**HỌC VIỆN NGÂN HÀNG**



**KHOA HỆ THỐNG THÔNG TIN QUẢN LÝ**

****

**BÀI TẬP LỚN**

**ĐỀ TÀI: THUẬT TOÁN SẮP XẾP**

**SHELL SORT**

|  |  |  |
| --- | --- | --- |
|  | **Giảng viên hướng dẫn: TS. Nguyễn Thanh Thụy** Nhóm thực hiện : Nhóm 12 |  |

**Hà Nội, ngày tháng 12 năm 2020**

**Nhóm 12**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **Mã sinh viên** | **Phần trăm đóng góp** |
| 1 | Trần Hải Yến | 22A4040139 | 25% |
| 2 | Nguyễn Thị Ánh Quyên | 22A4040100 | 25% |
| 3 | Trần Quý Huy | 18A4040092 | 25% |
| 4 | Bùi Thị Ánh Nguyệt | 21A4040081 | 25% |

**TÀI LIỆU THAM KHẢO**

1. **Giới thiệu thuật toán sắp xếp Shell Sort**

* Shell Sort (hay Shellsort, Shell’s method) là một phương thức sắp xếp so sánh tại chỗ được Donald Shell công bố vào năm 1959. Nó có thể coi như là phương thức tổng quát của sắp xếp bằng cách trao đổi (bubble sort) hay sắp xếp bằng cách chèn (insertion sort). Shell Sort thực hiện nhiều hành động hơn và có tỉ lệ bộ nhớ đệm cao hơn so với Quick Sort. Tuy nhiên, vì nó có thể triển khai bằng cách sử dụng ít mã và không sử dụng call stack, một vài chức năng triển khai của Quick Sort trong thư viện C standard được nhắm đến tại hệ thống nhúng sử dụng nó thay vì Quick Sort. Ví dụ, Shell Sort được sử dụng trong thư viện uClibc hay hạt nhân Linux. Shell Sort cũng có thể sử dụng như một thuật toán con của introspective sort, để sắp xếp các mảng con ngắn và ngăn chặn sự chậm lại khi độ sâu đệ quy vượt quá giới hạn cho trước.
* [2]Đầu tiên, giải thuật này sử dụng giải thuật sắp xếp chọn trên các phần tử có khoảng cách xa nhau, sau đó sắp xếp các phần tử có khoảng cách hẹp hơn. Khoảng cách này còn được gọi là **khoảng (interval)** – là số vị trí từ phần tử này tới phần tử khác. Khoảng này được tính dựa vào công thức sau:  
   **h=h\*3+1**<Trong đó: h là khoảng (interval) với giá trị ban đầu là 1>
* Giải thuật này khá hiệu quả với các tập dữ liệu có kích cỡ trung bình khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n), với n là số phần tử.

**2. Ý tưởng**

* [4]Là giải thuật cải tiến từ Insertion Sort.
* Ý tưởng chính của thuật toán là phân chia dãy ban đầu thành những dãy con mà mỗi phần tử của dãy cách nhau 1 vị trí là h. Insertion Sort áp dụng sau đó trên mỗi dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối (trong dãy con) 1 cách nhanh chóng.
* Sau đó tiếp tục giảm khoảng cách h để tạo thành các dãy con mới (Tạo điều kiện để so sánh một phần tử với nhiều phần tử khác trước đó không ở cùng dãy con với nó) và lại tiếp tục sắp xếp.
* Thuật toán dừng khi h = 1, lúc này bảo đảm tất cả các phần tử trong dãy ban đầu sẽ được so sánh với nhau để xác định trật tự cuối cùng.

**3. Giải thích**

* [1]Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau **h** vị trí
* Dãy ban đầu : **a1, a2, ..., an** được xem như sự xen kẽ của các dãy con sau :
* Dãy con thứ nhất : **a1 ah+1 a2h+1 ...**
* Dãy con thứ hai : **a2 ah+2 a2h+2 ...**
* ....
* Dãy con thứ h : **ah a2h a3h ...**
* Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối
* Giảm khoảng cách **h** để tạo thành các dãy con mới
* Dừng khi h=1
* Giả sử quyết định sắp xếp **k** bước, các khoảng cách chọn phải thỏa điều kiện :

hi > hi+1 và hk = 1

* hi = (hi-1 - 1)/3 và hk = 1, k = log3n-1

Ví dụ :127, 40, 13, 4, 1

* hi = (hi-1 - 1)/2 và hk = 1, k = log2n-1

Ví dụ : 15, 7, 3, 1

* h có dạng 3i+1: 364, 121, 40, 13, 4, 1
* Dãy fibonaci: 34, 21, 13, 8, 5, 3, 2, 1
* h là dãy các số nguyên tố giảm dần đến 1: 13, 11, 7, 5, 3, 1.

**4. Thuật toán**

* [1] Bước 1: Chọn **k** khoảng cách h[1], h[2], ..., h[k];

i = 1;

* Bước 2:

Phân chia dãy ban đầu thành các dãy con cách nhau h[i] khoảng cách.

Sắp xếp từng dãy con bằng phương pháp chèn trực tiếp;

* Bước 3: i = i+1;

Nếu i > k : Dừng   
         Ngược lại : Lặp lại Bước 2.

**5. Cài đặt thuật toán**

[1]void ShellSort(int a[], int n, int h[], int k)

{

int step, i, j, x, len;

for (step = 0 ; step <k; step ++)

{

len = h[step];

for (i = len; i <d.n; i++)

{

x = a[i];

j = i-len;

while ((x<a[j])&&(j>=0 ))

{

a[j+len] = a[j];

j = j - len;

}

a[j+len] = x;

}

}

}

**6. Ví dụ minh họa**

Cho dãy a:

11 16 12 71 51 54 5 25

Chọn các khoảng cách là h=(5,3,1)

**h=5**: Xem dãy ban đầu như các dãy con

**11 16 12 71 51 54 5 25**

**h=3**: Sau khi đã sắp xếp các dãy con ở bước trước

**11 5 12 71 51 54 16 25**

**h=1:** Sau khi đã sắp xếp các dãy con ở bước trước

**11 5 12 16 25 54 71 51**

**11 5 12 16 25 54 71 51**

**5 11 12 16 25 54 71 51**

**5 11 12 16 25 51 54 71**

**7. So sánh thuật toán [3]**

|  |  |  |
| --- | --- | --- |
| **Thuật toán** | **TH tốt nhất** | **TH xấu nhất** |
| Selection Sort | 0(n2) | 0(n2) |
| Insertion Sort | 0(n) | 0(n2) |
| Shell Sort | 0(n\* log(n)) | 0(n1,5) |
| Quick Sort | 0(n\* log(n)) | 0(n2) |
| Merge Sort | 0(n\* log(n)) | 0(n\* log(n)) |
| Heap Sort | 0(n) | 0(n\* log(n)) |

**8. Độ phức tạp của thuật toán**

 [4]Yếu tố quyết định chính của thuật toán chính là cách chọn khoảng cách h trong từng bước sắp xếp và số bước sắp xếp k. Nhưng phải thỏa 2 điều kiện sau: hi > hi+1 và hk = 1

Các phần tử h không được là bội số của nhau nhằm tránh hiện tượng mỗi bước sắp thứ tự phải tổ hợp 2 nhóm mà bước trước chúng không hề có ảnh hưởng lẫn nhau. Điều mong muốn là ảnh hưởng giữa các nhóm khác nhau càng nhiều càng tốt.

Việc đánh giá giải thuật Shell sort hiện nay rất phức tạp, thậm chí 1 số chưa được chứng minh. Nhưng có 1 điều chắc chắn là hiệu quả của thuật toán phụ thuộc vào dãy các độ dài được chọn. Trong trường hợp chọn dãy độ dài theo công thức hi =(hi-1 -1)/2 và hk = 1 ,k = log2 -1 thì giải thuật có độ phức tạp tương đương n1,2 << n2 .