (n + 1)

    def backtrack():
        if len(current) == n:
            result.append(current[:])
            return

        for i in range(1, n + 1):
            if not used[i]:
                used[i] = True
                current.append(i)
                backtrack()
                current.pop()
                used[i] = False

    backtrack()
    return result

# Ví dụ
print(permutations(3))
# Output: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

Giải thích:

- Ta sử dụng mảng `used` để đánh dấu các số đã được sử dụng
- Tại mỗi bước, thử tất cả các số chưa được sử dụng
- Độ phức tạp thời gian: O(n!) vì có n! hoán vị của n phần tử

3. Bài toán N-Queens cơ bản

python

```python
def solve_n_queens(n):
    board = [['.' for _ in range(n)] for _ in range(n)]
    solutions = []

    def is_safe(row, col):
        # Kiểm tra hàng ngang
        for i in range(col):
            if board[row][i] == 'Q':
                return False

        # Kiểm tra đường chéo trên bên trái
        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
            if board[i][j] == 'Q':
                return False

        # Kiểm tra đường chéo dưới bên trái
        for i, j in zip(range(row, n), range(col, -1, -1)):
            if board[i][j] == 'Q':
                return False

        return True

    def backtrack(col):
        if col == n:
            solutions.append([''.join(row) for row in board])
            return True

        for row in range(n):
            if is_safe(row, col):
                board[row][col] = 'Q'
                if backtrack(col + 1):
                    return True
                board[row][col] = '.'

        return False

    backtrack(0)
    return solutions[0] if solutions else None

# Ví dụ
solution = solve_n_queens(4)
for row in solution:
    print(row)
```

```
# Output:
# .Q..
# ...Q
# Q...
# ..Q.
```

Giải thích:

- Ta đặt quân hậu vào từng cột và kiểm tra xem có an toàn không
- Một quân hậu an toàn khi không bị tấn công bởi quân hậu khác
- Độ phức tạp thời gian: O(n!)

4. Tổ hợp chập K của N phần tử

python

```python
def combinations(n, k):
    result = []
    current = []

    def backtrack(start):
        if len(current) == k:
            result.append(current[:])
            return

        for i in range(start, n + 1):
            current.append(i)
            backtrack(i + 1)
            current.pop()

    backtrack(1)
    return result

# Ví dụ
print(combinations(4, 2))
# Output: [[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]
```

Giải thích:

- Tại mỗi bước, ta quyết định có chọn phần tử i hay không
- Khi đã chọn đủ k phần tử, ta thêm tổ hợp vào kết quả
- Độ phức tạp thời gian: O(C(n,k)) - số tổ hợp chập k của n phần tử

5. Chia tập hợp

python

```python
def partition_set(n, k):
    result = []
    partition = [[] for _ in range(k)]

    def backtrack(pos):
        if pos > n:
            # Kiểm tra xem tất cả các tập con có rỗng không
            if all(partition):
                result.append([p[:] for p in partition])
            return

        for i in range(k):
            partition[i].append(pos)
            backtrack(pos + 1)
            partition[i].pop()

    backtrack(1)
    return result

# Ví dụ
print(partition_set(3, 2))
# Output:
# [[[1, 2], [3]], [[1, 3], [2]], [[1], [2, 3]], [[2], [1, 3]], [[3], [1, 2]], [[1], [2], [3]]]
```

Giải thích:

- Với mỗi phần tử, ta thử đặt nó vào từng tập con
- Khi đã xử lý hết n phần tử, kiểm tra xem các tập con có rỗng không
- Độ phức tạp thời gian: O(k^n) vì mỗi phần tử có k lựa chọn

6. Sudoku Solver

python

```python
def solve_sudoku(board):
    def is_valid(row, col, num):
        # Kiểm tra hàng
        for x in range(9):
            if board[row][x] == num:
                return False

        # Kiểm tra cột
        for x in range(9):
            if board[x][col] == num:
                return False

        # Kiểm tra ô 3x3
        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(3):
            for j in range(3):
                if board[i + start_row][j + start_col] == num:
                    return False

        return True

    def solve():
        for i in range(9):
            for j in range(9):
                if board[i][j] == 0:
                    for num in range(1, 10):
                        if is_valid(i, j, num):
                            board[i][j] = num
                            if solve():
                                return True
                            board[i][j] = 0
                    return False
        return True

    solve()
    return board

# Ví dụ
board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
```

```
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]
solution = solve_sudoku(board)
for row in solution:
    print(row)
```

Giải thích:

- Tìm ô trống đầu tiên và thử điền các số từ 1 đến 9
- Nếu một số hợp lệ, ta điền nó vào và giải tiếp
- Nếu không thể điền tiếp, ta quay lui và thử số khác
- Độ phức tạp thời gian: O(9^(n*n)) trong trường hợp xấu nhất, với n là kích thước bàn cờ

7. Bài toán người du lịch (TSP)

python

```python
def tsp(graph):
    n = len(graph)
    all_cities = set(range(n))
    memo = {}

    def dp(curr, remaining):
        if not remaining:
            return graph[curr][0]

        if (curr, tuple(sorted(remaining))) in memo:
            return memo[(curr, tuple(sorted(remaining)))]

        min_cost = float('inf')
        for next_city in remaining:
            new_remaining = remaining - {next_city}
            cost = graph[curr][next_city] + dp(next_city, new_remaining)
            min_cost = min(min_cost, cost)

        memo[(curr, tuple(sorted(remaining)))] = min_cost
        return min_cost

    return dp(0, all_cities - {0})

# Ví dụ
graph = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
print(tsp(graph))  # Output: 80
```

Giải thích:

- Ta sử dụng quy hoạch động kết hợp với quay lui
- Trạng thái dp(curr, remaining) là chi phí nhỏ nhất để đi từ thành phố curr, qua tất cả các thành phố trong remaining và trở về thành phố 0
- Độ phức tạp thời gian: $O(n^2 \cdot 2^n)$

8. Bài toán tô màu đồ thị

python

```python
def graph_coloring(graph, m):
    n = len(graph)
    colors = [0] * n

    def is_valid(vertex, color):
        for i in range(n):
            if graph[vertex][i] == 1 and colors[i] == color:
                return False
        return True

    def backtrack(vertex):
        if vertex == n:
            return True

        for color in range(1, m + 1):
            if is_valid(vertex, color):
                colors[vertex] = color
                if backtrack(vertex + 1):
                    return True
                colors[vertex] = 0

        return False

    if backtrack(0):
        return colors
    return None

# Ví dụ
graph = [
    [0, 1, 1, 1],
    [1, 0, 1, 0],
    [1, 1, 0, 1],
    [1, 0, 1, 0]
]
print(graph_coloring(graph, 3))  # Output: [1, 2, 3, 2]
```

Giải thích:

- Tại mỗi đỉnh, ta thử các màu từ 1 đến m
- Một màu hợp lệ nếu không có đỉnh kề nào có cùng màu
- Độ phức tạp thời gian: O(m^n) trong trường hợp xấu nhất

9. Bài toán cái túi (Knapsack)

python

```python
def knapsack(values, weights, capacity):
    n = len(values)
    memo = {}

    def backtrack(pos, remaining_capacity):
        if pos == n:
            return 0

        if (pos, remaining_capacity) in memo:
            return memo[(pos, remaining_capacity)]

        # Không lấy vật thứ pos
        result = backtrack(pos + 1, remaining_capacity)

        # Lấy vật thứ pos nếu có thể
        if weights[pos] <= remaining_capacity:
            result = max(result, values[pos] + backtrack(pos + 1, remaining_capacity - weights[

        memo[(pos, remaining_capacity)] = result
        return result

    return backtrack(0, capacity)

# Ví dụ
values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print(knapsack(values, weights, capacity))  # Output: 220
```

Giải thích:

- Tại mỗi bước, ta quyết định có lấy vật phẩm hiện tại hay không
- Ta