Lớp: CS112.P11.CTTN
Nhóm: 14
Sinh viên: Lê Nguyễn Anh Khoa. MSSV: 23520742
Sinh viên: Cáp Kim Hải Anh. MSSV: 23520036

BÀI TẬP
Phương pháp thiết kế thuật toán gần đúng

Xây dựng các bước thiết kế thuật toán gần đúng và đưa ra 2 phương pháp gần đúng để giải quyết bài toán:

I. Bài 1: Set Cover:

1. Các bước thiết kế thuật toán gần đúng
- **Bước 1**: Xác định các phần tử chưa được bao phủ trong U.
- **Bước 2**: Chọn tập con S(i) trong S sao cho S(i) bao phủ nhiều phần tử chưa được bao phủ nhất.
- **Bước 3**: Loại bỏ các phần tử đã được bao phủ khỏi U và cập nhật tập S.
- **Bước 4**: Lặp lại Bước 2 cho đến khi tất cả phần tử trong U đều được bao phủ.

2. Thuật toán Tham Lam
- Bước 1: khởi tạo:
  - Tập S’ = ∅ (tập các tập con được chọn)
  - Tập C = U (các phần tử chưa được cover)
- Bước 2: Lặp cho đến khi C = ∅
  - Chọn tập S(i) ∈ S mà |S(i) ∩ C| lớn nhất (tập cover được nhiều phần tử chưa được cover nhất)
  - Thêm S(i) vào S'
  - Cập nhật C = C \ S(i)
- Độ phức tạp của thuật toán: O(|U| × |S|)
- Tỉ lệ xấp xỉ: Thuật toán này cho kết quả với |S'| ≤ ln|U| × OPT, trong đó OPT là số lượng tập con trong lời giải tối ưu.
- Mã giả của thuật toán:

```
GREEDY-SET-COVER(U, S):
    S' = ∅
    C = U
    while C ≠ ∅:
        Chọn Si ∈ S mà |Si ∩ C| lớn nhất
        S' = S' ∪ {Si}
        C = C \ Si
    return S'
```

3. Thuật toán Gần đúng dựa vào trọng số.

- Ý tưởng chính:
- Gán trọng số cho mỗi phần tử trong U
- Chọn tập con có chi phí trên mỗi phần tử mới được cover là nhỏ nhất
- Cập nhật trọng số của các phần tử còn lại sau mỗi lần chọn
- Các bước của thuật toán:
- Bước 1: Khởi tạo:

+ S' = ∅ (tập kết quả)

+ C = U (tập các phần tử chưa được cover)

+ w(e) = 1 cho mọi e ∈ U (trọng số ban đầu của mỗi phần tử)

- Bước 2: Lặp khi C ≠ ∅:

+ Với mỗi S(i) ∈ S, tính chi phí:

$$cost(S(i)) = 1/|S(i) \cap C|$$

+ Chọn tập S(i) có cost(S(i)) nhỏ nhất

+ Thêm S(i) vào S', C = C \ S(i)

+ Với mỗi e ∈ C: w(e) = w(e) × f, với f > 1 là hệ số tăng trọng số

- Độ phức tạp: $O(|U| \times |S| \times \log|U|)$
- Tỉ lệ xấp xỉ: $O(\log|U|)$
- Mã giả:

```
WEIGHTED-SET-COVER(U, S):
    S' = Ø
    C = U
    w(e) = 1 for all e ∈ U
    f = 2  // hệ số tăng trọng số

    while C ≠ Ø:
        For each Si ∈ S:
            cost(Si) = 1/|Si ∩ C|

        Choose Si with minimum cost(Si)
        S' = S' ∪ {Si}
        C = C \ Si

        For each e ∈ C:
            w(e) = w(e) × f

    return S'
```

II. Bài 2: TSP (Travelling Salesman Problem)

1. Các bước thiết kế thuật toán gần đúng

- Bước 1: Khởi tạo một cây khung tối thiểu (Minimum Spanning Tree, MST) của đồ thị
- Bước 2: Sử dụng MST để tạo ra một chu trình xấp xỉ
- Bước 3: Cải thiện chu trình bằng cách kiểm tra và giảm thiểu tổng chi phí

2. Thuật toán Nearest Neighbor

- Ý tưởng:
- • Bắt đầu từ một đỉnh bất kỳ
- • Mỗi bước, chọn đỉnh gần nhất chưa thăm
- • Kết thúc bằng cách quay về đỉnh xuất phát
- Mã giả:

```
NEAREST-NEIGHBOR-TSP(G, start):
    current = start
    path = [start]
    unvisited = V \ {start}

    while unvisited ≠ ∅:
        next = đỉnh v ∈ unvisited có c(current,v) nhỏ nhất
        path.append(next)
        unvisited = unvisited \ {next}
        current = next

    path.append(start)  // quay về điểm xuất phát
    return path
```

- Độ phức tạp: $O(n^2)$
- Tỉ lệ xấp xỉ: $O(\log n)$

3. **Thuật toán Christofides (cho đồ thị đầy đủ thỏa mãn bất đẳng thức tam giác):**

- Ý tưởng:
- Tạo cây khung nhỏ nhất (MST)
- Tìm các đỉnh bậc lẻ trong MST
- Tạo cặp ghép hoàn hảo trọng số nhỏ nhất cho các đỉnh bậc lẻ
- Tạo chu trình Euler từ đồ thị kết hợp
- Chuyển chu trình Euler thành chu trình Hamilton
- Độ phức tạp: $O(n^3)$
- Tỉ lệ xấp xỉ: 1.5 (cho đồ thị metric)
- Mã giả:

```
CHRISTOFIDES-TSP(G):
    // Bước 1: Tạo MST
    T = MINIMUM-SPANNING-TREE(G)

    // Bước 2: Tìm đỉnh bậc lẻ
    O = {v ∈ V | bậc của v trong T là lẻ}

    // Bước 3: Tìm cặp ghép hoàn hảo M cho O
    M = MINIMUM-WEIGHT-PERFECT-MATCHING(G[O])

    // Bước 4: Kết hợp T và M
    H = T ∪ M

    // Bước 5: Tìm chu trình Euler
    E = FIND-EULER-CIRCUIT(H)

    // Bước 6: Chuyển thành chu trình Hamilton
    path = []
    for v in E:
        if v không trong path:
            path.append(v)
    path.append(path[0])

    return path
```