Lời giải bài tập tham lam nhóm 10

Đề bài: https://github.com/lenhanbo/CS112_BTVN

Tóm tắt bài toán:

Ta có:

- Một cây công ty gồm n người (gốc là CEO – nhân viên 1).
- Mỗi người i có: Khối lượng công việc $w[i]$ và một tập các nhân viên con trực tiếp.
- Chỉ số phức tạp công việc của nhân viên thứ i:

$$C_i = w_i + \sum_{j \in con(i)} w_j$$

Nếu sa thải nhân viên x:

- Toàn bộ nhân viên mà x quản lý sẽ được chuyển lên quản lý trực tiếp bởi cha của x.
- Khối lượng công việc $w[x]$ cũng cộng thêm vào cha của x.

Ràng buộc:

- Tại mọi thời điểm (trước và sau sa thải), chỉ số phức tạp của mọi nhân viên $\leq m$.
- Không thể sa thải người 1.

Subtask 1 (n $\leq$ 18): brute-force

Vì n nhỏ, ta có thể duyệt tất cả các tập con các node bị sa thải (ngoại trừ root). Với mỗi tập con:

- Lập parent hiệu dụng: $parent'[i]$ = tổ tiên gần nhất không bị sa thải (= 0 nếu root).
- Tính $C[u]$ cho tất cả node không bị sa thải bằng cách cộng $w[child]$ vào cha hiệu dụng.
- Nếu mọi $C[u] \leq m$ thì cập nhật đáp án $ans = \max(ans, \text{size(tập sa thải)})$.

Độ phức tạp: $O(2^{(n-1)} \times n)$.

Subtask 2: Quy hoạch động

Với mỗi nút u, ta định nghĩa $dp[u][s]$ = số lượng tối đa nhân viên bị sa thải trong cây con của u sao cho lượng khối lượng chuyển lên $parent(u) = s$ (gọi $sum\_up(u) = s$).

Ta chỉ lưu s trong khoảng $0..m$ (m lớn nhất mà parent cần quan tâm). Nếu trạng thái s không thể đạt được thì $dp[u][s] = -\infty$ (hoặc -1).

- Khi u không bị sa thải, $sum\_up(u) = w[u]$.
- Khi u bị sa thải, $sum\_up(u) = w[u] + S_{\text{children}}$ (với S_{children} là tổng các sum_up mà các con chuyển lên u).

Ta gộp các con của u bằng phép convolution kiểu knapsack: tổng $S_{\text{children}} = \sum s_{v_i}$ với mỗi con chọn s_{v_i} từ $dp[v_i][*]$.

Sau khi gộp con, ta tạo $dp[u][*]$ bằng hai lựa chọn:

- Giữ u: nếu $w[u] + S_{\text{children}} \leq m$ thì cập nhật

$$dp[u][w[u]] = \max(dp[u][w[u]], \sum dp[v_i][s_{v_i}])$$

- Sa thải u (nếu $u \neq 1$): nếu $w[u] + S_{\text{children}} \leq m$ thì cập nhật

$$dp[u][w[u] + S_{\text{children}}] = \max(dp[u][w[u] + S_{\text{children}}], \sum dp[v_i][s_{v_i}] + 1)$$

Chú ý: đối với lá, $S_{\text{children}} = 0$.

Độ phức tạp: $O(n \times m^2)$.

Subtask 3: Tham lam

Khi sa thải v (v là con của u):

- u mất một cấp dưới, nên $c_u \leftarrow c_u - 1$.
- u nhận thêm toàn bộ công việc của v, nên $w_u \leftarrow w_u + w_v + c_v$ (vì $w_v + c_v$ thể hiện tổng tải mà v đang chịu).

Sau khi sa thải v, độ phức tạp của u sẽ tăng thêm:

$$\Delta = w_v + c_v - 1$$

Để u không vượt giới hạn m, ta cần:

$$w_u + c_u + (w_v + c_v - 1) \leq m$$

Tại node u, ta có thể thử sa thải từng con, sao cho tổng độ phức tạp vẫn $\leq m$.

Bài toán này tương tự như: Có k món đồ (mỗi món có giá $= w_v + c_v - 1$), ta muốn chọn nhiều nhất số món có tổng giá $\leq$ một giới hạn cho trước.
Đây là bài toán Knapsack 0–1 đặc biệt, trong đó:

- Lợi ích mỗi món $= 1$.
- Giá trị $= w_v + c_v - 1$.
- Giới hạn $= m - (w_u + c_u)$.

Bài toán Knapsack nói trên nếu làm bằng quy hoạch động sẽ rất chậm $O(n \cdot m)$, nhưng ta nhận thấy lợi ích mỗi món đều bằng nhau (1).
$\rightarrow$ Trong trường hợp này, chọn các món rẻ nhất trước luôn là tối ưu.
Lý do:
Nếu chọn món đắt hơn, ta sẽ nhanh hết giới hạn mà không tăng số món được chọn.
Nên Greedy theo chi phí tăng dần là chính xác.
Vì vậy, tại mỗi node u:

- Gọi DFS xử lý xong tất cả cây con.
- Với mỗi con v, ta biết được "tổng độ phức tạp riêng" $w[v] + c[v]$.
- Sắp xếp các con theo $(w[v] + c[v])$ tăng dần.
- Duyệt theo thứ tự này, thử sa thải từng người:
  + Nếu việc sa thải v không khiến $C_u > m$, ta sa thải và cập nhật w_u, c_u, ans.
  + Nếu không, bỏ qua (vì những người sau còn nặng hơn, không thể sa thải được).

Độ phức tạp: $O(nlogn)$.
Code:

```
1
2  /*====================================
3  Loi giai: Tham lam (Greedy)     Subtask 3
4  ====================================*/
5
6  import sys
7  sys.setrecursionlimit(10**7)
8
9  n, m = map(int, input().split())
10 w = [0] + list(map(int, input().split()))
11 c = [0] * (n + 1)
12 adj = [[] for _ in range(n + 1)]
```

```
13
14 for i in range(1, n):
15     x = int(input())
16     c[x] += 1
17     adj[x].append(i + 1)
18
19 ans = 0
20
21 def cmp_key(v):
22     return w[v] + c[v]
23
24 def dfs(u):
25     global ans
26     if not adj[u]:
27         return
28     for v in adj[u]:
29         dfs(v)
30     adj[u].sort(key=cmp_key)
31     for v in adj[u]:
32         if w[u] + c[u] + w[v] + c[v] - 1 <= m:
33             w[u] += w[v] + c[v]
34             c[u] -= 1
35             ans += 1
36 dfs(1)
37 print(ans)
```