CÁC THUẬT TOÁN SẮP XẾP

Bùi Tiến Lên

01/01/2017

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Bài toán sắp xếp

- Sắp xếp một danh sách các đối tượng theo một thứ tự nào đó là một trong những công việc phổ biến
- Sắp xếp là một yêu cầu không thể thiếu trong việc viết phần mềm ứng dụng
- Do đó, nghiên cứu các phương pháp sắp xếp là cần thiết cho những ai học lập trình

Bài toán sắp xếp (cont.)

Định nghĩa 1

Cho một dãy a có n phần tử có thứ tự. Hãy sắp xếp dãy $\{a_0, a_1, ..., a_{n-1}\}$ theo thứ tự tăng dần.

Các phương pháp sắp xếp

Có rất nhiều phương pháp sắp xếp khác nhau. Mỗi phương pháp có những đặc điểm riêng

- Phương pháp Selection Sort
- Phương pháp Insertion Sort
- Phương pháp Bubble Sort
- Phương pháp Shell Sort
- Phương pháp Heap Sort
- Phương pháp Merge Sort
- Phương pháp Quick Sort
- Phương pháp Radix Sort
- Phương pháp Counting Sort

SELECION SORT

Selection Sort

Ý tưởng của thuật toán như sau: Giả sử dãy a được chia làm hai phần: phần trên trái **đã sắp xếp** s và phần bên phải **chưa sắp xếp** u

1. $s = \emptyset$ và $u = a$
2. Tìm phần tử nhỏ nhất x_m của u
3. Loại x_m ra khỏi u thêm vào cuối của s
4. Nếu u vẫn còn phần tử thì quay lại bước 2

Selection Sort (cont.)

```
1  void SelectionSort(int a[], int n)
2  {
3      int min;
4      for (int i = 0; i < n; i++)
5      {
6          min = i;
7          for (int j = i + 1; j < n; j++)
8              if (a[j] < a[min])
9                  min = j;
10         if (a[min] < a[i])
11             Swap(a[min], a[i]);
12     }
13 }
```

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

INSERTION SORT

Insertion Sort

Ý tưởng của thuật toán như sau: Giả sử dãy a được chia làm hai phần: phần trên trái **đã sắp xếp** s và phần bên phải **chưa sắp xếp** u

1. $s = \emptyset$ và $u = a$
2. Lấy phần tử đầu tiên x của u
3. Loại x ra khỏi u
4. Chèn x vào s sao cho đúng vị trí
5. Nếu u vẫn còn phần tử thì quay lại bước 2

Insertion Sort (cont.)

```
 1  void InsertionSort(int a[], int n)
 2  {
 3      int saved;
 4      for (int i = 1; i < n; i++)
 5      {
 6          saved = a[i];
 7          for (int j = i; j > 0 && saved < a[j - 1]; j--)
 8              a[j] = a[j - 1];
 9          a[j] = saved;
10      }
11  }
```

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

BUBBLE SORT

Bubble Sort

Thuật toán Bubble Sort là một trường hợp cụ thể của Selection Sort. Giai đoạn tìm phần tử nhỏ nhất được thực hiện bằng cách làm cho phần tử nhỏ nhất nổi ra phía đầu của dãy u

Bubble Sort (cont.)

```
1  void BubbleSort(void)
2  {
3      int i, j;
4      for (i = 0; i <= n - 2; i++)
5          for (j = n - 1; j >= i + 1; j--)
6              if (a[j] < a[j - 1])
7                  Swap(a[j], a[j - 1]);
8  }
```

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

SHELL SORT

Shell Sort

- Được đề xuất bởi [Shell, 1959]
- Thuật toán này là cải tiến của thuật toán Insertion Sort
- Có sự đột phá về rào cản chi phí $O(n^2)$ của những thuật toán trước

Ý tưởng của thuật toán

- Chia dãy a thành h dãy con

$$a_0, a_{0+h}, a_{0+2h}, \cdots$$
$$a_1, a_{1+h}, a_{1+2h}, \cdots$$
$$a_2, a_{2+h}, a_{2+2h}, \cdots$$
$$\cdots$$

- Sắp xếp từng dãy con bằng cách sử dụng phương pháp chèn trực tiếp Insertion Sort

Shell Sort (cont.)

Thuật toán sẽ

- Sử dụng một 1 dãy $\{h_1, h_2, ..., h_t\}$ là một dãy giảm dần và có $h_t = 1$.
- Vấn đề là lựa chọn dãy h_i như thế nào cho hiệu quả

Các chiến lược cho dãy h_i như sau

- Shell đề xuất

$$h_1 = \frac{n}{2}, h_{i+1} = \frac{h_i}{2}$$

- Hibbard đề nghị

$$h_i = 2^i - 1$$

- Knuth đưa ra

$$h_1 = 1, h_{i+1} = 3h_i + 1$$

- Pratt đề xuất

$$h_1 = 1, h_i = 2^p 3^q$$

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

HEAP SORT

Heap Sort

Định nghĩa 2

Cây heap là một cây nhị phân bộ phận hoàn chỉnh

- Có hai cây heap
  - Max heap: khóa của mỗi nút không nhỏ hơn khóa của các con của nó
  - Min heap: khóa của mỗi nút không lớn hơn khóa của các con của nó
- Nút gốc của cây bộ phận là phần tử nhỏ nhất của cây

Biểu diễn Heap bằng mảng

Ta có thể biểu diễn cây nhị phân đầy đủ bằng mảng. Ví dụ sau sẽ minh họa cách biểu diễn một dãy gồm có 6 phần tử $\{a_0, a_1, a_2, a_3, a_4, a_5\}$ bằng cây nhị phân đầy đủ

- Nút gốc là a_0
- Nút lá là a_3, a_4, a_5

Ví dụ 1

Hãy biểu diễn một dãy {1, 4, 8, 9, 3, 7} bằng một cây nhị phân đầy đủ

Có một số nhận xét sau về biểu diễn mảng của cây nhị phân đầy đủ

- Nút gốc ở chỉ số [0]
- Nút cha của nút [i] có chỉ số là [(i-1)/2]

Biểu diễn Heap bằng mảng (cont.)

- Các nút con của nút [i] có chỉ số là [2i+1] và [2i+2]
- Nút con của nút [0] là [1] và [2]
- Nút cha của nút [7] là [3]
- Nút cha của nút [6] là [2]

Thao tác điều chỉnh cây nhị phân thành Heap

Xét một nút trên cây

1. Nếu nút đang xét có giá trị lớn hơn giá trị của nút con của nó thì tiến hành đổi chỗ với nút con có giá trị lớn nhất
2. Tiếp tục tiến hành tương tự cho nút con vừa đổi

Cài đặt hiệu chỉnh cây

Sau đây là một hàm cài đặt bằng C cho thao tác hiệu chỉnh cây thành Heap

```
1  void Heapify(int a[], int n, int i)
2  {
3      int saved = a[i];
4      while (i < n / 2)
5      {
6          int child = 2 * i + 1;
7          if (child < n - 1)
8              if (a[child] > a[child + 1])
9                  child++;
10         if (saved <= a[child])
11             break;
12         a[i] = a[child];
13         i = child;
14     }
15     a[i] = saved;
16 }
```

Thuật toán Heap Sort

Thuật toán sắp xếp Heap Sort

- Xây dựng cây Heap: Sử dụng thao tác hiệu chỉnh Heapify để chuyển một mảng bình thường thành một mảng Heap
- Thao tác sắp xếp
  1. Hoán vị phần tử cuối cùng của mảng Heap với phần tử đầu tiên của Heap
  2. Loại bỏ phần tử cuối cùng khỏi mảng Heap
  3. Thực hiện thao tác hiệu chỉnh Heapify với phần còn lại mảng Heap

Khi xây dựng mảng Heap hãy lưu ý một số điểm sau

- Tất cả các phần tử trên mảng Heap có chỉ số [n/2] đến [n-1] đều là nút lá
- Thực hiện thao tác hiệu chỉnh Heapify cho các phần tử có chỉ số từ [n/2-1] đến [0]

Thuật toán Heap Sort (cont.)

Chương trình 1: Sau đây là cài đặt thuật toán hàm xây dựng mảng Heap

```
1  void BuildHeap(int a[], int n)
2  {
3      for (int i = n / 2 - 1; i >= 0; i--)
4          Heapify(a, n, i);
5  }
```

Thuật toán Heap Sort (cont.)

Chương trình 2: Sau đây là cài đặt thuật toán sắp xếp

```
1  void HeapSort(int a[], int n)
2  {
3      BuildHeap(a, n);
4      for (int i = n - 1; i >= 0; i--)
5      {
6          Swap(a[0], a[i]);
7          Heapify(a, i, 0);
8      }
9  }
```

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

MERGE SORT

Merge Sort

Hàm MergeSort nhận một danh sách có độ dài n và trả về một danh sách đã được sắp xếp. Hàm Merge nhận hai danh sách đã được sắp L1 và L2 mỗi danh sách có độ dài $n/2$, trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự.

```
1  List MergeSort(List L)
2  {
3      List L1, L2;
4      if (L.length <= 1)
5          return (L);
6      else
7      {
8          Split(L, L1, L2);
9          return (Merge(MergeSort(L1), MergeSort(L2)));
10     }
11 }
```

Minh họa hoạt động Merge Sort

Sắp xếp danh sách L gồm 8 phần tử $\{7, 4, 8, 9, 3, 1, 6, 2\}$

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

QUICK SORT

Quick Sort

Ý tưởng của thuật toán như sau: Đây cũng là một hướng tiếp cận "chia nhỏ bài toán"

1. Chia dãy a thành hai dãy con bên trái a_{left} và bên phải a_{right} sao cho các phần tử thuộc dãy trái đều nhỏ hơn các phần tử thuộc dãy bên phải
2. Sắp xếp cho các dãy a_{left} và a_{right}
3. Nối hai dãy trái và phải với nhau $a = a_{left}\, a_{right}$

Quick Sort (cont.)

Chương trình 3: Sau đây là cài đặt hàm Quick Sort

```
1  List QuickSort(List a)
2  {
3      List aleft, aright;
4      if (a.Length <= 1)
5          return a;
6      else
7      {
8          Split(a, aleft, aright);
9          return Join(QuickSort(aleft), QuickSort(aright)
               )
10     }
11 }
```

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

RADIX SORT

Radix Sort

Ý tưởng

- Các phương pháp sắp xếp trước đây chỉ sử dụng phương pháp so sánh thử và sai
- Radix Sort là phương pháp sắp xếp sử dụng thông tin của khóa.
- Cụ thể là phương pháp này sẽ nhóm các khóa có cùng chữ số thứ i thành một nhóm đây là bước phân bố
- Bước kế tiếp là kết hợp các nhóm này lại thành một dãy duy nhất. Quá trình phân bố-kết hợp được thực hiện cho đến khi hết các chữ số
- Vậy có hai phương pháp chính: *Least significant digit* đi từ phải qua trái và *Most significant digit* đi từ trái qua phải

Cài đặt

```
1  void RadixSort(int a[], int n)
2  {
3      int i, j, k, factor;
4      const int radix = 10;
5      const int digits = 6;
6      Queue<long> queues[radix];
7      for (i = 0, factor = 1; i < digits; factor *= radix
           ; i++)
8      {
9          // Phan bo
10         for (j = 0; j < n; j++)
11             queues[(a[j] / factor) % radix].enqueue(a[j
                   ]);
12         // Ket hop
13         for (j = k = 0; j < radix; j++)
14             while (!queues[j].empty())
15                 a[k++] = queues[j].dequeue();
16     }
17 }
```

Minh họa thuật toán

- Cho một dãy chưa sắp xếp {170, 45, 75, 90, 802, 2, 24, 66}
- Viết lại cho đủ 3 chữ số {170, 045, 075, 090, 802, 002, 024, 066}

Minh họa thuật toán (cont.)

- Phân bố theo nhóm hàng đơn vị {170, 045, 075, 090, 802, 002, 024, 066}

```
0: 170 090
1:
2: 002 802
3:
4: 024
5: 045 075
6: 066
7:
8:
9:
```

- Gộp các nhóm {170, 090, 002, 802, 024, 045, 075, 066}

Minh họa thuật toán (cont.)

- Phân bố theo nhóm hàng chục {170, 090, 002, 802, 024, 045, 075, 066}

```
0: 002, 802
1:
2: 024
3:
4: 045
5:
6: 066
7: 170, 075
8:
9: 090
```

- Gộp các nhóm {002, 802, 024, 045, 066, 170, 075, 090}

Minh họa thuật toán (cont.)

- Phân bố theo nhóm hàng trăm {002, 802, 024, 045, 066, 170, 075, 090}

```
0: 002, 024, 045, 066, 075, 090
1: 170
2:
3:
4:
5:
6:
7:
8: 802
9:
```

- Gộp các nhóm {002, 024, 045, 066, 075, 090, 170, 802}

Đánh giá

Phân tích chi phí thực hiện theo n (số lượng phần tử của mảng)

Trường hợp	$O(g(n))$
tốt nhất	?
trung bình	?
xấu nhất	?

COUNTING SORT

Counting Sort

Ý tưởng

- Counting Sort là phương pháp sắp xếp sử dụng thông tin của khóa.
- Các số sẽ được đếm
- Căn cứ vào giá trị đếm thì sẽ xác định chính xác vị trí của số trong mảng

Cài đặt

```
1  const int RANGE = 256;
2  void countSort(int a[], int n)
3  {
4    int i, j;
5    int count[RANGE];
6    memset(count, 0, sizeof(count));
7    for (i = 0; i < n; i++)
8      count[a[i]]++;
9    i = 0;
10   for (int j = 0; j < RANGE; j++)
11   {
12     while (count[j] > 0) {
13       a[i] = j;
14       count[j]--;
15       i++;
16     }
17   }
18 }
```

Tài liệu tham khảo


Shell, D. L. (1959).
A high-speed sorting procedure.
Communications of the ACM, 2(7):30–32.