Phân tích và thiết kế thuật toán - CS112.Q11.KHTN - Solution bài tập về nhà từ nhóm 10 (Nhóm 4 thực hiện)

Sinh viên nhóm 4 thực hiện:

- Bảo Quý Định Tân - 24520028
- Lê Văn Thức - 24521748

Ở tất cả subtask ta có: $0 \leq c_i \leq m, 0 \leq w_i \leq m, p_i \leq i$.

Subtask 1: $n \leq 18$

Ở subtask này ta có thể duyệt trâu tất cả các trạng thái để quyết định xem ở mỗi nút trên cây ta có nên xóa hay không. Số lần duyệt qua các trạng thái là khoảng 2^n.

```
int ans = 0;
for (int mask = 0; mask < (1 << n); mask++) {
    if (mask & 1) continue;
    for (int i = 0; i < n; i++) {
        isDeleted[i + 1] = (mask >> i) & 1;
    }
    check = true;
    dfs(1);
}
```

Sau đó ta tốn thêm khoảng n thao tác nữa để kiểm tra xem cách lựa chọn này có hợp lệ hay không rồi cập nhật vào đáp án.

```
int dfs(int u) {
    int currStress = w[u];
    for (int v : adj[u]) {
        int tmp = dfs(v);
        currStress++;
        if (isDeleted[v]) {
            currStress += tmp - 1;
        }
    }
    if (currStress > m) {
        check = false;
    }
```

```
    return currStress;
}
```

Xác định độ phức tạp thời gian:

1. Tham số đầu vào: n
2. Phép toán cơ sở: 1 lần gọi hàm dfs là 1 phép toán cơ sở
3. Đếm số phép toán cơ sở: vòng for các $mask$ được chạy qua tất cả 2^n $mask$, thực hiện phép toán cơ sở 2^n lần
4. Tổng hợp lại bỏ qua hằng số và bậc thấp hơn ta có độ phức tạp là $\Theta(2^n \times n)$ hay $O(2^n \times n)$
   Độ phức tạp không gian: $O(n)$.

Subtask 2: $n \leq 676767, m \leq 369$

Bắt đầu từ đây ta không thể làm bừa được nữa. Ta bắt đầu với nhận xét rằng ở mỗi nút u, với một cách xóa nào đó ở gốc cây con của u, u sẽ truyền lên trên nút cha một giá trị là tổng của các w_j bị xóa và số lượng nút con trực tiếp của u hay còn gọi là C_u (sau khi thực hiện một vài thao tác xóa nào đó trong cây con gốc u).

Ta gọi con của u là $v_1, v_2, \ldots, v_k$.

Ta thấy rằng khi bỏ đi một nút v_i nào đó giống như việc u sẽ gánh thêm phần stress C_{v_i}.

Và với nhận xét rằng C_u phải $\leq m \leq 369$, giới hạn của m ở subtask này khá nhỏ nên ta có thể một hướng giải bằng quy hoạch động.

Ta gọi $dp[u][c] =$ số người tối đa có thể sa thải trong cây con gốc nếu sau khi sa thải, $C_u = c$ (ban đầu khởi tạo bằng $-inf$ hết).

Ban đầu, khi u là nút lá, ta có: $dp[u][w_u] = 0$.

Nếu u không phải nút lá, ta có hai lựa chọn cho mỗi con của u:

1. Giữ lại : Khi này chịu thêm $+1$ vì có thêm một con trực tiếp.
   $maximize(dp[u][c_u + 1], dp[u][c_u + 1])$
2. Xóa đi v_i: Khi này ta phải duyệt qua các giá trị của C_{v_i} để cập nhật trạng thái, giá trị mới của C_u là $C_u + C_{v_i}$. $maximize(dp[u][c_u + c_v], dp[u][c_u] + 1)$.

- Với tham số đầu vào là n và m độ phức tạp là $O(n \times m^2)$ với phép toán cơ bản là thực hiện tính toán khi quy hoạch động và ở mỗi nút phải thực hiện lần nhân thêm m lần cho mỗi nút con và chỉ có tối đa n nút con thôi. Mình nghĩ là vẫn chưa tối ưu cho subtask này :(nhưng đây là cách tốt nhất mình có thể nghĩ ra.
- Độ phức tạp không gian là $O(n \times m)$ tại vì ta phải khai báo mảng đệ quy 2 chiều $n \times m$.

Subtask 3: $n \leq 696969, m \leq 36363636$**.**

Ở subtask này chắc chắn hướng giải quy hoạch động kia sẽ không thực hiện được, nên ta có thể nghĩ đến một hướng mới là tham lam.

Vì khi xét riêng một nút, ta thấy nếu giả sử trong các cây con của các nút con trực tiếp đã được xóa một cách tối ưu. Thì bây giờ ta sẽ chỉ đơn giản là chọn những nút con trực tiếp nào mà khi xóa, lượng stress nhận được là nhỏ nhất có thể cho tới khi không thể xóa được nữa thôi.

Còn một vấn đề nữa xảy ra là chắc gì việc cho các cây con của các nút con trực tiếp v_i xóa trước sẽ tối ưu hơn? Tại vì thường xóa tối ưu sẽ đưa mức độ stress của nút gốc cây con v_i đó lên đến gần m lận, việc này khiến cho sau này các nút cha u khó lòng gánh được stress của nút này =)). Thì ta cũng dễ thấy rằng nếu nút con v_i đó không xóa đến hết khả năng để các nút cha u có thể gánh được stress của nó khi xóa nó đi thì ta huề vốn (dễ thấy), thậm chí là lỗ. Trong khi ta có lẽ xóa được hơn chắc chắn ≥ 1 khi để nút con trực tiếp v_i này gánh stress từ việc xóa đi một vài nút trong cây con của nó và nhường phần slot chịu được stress của nút u hiện tại cho cây con trực tiếp v_i khác <(").

Minh chứng ở trên cũng chứng minh được rằng việc tối ưu ở bài toán cục bộ không làm ảnh hưởng đến kết quả của kết quả toàn cục. Là điểm lý tưởng để ta thực hiện tham lam.

```
int dfs(int u) {
    int currStress = w[u];
    priority_queue<int, vector<int>, greater<int>> pq;

    for (int v : adj[u]) {
        currStress++;
        int childStress = dfs(v);
        pq.push(childStress - 1);
    }

    if (currStress > m) {
        cout << 0;
```

```
            exit(0);
        }

        while (!pq.empty() && currStress <= m) {
            int v = pq.top();

            if (currStress + v <= m) {
                currStress += v;
                pq.pop();
                ans++;
            }
            else {
                break;
            }
        }

        return currStress;
    }
```

- Với tham số đầu vào là n, phép toán cơ bản là việc tính giá trị tối ưu theo hướng tham lam ở mỗi nút, mỗi thao tác tốn $O(\log n)$ (tại vì ta sử dụng heap) nên tổng độ phức tạp sẽ là $O(n \log n)$.
- Độ phức tạp không gian là $O(n)$ tại vì mảng, heap, stack đệ quy tối đa cũng chỉ n phần tử.