TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM

KHOA KHOA HỌC MÁY TÍNH

Bài tập môn Phân tích và thiết kế thuật toán

Môn học: CS112.P11.KHTN – Phân tích và thiết kế thuật toán

Thực hiện bởi nhóm Nhóm 4, bao gồm:

1. Phan Nhật Tân
2. Nguyễn Huy Phước

Bài làm

1. Huffman Coding:

1.1. Phân tích và xác định độ phức tạp của thuật toán

- Bước khởi tạo:
  - Ta phải khởi tạo n cây cho mỗi ký tự trong bảng chữ cái α, do đó bước này có độ phức tạp O(n), với n là số lượng ký tự trong tập.
- Bước chọn hai cây có xác suất nhỏ nhất:
  - Nếu ta dùng một danh sách để lưu trữ các cây, việc chọn hai cây có xác suất nhỏ nhất sẽ cần duyệt qua toàn bộ danh sách. Việc này tốn thời gian O(n) cho mỗi lần chọn.
  - Vì ta phải thực hiện việc chọn này n-1 lần (vì có n cây và sau mỗi vòng lặp giảm 1 cây), nên tổng thời gian cho việc chọn sẽ là $O(n^2)$ nếu sử dụng danh sách.

1.2. Giải pháp để tối ưu thuật toán:

Tối ưu bằng cách sử dụng Min-Heap:

- Nếu sử dụng Min-Heap để lưu trữ các cây, việc lấy ra hai cây có xác suất nhỏ nhất sẽ có độ phức tạp O(log n) cho mỗi lần.
- Tổng số lần thực hiện việc chọn này là n-1, do đó tổng độ phức tạp của phần này sẽ là O(n log n).

2. Thuật toán Minimum Spanning Tree:

2.1. Prim:

2.1.1. Mã giả và phân tích độ phức tạp của thuật toán:

2.1.1.1 Mã giả:

```
// Input:
// G = (V, E) - Đồ thị với tập đỉnh V và tập cạnh E
// c(e) - trọng số của mỗi cạnh e ∈ E

// Output:
// T - Các cạnh của cây khung nhỏ nhất (MST)

// Pseudocode Prim:

1. function Prim(G, c)
2.     Choose an arbitrary start vertex s ∈ V
3.     X := {s}            // Tập các đỉnh thuộc cây khung
4.     T := ∅              // Tập các cạnh của cây khung (MST)
5.     MinHeap H           // Cấu trúc Min-Heap (Priority Queue) để quản lý các cạnh theo
trọng số
6.
7.     // Thêm tất cả các cạnh (s, v) của đỉnh s vào Heap
8.     for each edge (s, v) ∈ E do
9.         Insert edge (s, v) into H with weight c(s, v)
10.
11.    // Vòng lặp chính của thuật toán
12.    while |X| < |V| do
13.        (u, v) := ExtractMin(H)       // Lấy cạnh (u, v) có trọng số nhỏ nhất
14.        if v ∉ X then                  // Nếu đỉnh v chưa thuộc cây khung
15.            Add vertex v to X          // Thêm v vào cây khung
16.            Add edge (u, v) to T       // Thêm cạnh (u, v) vào cây khung
17.
18.            // Cập nhật Min-Heap với các cạnh kết nối với đỉnh mới v
19.            for each edge (v, w) ∈ E do
20.                if w ∉ X then
21.                    Insert edge (v, w) into H with weight c(v, w)
22.    end while
23.
24.    return T       // Trả về cây khung nhỏ nhất (MST)
```

2.1.1.2. Độ phức tạp:

Giả sử đồ thị đầu vào có:

- n: số đỉnh
- m: số cạnh

Thuật toán Prim có hai phần chính:

- **Khởi tạo:** O(1) do chỉ chọn một đỉnh và khởi tạo tập X và T.
- **Vòng lặp chính:**
    - Ở mỗi bước, việc tìm cạnh có trọng số nhỏ nhất kết nối X với các đỉnh chưa thuộc X có thể được thực hiện với nhiều cách tiếp cận khác nhau, và độ phức tạp của thuật toán sẽ phụ thuộc vào cách này.

Trường hợp sử dụng danh sách liên kết hoặc ma trận trọng số:

- Tìm cạnh có trọng số nhỏ nhất bằng cách duyệt toàn bộ các cạnh có một đỉnh thuộc X và một đỉnh không thuộc X. Với mỗi đỉnh, việc tìm cạnh nhỏ nhất có thể mất thời gian O(m), và tổng cộng sẽ có n-1 lần lặp (vì mỗi lần thêm một đỉnh mới vào X).
- **Độ phức tạp tổng quát:** O(nm), vì mỗi vòng lặp duyệt qua tất cả các cạnh có thể mất O(m) thời gian.

2.1.2. Phương pháp với độ phức tạp O((n+m)log(n)):

Sử dụng Min-Heap để lưu trữ các cạnh với trọng số của chúng, trong đó ta có thể nhanh chóng lấy ra cạnh có trọng số nhỏ nhất và cập nhật các cạnh mới khi thêm các đỉnh mới vào tập X.

Các bước cụ thể để thực hiện tối ưu hóa:

1. **Khởi tạo:**
    - Thêm tất cả các cạnh nối với đỉnh khởi tạo s vào Min-Heap.
2. **Vòng lặp chính:**
    - Ở mỗi vòng lặp, chọn cạnh có trọng số nhỏ nhất từ Min-Heap.
    - Thêm đỉnh tương ứng với cạnh này vào tập X.
    - Cập nhật Min-Heap với các cạnh mới kết nối đỉnh mới thêm vào tập X với các đỉnh chưa thuộc X.
3. **Độ phức tạp:**
    - Với Min-Heap, việc lấy ra cạnh nhỏ nhất tốn O(log n), và việc cập nhật heap tốn O(log n) cho mỗi cạnh.
    - Tổng cộng, ta duyệt n-1 cạnh và cập nhật heap với tối đa m cạnh, do đó độ phức tạp là O((n + m) log n).

2.2. Kruskal:

2.2.1. Mã giả và phân tích độ phức tạp thuật toán:

2.2.1.1. Mã giả:

```
// Input: G = (V, E) là đồ thị, c(e) là trọng số của cạnh e
// Output: T là tập các cạnh của cây khung nhỏ nhất

1. function Prim(G, c)
2.     X := {s}            // Bắt đầu với một đỉnh tùy ý s
3.     T := ∅              // Khởi tạo tập cạnh của cây khung nhỏ nhất
4.
5.     while |X| < n do    // Khi chưa bao phủ hết các đỉnh
6.         Find the smallest edge (u, v) with u ∈ X and v ∉ X   // Tìm cạnh nhỏ nhất với 1
đỉnh trong X và 1 đỉnh ngoài X
7.         Add vertex v to X                                    // Thêm v vào X
8.         Add edge (u, v) to T                                 // Thêm cạnh (u, v) vào cây
khung
9.
10.    return T            // Trả về tập các cạnh của cây khung nhỏ nhất
```

2.2.1.2. Độ phức tạp của thuật toán:

- Mỗi vòng lặp trong thuật toán, ta tìm cạnh có trọng số nhỏ nhất nối từ một đỉnh trong X đến một đỉnh không thuộc X. Việc này tốn **O(n)** thời gian vì phải duyệt qua tất cả các đỉnh.
- Thuật toán phải lặp lại thao tác này cho **n** đỉnh (trừ đỉnh đầu tiên đã chọn), nên tổng thời gian tốn là **$O(n^2)$**.

Độ phức tạp: $O(n^2)$

2.2.2. Thuật toán với độ phức tạp O((n+m)log n)

Pseudocode Kruskal (tối ưu với Union-Find):

```
// Input: G = (V, E) - đồ thị với tập đỉnh V và tập cạnh E
// c(e) - trọng số của mỗi cạnh e ∈ E

// Output:
// T - tập các cạnh của cây khung nhỏ nhất (MST)

// Pseudocode Kruskal:

1. function Kruskal(G, c)
2.     T := ∅                  // Khởi tạo tập cạnh của cây khung nhỏ nhất
3.     Sort all edges in E by increasing weight c(e)   // Sắp xếp các cạnh theo trọng số
tăng dần (O(m log m))
4.     Initialize Union-Find data structure for vertices in V   // Dùng Disjoint Set để
kiểm tra chu trình (O(n))
5.
6.     // Vòng lặp chính
7.     for each edge (u, v) ∈ E (in sorted order) do
```

```
8.          if Find(u) ≠ Find(v) then   // Kiểm tra xem hai đỉnh u và v có thuộc cùng một
tập hay không (O(α(n)))
9.              Add edge (u, v) to T    // Thêm cạnh (u, v) vào cây khung
10.             Union(u, v)             // Hợp nhất hai tập chứa u và v (O(α(n)))
11.
12.     return T   // Trả về tập các cạnh của cây khung nhỏ nhất
```

Độ phức tạp:

- **Bước sắp xếp các cạnh** có độ phức tạp **O(m log m)** (với m là số cạnh), nhưng vì m luôn nhỏ hơn hoặc bằng n^2, nên ta có thể giới hạn thành **O(m log n)**.
- **Union-Find** với path compression và union by rank tốn gần hằng số cho mỗi thao tác, nên kiểm tra và hợp nhất các đỉnh mất **O(α(n))**, với α(n) rất nhỏ. Với mỗi cạnh, ta chỉ mất **O(1)** thời gian để kiểm tra chu trình và hợp nhất các đỉnh.
- Tổng thời gian của phần này là **O(m α(n))**, và có thể coi là **O(m)**.

Tổng độ phức tạp:

- Bước sắp xếp: **O(m log n)**
- Bước Union-Find: **O(m α(n)) ≈ O(m)**

Như vậy, tổng độ phức tạp của thuật toán Kruskal tối ưu sẽ là **O(m log n)**. Khi m là O(n + m), thuật toán này có thể đạt độ phức tạp **O((n + m) log n)**.