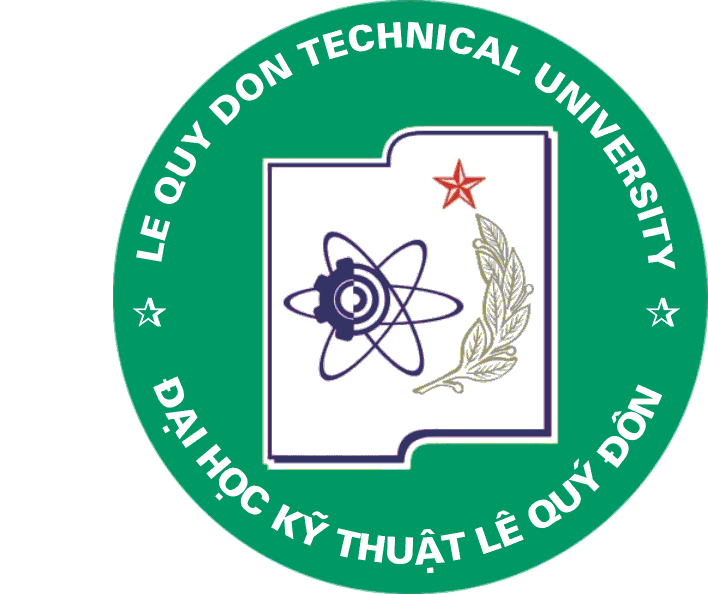
**HỌC VIỆN KỸ THUẬT QUÂN SỰ**

**KHOA CÔNG NGHỆ THÔNG TIN**



BÀI BÁO CÁO MÔN

PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT

Nội dung tìm hiểu : Select\_K

Giáo viên hướng dẫn: GVC.Nguyễn Mậu Uyên

Sinh viên thực hiện: Trần Thị Thúy Mai – CNTT14

Mã SV:15150334

Hà Nội,2017

Mục lục

[I/ Nêu bài toán: 3](#_Toc501168662)

[II/Thuật toán 3](#_Toc501168663)

[III/ Đánh giá độ phức tạp của thuật toán 5](#_Toc501168664)

[IV/Ví dụ 5](#_Toc501168665)

[VI/Kết quả thực hiện chương trình 6](#_Toc501168666)

# I/ Nêu bài toán:

Quickselect là một thuật toán chọn để tìm phần tử thứ K trong một danh sách không có thứ tự.Nó liên quan đến thuật toán sắp xếp nhanh.Quickselect sử dụng cùng một cách tiếp cận tổng thể như quicksort.chọn một phần tử làm trục và phân chia dữ liệu theo hai trục dựa trên trục,vì vậy là lớn hơn hoặc nhỏ hơn trục quay.Thay vì đệ quy vào cả hai mặt như trong quicksort, quickselect chỉ recurses vào một mặt-bên với các yếu tố nó đang tìm kiếm.Điều này làm giảm độ phức tạp trung bình từ O(nlogn) sang O(n),với một trường hợp xấu nhất của O(n2).

II/Thuật toán

Trong quicksort, có một subprocedure được gọi là phân vùng có thể, trong thời gian tuyến tính, nhóm một danh sách (từ các chỉ số leftđến right) thành hai phần, những người ít hơn một phần tử nhất định, và những người lớn hơn hoặc bằng phần tử. Đây là giả mã thực hiện phân vùng về phần tử list[pivotIndex]:

phân vùng chức năng (danh sách, trái, phải, pivotIndex)

pivotValue: = list [pivotIndex]

danh sách trao đổi [pivotIndex] và danh sách [phải] *// Chuyển pivot để kết thúc*

storeIndex: = trái

cho i từ trái sang phải-1

nếu danh sách [i] <pivotValue

danh sách trao đổi [storeIndex] và danh sách [i]

increment storeIndex

danh sách trao đổi [phải] và danh sách [storeIndex] */ / Di chuyển pivot đến nơi cuối cùng của nó*

trở lại storeIndex

Trong quicksort, chúng ta đệ quy cả hai nhánh, dẫn đến thời gian O ( *n* log *n* ) tốt nhất. Tuy nhiên, khi thực hiện lựa chọn, chúng ta đã biết được phân vùng mà phần tử mong muốn của chúng ta nằm ở đâu, vì trục chính nằm ở vị trí đã được sắp xếp cuối cùng, với tất cả các vị trí trước nó theo thứ tự không được sắp xếp và tất cả những gì theo nó theo thứ tự không được sắp xếp. Vì vậy, một cuộc gọi đệ quy đơn giản xác định vị trí của phần tử mong muốn trong phân vùng chính xác, và chúng ta xây dựng dựa trên phép này cho quickselect:

*/ / Trả về phần tử nhỏ thứ k trong danh sách bên trong trái .. bao gồm*

*// (nghĩa là trái <= k <= bên phải).*

*// Không gian tìm kiếm trong mảng đang thay đổi cho mỗi vòng - nhưng danh sách*

*// vẫn có cùng kích thước. Vì vậy, k không cần phải được cập nhật với mỗi vòng.*

chức năng chọn (danh sách, trái, phải, k)

nếu trái = đúng *// Nếu danh sách chỉ chứa một phần tử,*

trở lại danh sách [left] *// trở lại yếu tố đó*

pivotIndex: = ... *// chọn một pivotIndex giữa trái và phải ,*

*// eg,* left + floor (rand ()% (phải - trái + 1))

pivotIndex: = phân vùng (danh sách, trái, phải, pivotIndex)

*// Các trục ở vị trí sắp xếp cuối cùng của nó*

nếu k = pivotIndex

trở lại danh sách [k]

else if k <pivotIndex

trở chọn (danh sách, trái, pivotIndex - 1, k)

khác

trở chọn (danh sách, pivotIndex + 1, bên phải, k )

Lưu ý sự giống nhau đến quicksort: cũng giống như các thuật toán lựa chọn tối thiểu dựa trên là một sắp xếp chọn một phần, đây là một quicksort một phần, tạo và phân vùng chỉ O (log *n* ) O của nó ( *n* ) phân vùng. Thủ tục đơn giản này đã dự kiến ​​hoạt động tuyến tính, và, như quicksort, đã thực hiện khá tốt trong thực tế. Nó cũng là một [thuật toán tại chỗ](https://en.wikipedia.org/wiki/In-place_algorithm), chỉ đòi hỏi bộ nhớ liên tục trên không nếu tối ưu hóa đuôi gọi là có sẵn, hoặc nếu loại bỏ [đệ quy đuôi](https://en.wikipedia.org/wiki/Tail_recursion) với một vòng lặp:

chức năng chọn (danh sách, trái, phải, k)

vòng lặp

nếu trái = phải

trở lại danh sách [left]

pivotIndex: = ... *// chọn pivotIndex giữa trái và phải*

pivotIndex: = phân vùng (danh sách, trái, phải, pivotIndex)

nếu k = pivotIndex

return list [k]

else nếu k <pivotIndex

phải: = pivotIndex - 1

khác

trái: = pivotIndex + 1

# III/ Đánh giá độ phức tạp của thuật toán

Giống như quicksort, quickselect có hiệu suất trung bình tốt, nhưng nhạy cảm với trục quay được chọn. Nếu các trục quay tốt được chọn, có nghĩa là các trục chính xác định giảm tập hợp tìm kiếm do một phân số nhất định, thì bộ tìm kiếm sẽ giảm kích thước theo cấp số nhân và bằng cách đưa ra (hoặc tổng hợp các [hình học](https://en.wikipedia.org/wiki/Geometric_series) ) một lần thấy hiệu suất là tuyến tính, vì mỗi bước là tuyến tính và thời gian tổng thể là một lần liên tục này (phụ thuộc vào cách nhanh chóng tìm kiếm tập giảm). Tuy nhiên, nếu pivot xấu được lựa chọn một cách nhất quán, chẳng hạn như chỉ giảm mỗi lần một phần tử, sau đó hiệu suất trường hợp xấu nhất là bậc hai: O ( *n*2 ). Điều này xảy ra ví dụ như khi tìm kiếm phần tử tối đa của tập hợp, sử dụng phần tử đầu tiên làm trục chính, và có dữ liệu được sắp xếp.

**IV/Ví dụ**

1/

Arr = [5 1 4 3 2]

Pivot = [4]

Các bước:

trao đổi [5] và [2] như 5> = 4 và 2 <4

[2 1 4 3 5]

hoán đổi [4] và [3] như 4> = 4 và 3 <4

[2 1 3 4 5]

Khi chúng ta kết thúc với lần lặp đầu tiên, chúng ta biết những điều sau:   
Tất cả các phần tử <4 nằm ở bên trái của 4   
Tất cả các phần tử> = 4 nằm ở bên phải của 4 (bao gồm cả 4 chính nó)

Kết quả phần tử nhỏ thứ 2(k=2)trong mảng là:3

2/

Arr = [7 4 6 3 9 1]

Pivot = [3]

Các bước:

trao đổi [7] và [1] như 7> = 4 và 1 <3

[1 4 6 3 9 7]

hoán đổi [4] và [3] như 4> = 4 và 3 =3

[1 3 6 4 9 7]

hoán đổi [4] và [6] để được dãy theo thứ tự

[1 3 4 6 9 7]

hoán đổi [9] và [7] để được dãy theo thứ tự

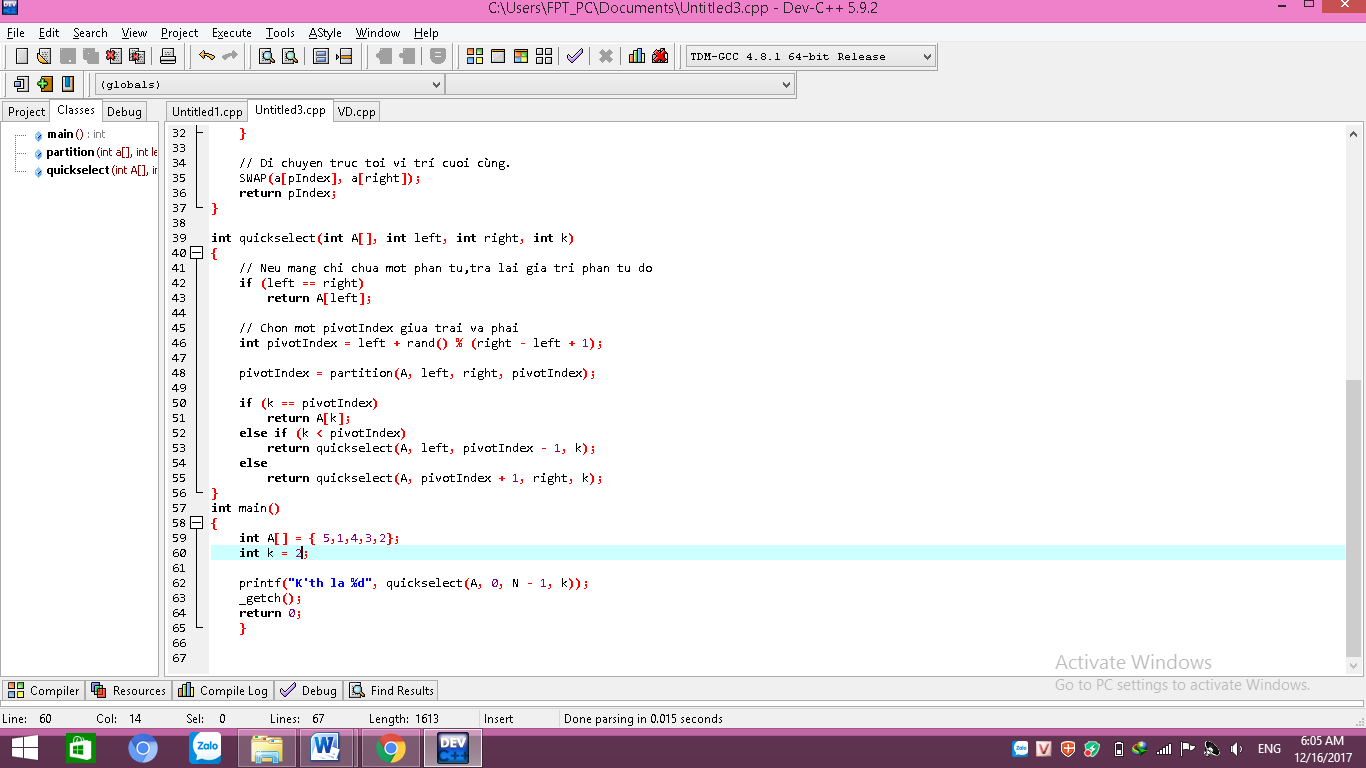
[1 3 4 6 7 9]

Đảm bảo tất cả các phần tử nhỏ hơn 3 nằm phía bên tái của 3,phần tử lớn hơn 3 nằm ở phía bên phải của 3 theo một thứ tự nhất định.

Kết quả thu được phần tử nhỏ thứ 0(k=0) là:1

# VI/Kết quả thực hiện chương trình

Chọn nhanh:



Kết quả thu được khi k=2:

