ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH THIẾT KẾ THUẬT TOÁN

Bài tập nhóm 6

Nhóm 13: Đỗ Quang Lực - 23520902
Nguyễn Minh Huy - 23520634

Ngày 8 tháng 10 năm 2024

Mục lục

1 Bài 1 **3**

2 Bài 2 **4**
2.1 Câu 1 4
2.2 Câu 2 5

1 Bài 1

```
//init
For each a in α do:
        T_a = tree containing only one node, labeled "a"
        P(T_a) = p_a
        F = {T_a} //invariant: for all T in F, P(T) = ∑_{a∈T} p_α
//main loop
While length(F) >= 2 do
        T_1 = tree with smallest P(T)
        T_2 = tree with second smallest P(T)
        remove T1 and T2 from F
        T_3 = merger of T1 and T2
        // root of T1 and T2 is left, right children of T3
        P(T_3) = P(T_1) + P(T_2)
        add T_3 to F
Return F[0]
```

Câu 1: Hãy phân tích và xác định độ phức tạp của thuật toán trên

Đây là thuật toán Huffman Coding, có hai phần chính đó là

- Phần khởi tạo: với mỗi kí tự a trong bảng chữ cái α, tạo một cây T_α chứa duy nhất một nút và nhãn a cùng với chi phí p_a. Độ phức tạp ở bước này bằng với số kí tự trong bảng chữ cái α (tạm gọi là n) nên sẽ có độ phức tạp là O(n)
- Vòng lặp chính: trong mỗi lần lặp, hai cây có chi phí nhỏ nhất được chọn và hợp nhất lại thành một cây. Sau đó thêm cây mới này vào trong danh sách. Vì có n cây nên sẽ có $n-1$ bước lặp. Độ phức tạp ở bước này sẽ là O(n * (chi phí chọn được hai cây có chi phí nhỏ nhất))

Vậy, tổng độ phức tạp là là O(n * (chi phí chọn hai cây có chi phí nhỏ nhất))

Câu 2: Hãy đưa ra một giải pháp để tối ưu thuật toán trên.

Ở thuật toán trên, chi phí để chọn được hai cây có chi phí nhỏ nhất ta vẫn chưa xác định được. Có thể nghĩ ngay đến việc duyệt qua hết tất cả các chi phí của các cây rồi chọn ra được 2 cây có chi phí nhỏ nhất. Độ phức tạp cho bước này sẽ là O(n), và độ phức tạp cho thuật toán trên sẽ là $O(n^2)$

Tuy nhiên, ở đây chúng ta có thể sử dụng một cấu trúc dữ liệu đã được học đó là heap (hay còn gọi là priority_queue) - là một cấu trúc dữ liệu cho phép chúng ta thêm vào một phần tử và lấy ra phần tử nhỏ nhất trong độ phức tạp O(log(n)). Ta có một thuật toán với độ phức tạp tốt hơn.

2 Bài 2

2.1 Câu 1

Prim

Input: connected undirected graph $G = (V, E)$ in adjacency-list representation and a cost c_e for each edge $e \in E$.
Output: the edges of a minimum spanning tree of G.

// Initialization
$X := \{s\}$ // s is an arbitrarily chosen vertex
$T := \emptyset$ // invariant: the edges in T span X
// Main loop
while there is an edge (v, w) with $v \in X, w \notin X$ **do**
    $(v^*, w^*) :=$ a minimum-cost such edge
    add vertex w^* to X
    add edge (v^*, w^*) to T
return T

Yêu cầu: hãy đưa ra mã giả chi tiết cho thuật toán trên và phân tích độ phức tạp. Hãy đề xuất một phương pháp để được độ phức tạp O((|V| + |E|)log(|V|))
Mã giả:

```
def Prim(G):
    //Chọn một đỉnh bất kỳ s trong G
    T = {}  // Tập cạnh của cây khung
    X = {s}  // Tập đỉnh đã được chọn

    while (|X| < |V|):  // Lặp đến khi tất cả đỉnh đều được chọn
        (v*, w*) = cạnh có chi phí nhỏ nhất với v* thuộc X và w* không thuộc X
        T = T u {(v*, w*)}  // Thêm cạnh vào cây khung
        X = X u {w*}  // Thêm đỉnh w* vào tập X

    return T
```

Vì mỗi lần ta thêm 1 đỉnh vào tập X, nên vòng lặp while sẽ lặp |V| lần. Ở bước tìm cạnh ta dùng ý tưởng của priority_queue ở bài 1, khi đó chúng ta chỉ cần tìm cạnh đó trong độ phức tạp O(log|E|). Vậy độ phức tạp của thuật toán sẽ là O(|V|log(|E|)). Độ phức tạp đưa ra khá giống với yêu cầu.

2.2 Câu 2

Kruskal

Input: connected undirected graph $G = (V, E)$ in adjacency-list representation and a cost c_e for each edge $e \in E$.
Output: the edges of a minimum spanning tree of G.

// Preprocessing
$T := \emptyset$
sort edges of E by cost // e.g., using MergeSort[26]
// Main loop
for each $e \in E$, in nondecreasing order of cost **do**
    if $T \cup \{e\}$ is acyclic **then**
        $T := T \cup \{e\}$
return T

Yêu cầu: hãy đưa ra mã giả chi tiết cho thuật toán trên và phân tích độ phức tạp. Hãy đề xuất một phương pháp để được độ phức tạp O((|V| + |E|)log(|V|))
Mã giả:

```
def Kruskal(G):
    T = {}  // Tập cạnh của cây khung nhỏ nhất
    Sắp xếp các cạnh e thuộc E theo thứ tự không giảm của chi phí c_e

    for mỗi cạnh e trong E theo thứ tự đã sắp xếp:
        if T u {e} không tạo chu trình:
            T = T u {e}  // Thêm cạnh e vào cây khung

    return T  // Trả về cây khung nhỏ nhất
```

Thuật toán này có hai bước:

- Sắp xếp các cạnh e thuộc E theo thứ tự không giảm: có thể sử dụng các thuật toán sắp xếp tốt hiện nay như quick sort, merge sort với độ phức tạp O(|E|log(|E|)).
- Duyệt qua từng cạnh và thêm nó vào tập cạnh của cây khung nếu không tạo ra chu trình: bước duyệt với độ phức tạp O(|E|), ở phần kiểm tra chu trình ta có thể sử dụng cấu trúc DisjointSet: cho phép kiểm tra chu trình trong độ phức tạp O($\alpha(|E|)$), với $\alpha(|E|)$ là hàm Ackermann nghịch đảo, có giá trị rất nhỏ (thông thường nhỏ hơn cả log(|E|)). Nên bước này có độ phức tạp là O(|E|$\alpha(|E|)$).

Vậy độ phức tạp của thuật toán sẽ là O(|E|log(|E|)).