Hướng dẫn giải Code Challenge #7

Big-O & VNG - 29/11/2022

BÀI A: NERV Tree

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Subtask1

Do đỉnh a và b là leaf node nên để kiểm tra 1 node c có nằm trên đường đi a và b ta chỉ cần kiểm tra $D(a,b) = D(a,c) + D(c,b)$ với $D(u,v)$ là độ dài đường đi từ u tới v. Để tính nhanh có thể dùng **LCA** để tính.

Subtask2

Để xác định c có nằm trên đường đi dài nhất **đi qua** a và b. Nhận thấy đường đi dài nhất đi qua đỉnh a, b, c phải có độ dài bằng đường đi dài nhất đi qua a và b.
Bài toán đầu tiên cần giải là với 3 đỉnh a, b, c hãy tìm 2 đỉnh endpoints hoặc 3 đỉnh đó không tạo thành đường đi đơn. Ta có các trường hợp như sau:

- $D(a,b) + D(b,c) = D(a,c)$ thì 2 đỉnh endpoints là a, c.
- $D(a,c) + D(b,c) = D(a,b)$ thì 2 đỉnh endpoints là a, b.
- $D(b,a) + D(a,c) = D(b,c)$ thì 2 đỉnh endpoints là b, c.
- Nếu không thuộc các trường hợp trên thì a, b, c không tạo thành đường đi đơn.

Bài toán thứ hai là với 2 đỉnh a b, hãy tìm độ dài đường đi dài nhất đi qua 2 đỉnh đó. Gọi p là cha con chung gần nhất của a và b (**LCA**). Giả sử $h(a) < h(b)$, với h là độ cao trên cây. Có các trường hợp sau:

- $p = a$: Đô dài đường đi dài nhất là $dp_{down}(b) + D(a,b) + dp_{up}(a,b)$
- Ngược lại thì là $dp_{down}(b) + D(a,b) + dp_{down}(a)$

Với $dp_{down}(u) = \max(D(u,k)) \; \forall k \in \text{subtree}(u)$
$dp_{up}(u,v) = \max(D(u,k)) \; \forall k \notin \text{subtree}(t_1)$ với $\{u, t_1, t_2, ..., v\}$ là đường đi từ u đến v.

Để tính dp_{down} ta có thể dùng DFS kết hợp với quy hoạch động.
Để tính dp_{up} ta có thể sử dụng kết quả của dp_{down} trên, lưu đường đi max1, max2 của từng đỉnh.

Time complexity: $O(N + Q \log N)$

Mã nguồn tham khảo

- subtask1: https://ideone.com/dqdrl0
- subtask2: https://ideone.com/q4irP7
- subtask3: https://ideone.com/PgyhVP

BÀI B: Two Rect

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Ý tưởng

Việc chọn ra hai hình chữ nhật (HCN) toàn 0 mà không giao nhau thì chỉ có 2 trường hợp: Hai HCN tách nhau theo chiều ngang hoặc dọc:

Ở đây sẽ hướng dẫn xử lý trường hợp bên trái, sau đó xoay ma trận 90 độ là có thể tận dụng lại hàm xử lý để tìm đáp án cho trường hợp bên phải. Kết quả cuối cùng là đáp án lớn nhất trong cả hai trường hợp.

Để giải quyết trường hợp bên trái, ta có thể dùng quy hoạch động như sau:

- Với $1 \leq i \leq M$:
  - Gọi `dp1[i]` là diện tích HCN toàn 0 lớn nhất mà chỉ số hàng của cạnh dưới nhỏ hơn hoặc bằng `i`
  - Gọi `dp2[i]` là diện tích HCN toàn 0 lớn nhất mà chỉ số hàng của cạnh trên lớn hơn hoặc bằng `i`
- Khi đó, kết quả tối ưu là giá trị lớn nhất của `dp1[i] + dp2[i + 1]` với mọi $0 \leq i \leq M$

  (Với `dp1[0] = dp2[M+1] = 0`)

Tính `dp1`

(Để tính `dp2` ta cũng áp dụng ý tưởng tương tự dưới đây)

Nhắc lại, `dp1[i]` là diện tích HCN lớn nhất mà chỉ số hàng của cạnh dưới nhỏ hơn hoặc bằng `i`, ta dùng quy hoạch động với mã giả đơn giản như sau:

```
dp1[0] = 0
for i := 1 to M:
    maxRect := <Diện tích HCN toàn 0 lớn nhất mà chỉ số hàng của cạnh dưới bằng i>
    dp1[i] := max(dp1[i - 1], maxRect)
```

Tìm `maxRect` ở mã giả trên như thế nào?

Trước khi quy hoạch động tính `dp1`, ta cần khởi tạo mảng 2 chiều `h` với `h[i][j]` là "độ cao" của cột toàn 0 kết thúc tại ô (i, j), mã giả như sau:

```
for j := 1 to N:
    h[0][j] = 0
for i := 1 to M:
    for j := 1 to N:
        if (a[i][j] == 0):
            h[i][j] = h[i - 1][j] + 1
        else:
            h[i][j] = 0
```

Xét ví dụ sau, giả sử ta muốn tính HCN toàn 0 lớn nhất mà cạnh dưới là hàng thứ 5 ($i = 5$):

Ta có `h[5][1], ..., h[5][7]` là `[4, 1, 2, 1, 4, 5, 4]`

Tiếp theo, với mọi cột `j`, ta sẽ tìm `L[j]`, `R[j]` sao cho:

- $L[j] \leq j \leq R[j]$
- `h[i][j]` là min của đoạn `h[i][L[j]] .. h[i][R[j]]`
- `L[j]` nhỏ nhất có thể, `R[j]` lớn nhất có thể

Cuối cùng, `maxRect` tại hàng `i` là giá trị lớn nhất của `h[i][j] * (R[j] - L[j] + 1)` với mọi cột `j`

Ở ví dụ trên, ta có:

- `L[1], ..., L[7]` là `[1, 1, 3, 1, 5, 6, 5]`
- `R[1], ..., R[7]` là `[1, 7, 3, 7, 7, 6, 7]`
- `maxRect = h[5][7] * (R[7] - L[7] + 1) = 4 * (7 - 5 + 1) = 12`

Để tìm `L[], R[]` trong độ phức tạp thời gian $O(N)$, tham khảo tại đây.

Độ phức tạp

Không gian: $O(M \times N)$

Thời gian:

- Bước khởi tạo để tính mảng `h` là $O(M \times N)$
- Mỗi lần tìm `maxRect` mất $O(N)$, do đó tính `dp1` mất $O(M \times N)$

Mã nguồn tham khảo

- C++: https://ideone.com/ZM75vi

BÀI C: Scary Night

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Chuỗi S sẽ là lộ trình hợp lệ nếu số ký tự F bằng B, và R bằng L.

Ta sử dụng segment tree để đếm số lượng từng kí tự F, R, B, L trong chuỗi S. Như vậy, mỗi node (left, right, index) trên segment tree sẽ chứa 4 giá trị đếm khác nhau, tương ứng với số lượng ký tự F, R, B, L trong đoạn S[left:right].

Ngoài ra, vì bài toán có yêu cầu cập nhật một đoạn trong chuỗi, ta sẽ sử dụng lazy propagation. Tuy nhiên mỗi node chỉ có 1 giá trị lazy duy nhất, chính là số lần mà tất cả các kí tự trong đoạn đó cần "xoay". Cách xoay đoạn (left, right) một góc k * 90 độ như sau:

- Giả sử old_cnt[4] lần lượt là biến đếm số ký tự F, R, B, L của đoạn này trước khi xoay (lưu ý rằng 4 ký tự được liệt kê theo chiều kim đồng hồ).
- Ta nhận thấy rằng mỗi kí tự khi xoay 90 độ sẽ trở thành ký tự tiếp theo (F thành R, R thành B,...). Như vậy nếu xoay k * 90 thì sẽ thành ký tự cách nó k vị trí. Đồng nghĩa với việc sau khi xoay, biến đếm của một ký tự sẽ bằng với biến đếm của ký tự trước nó k vị trí.
- Vậy ta sẽ có new_cnt[(i + k) % 4] = old_cnt[i].
- Đồng thời ta nhận xét rằng nếu k = 0 thì việc xoay sẽ không thay đổi biến đếm.

Sau khi khởi tạo segment tree trên chuỗi S ban đầu, ta lần lượt đọc vào Q truy vấn:

- Truy vấn 1 i c: Ta sẽ di chuyển từ root xuống node lá tương ứng với vị trí i cần cập nhật, sau đó gán biến đếm của ký tự c bằng 1, và của các ký tự còn lại bằng 0.
- Truy vấn 2 i j k: Ta thực hiện thao tác cập nhật đoạn theo lazy propagation. Mỗi khi đi qua một node nào đó, ta cần cập nhật giá trị lazy vào cây (bằng cách "xoay" các biến đếm). Đối với truy vấn loại 1, khi ta ở node (left, right) với 2 node con (left, mid) và (mid+1, right), mặc dù giá trị i chắc chắn chỉ nằm ở 1 trong 2 nhánh, nhưng ta cần gọi đệ quy cả 2 nhánh này nhằm giúp giá trị lazy ở 2 node con đều được cập nhật vào cây (vì sau đó, ta cần sử dụng giá trị ở 2 node con này để tính toán lại biến đếm của node hiện tại).

Sau mỗi truy vấn, ta sử dụng 4 biến đếm của note root để kiểm tra nếu F = B và R = L. Nếu có, ta sẽ tăng giá trị biến đếm số chuỗi hợp lệ.

##Độ phức tạp: O(N + Qlog(N))

Mã nguồn tham khảo

- C++: https://ideone.com/ngvAHs

BÀI D: An Esoteric Programming Language

Tóm tắt đề bài:

Xem tại đây

Hướng dẫn giải

Bước 1: Đọc vào chuỗi và chỉ lưu trữ lại các câu lệnh ><[].+-

Bước 2: Duyệt qua các câu lệnh và lấy ra các câu lệnh [ ] và lưu giữ index tương ứng của các vòng lặp dưới dạng hashmap (key, value) - (vị trí của [, vị trí của] tương ứng) và ngược lại. Làm vậy để việc truy xuất vị trí câu lệnh bắt đầu vòng lặp, kết thúc vòng lặp nhanh hơn

Bước 3: Duyệt qua từng câu lệnh và thực hiện câu lệnh đó.

Độ phức tạp thuật toán: O(n)

Mã nguồn tham khảo

- Python: https://ideone.com/o3IXKa