CHỦ ĐỀ: PHÂN TÍCH ĐỘ PHỨC TẠP THUẬT TOÁN

NHÓM 5:

Nguyễn Quốc Huy Hoàng – 20520051

Lê Nguyễn Khánh Nam – 20520073

Câu 1. Phân tích độ phức tạp thời gian thuật toán quickselect.

Trả lời:

Quick select là thuật toán tìm ra số lớn thứ k trong mảng có n phần tử.

Thuật toán của quick select tìm số lớn thứ k trong khoảng từ left đến right của mảng arr :

Bước 1 : Gọi hàm partition : Lấy chỉ số left làm pivot (có thể lấy bất kì cũng được) , sắp xếp các phần tử của mảng từ left đến right sao cho chia làm 2 phần, một phần nhỏ hơn a[pivot], phần còn lại lớn hơn a[pivot], sau đó trả về index bằng vị trí của a[pivot] hiện tại. (Thuật có độ phức tạp $O(n)$).

Bước 2 : Nếu index – left > k – 1, right = index – 1, quay lại bước 1.

Nếu index – left < k – 1, left = index + 1 , k = k - index + l – 1, quay lại

bước 1.

Bước 3 : Trả về arr[index] .

Mỗi lần thực hiện quick select ta đều gọi hàm partition với độ phức tạp $O(n)$, tìm được chỉ số mà phân mảng thành 2 phần, sau đó gọi đệ quy hàm quick select trên một phần nào đó, về mặt trung bình thì quick select lúc sau sẽ xử lí trên một nửa phần tử còn lại. Nói chung, ta sẽ có độ phức tạp thời gian của quick select được tính toán như sau :

$$
\begin{aligned}
T(n) &= n + T\left(\frac{n}{2}\right) \\
&= n + \frac{n}{2} + T\left(\frac{n}{4}\right) \\
&= n + \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{\log_2(n)}} \\
&= n\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{\log_2(n)}}\right) \\
&< n\left(\frac{1}{1-\frac{1}{2}}\right) = 2n
\end{aligned}
$$

Do đó độ phức tạp trung bình của quick select là $O(n)$, nhưng độ phức tạp của thuật toán này phụ thuộc rất nhiều vào việc chọn pivot, sao cho số phần tử cần phải xử lý sau này giảm theo một tỉ lệ nào đó, nếu xui xẻo khi liên tục chọn pivot mà chỉ giảm số phần tử cần xử lý đi một, thì thuật toán có thể dẫn đến độ phức tạp $O(n^2)$ (cụ thể trong trường hợp arr[pivot] là nhỏ nhất trong khoảng left đến right, nếu tiếp tục tìm kiếm ở phía bên phải mảng thì ta chỉ giảm số lượng phần tử cần tìm kiếm đi 1 phần tử, do đó có thể mất n lần gọi hàm partition ta mới tìm ra được kết quả, mà hàm partition lại có độ phức tạp $O(n)$ nên độ phức tạp của thuật toán trong trường hợp xấu nhất có thể dẫn tới $O(n^2)$).

Câu 2. Hãy dùng phương pháp potential để tính độ phức tạp của hàm thêm vào trong cấu trúc dữ liệu vector.

Trả lời:

Đầu tiên, để tính được độ phức tạp của hàm push_back trong cấu trúc dữ liệu vector, ta cần biết thuật toán của nó, trong vector ngoài lưu trữ dữ liệu dạng danh sách, thì nó còn chứa 2 thuộc tính là size (số phần tử hiện có trong vector) và capacity (số lượng phần tử lớn nhất mà vector có thể lưu trữ được hiện tại) kèm theo đó là phương thức reserve (int) để định giá trị capacity cho vector (độ phức tạp $O(size)$), ta có được thuật toán của hàm push_back (value), thêm dữ liệu value vào cuối vector :

- Bước 1 : Nếu capacity = 0 , gọi hàm reverse(1) (tăng capacity từ 0 lên 1) .
- Bước 2 : Nếu size = capacity, gọi hàm reserse(2*capacity) (tăng capacity gấp đôi) .
- Bước 3 : Gán value vào ô nhớ thứ size , tăng size lên 1 .

Ta định nghĩa hàm potential như sau :

$\Phi = 2*size - capacity$

Gọi c_i là chi phí cho lần push_back thứ i.

Nếu push_back không làm tăng capacity :

$c_i = 1 + \Phi_i - \Phi_{(i-1)}$

$= 1 + (2*(size+1) – capacity) – (2*size – capacity)$

$= 3$

Nếu push_back làm tăng capacity :

$capacity_i = 2*capacity_{i-1}$, $size_{i-1} = capacity_{i-1}$,$size_i = capacity_{i-1} + 1$

$c_i = capacity_{i-1} + 1 + \Phi_i - \Phi_{i-1}$

$= capacity_{i-1} + 1 +(2*(capacity_{i-1}+ 1) – 2*capacity_{i-1})–(2*capacity_{i-1} - capacity_{i-1})$

$= 3$

Vậy trung bình hàm push_back trong vector hao phí thời gian là hằng số, độ phức tạp là $O(1)$.

Câu 3. Các bạn hẳn còn nhớ bài toán 8 hậu, rằng tìm một cách đặt 8 quân hậu sao cho không con nào ăn được nhau. Nếu thuật toán của của ta có các bước như sau:

- Bước 1: Khởi tạo bàn cờ.
- Bước 2: Nếu đã đặt đủ quân hậu, xuất đáp án.
- Bước 3: Tìm tất cả vị trí trong hàng có để đặt được quân hậu sao cho không quân hậu nào ăn được quân hậu khác.
- Bước 4: Nếu không có vị trí nào thỏa mãn, đi đến bước 1.
- Bước 5: Nếu có, chọn ngẫu nhiên một vị trí, di chuyển đến hàng tiếp theo và quay lại bước 3.

Thì kỳ vọng bạn sẽ phải khởi tạo bao nhiêu bàn cờ?

Trả lời:

Nhận xét đề bài:

- Bước 1: Khởi tạo bàn cờ .
  ⇨ Bàn cờ kích thước 8 x 8 ở **trạng thái rỗng**
- Bước 2: Nếu đã đặt đủ quân hậu, xuất đáp án .
  ⇨ Vì ở **bước 1 bàn cỡ rỗng** nên bước 2 **không thể đặt đủ quân hậu để xuất đáp án** được.
- Bước 4: : Nếu không có vị trí nào thỏa mãn, đi đến bước 1.
  ⇨ Đến bước này ta nhận ra một điều là thuật toán này **sẽ lặp vô tận**, bởi vì khi thực hiện từ *Bước 1* đến *Bước 4*, khi gặp phải **trường hợp không thõa mãn đầu tiên** thì sẽ bị quay lại *Bước 1*, lúc này bàn cờ bị reset về rỗng. Rồi thực thi tiếp *Bước 2, Bước 3* rồi đến *Bước 4* gặp phải trường hợp không thõa mãn rồi lại quay về *Bước 1*. Cứ như thế sẽ lặp thành **vòng lặp vô tận:**

  Bước 1 -> Bước 2 -> Bước 3 -> Bước 4 -> Bước 1 ….

Đáp án của bài toán: Vậy nên kỳ vọng sẽ phải khởi tạo **vô tận** bàn cờ.

Góp ý thêm: **Giả sử *Bước 4* có thể đến *Bước 5***, ta nhận thấy sẽ không bao giờ tìm được bàn cờ đúng. Bởi bước xuất đáp án duy nhất là *Bước 2*, trong khi từ vòng lặp {*Bước 3*, *Bước 4*, *Bước 5}* muốn đến *Bước 2* để xuất đáp án thì chỉ có thể thông qua *Bước 4* -> *Bước 1*, nhưng *Bước 4* thì chỉ có **trường hợp SAI thì mới xuống *Bước 1* được**, và *Bước 1* thì lại luôn luôn khởi tạo bàn cờ => Dẫn tới ***Bước 2* không bao giờ có đáp án để in ra**