Trường ĐH Bách Khoa Tp.HCM
Khoa KH&KT Máy tính

ĐÁP ÁN

Đề thi giữa kỳ HK1/2009

Môn: Cấu trúc dữ liệu và Giải thuật

Thời gian: 60 phút (Không sử dụng tài liệu)

Ghi chú: *đề thi gồm tất cả 7 câu. Sinh viên lớp KSTN làm hết 7 câu, thang điểm 12/12. Sinh viên lớp thường làm 6 câu (từ câu 1 đến câu 6), thang điểm 10/10.*

Câu 1 (1.5 điểm): Tính toán big-O của các hàm dưới đây và sắp xếp chúng theo thứ tự từ nhỏ đến lớn theo big-O:

Đáp áp:

a) (1 điểm) Tính big-O

a. $2^n = O(2^n)$

b. $n! = O(n!)$

c. $n^{3.5} = O(n^{3.5})$

d. $n + n^2 + n^3 = O(n^3)$

e. $10^5 = O(1)$

f. $150,000 = O(1)$

g. $n\log_2(n) = O(n\log_2(n))$

b) (0.5 điểm) Sắp xếp theo big-O:

$10^5 <= 150,000 < n\log_2(n) < (n+n^2+n^3) < n^{3.5} < 2^n < n!$

Câu 2 (2.5 điểm): Cho một DSLK đơn gồm các số nguyên có cấu trúc như hình bên. Trong đó, mỗi thành phần của DSLK đơn là một cấu trúc có data là số nguyên và con trỏ link trỏ đến phần tử kế tiếp. DSLK đơn chỉ dùng một con trỏ head để chỉ đến phần tử đầu tiên của danh sách. Nếu danh sách rỗng, con trỏ head này là null. Viết một **phương thức** bằng pseudocode nhận vào một số nguyên, tìm trong DSLK đơn và loại bỏ đi các phần tử có giá trị bằng hoặc hơn số nguyên này 1 hoặc 2. Lưu ý, không dùng thêm bất kỳ phương phức hoặc hàm phụ trợ nào (kể cả tự viết lại).

```
Node
  data <int>
  link <pointer>
end Node

Linked List
  head <pointer>
end Linked List
```

Ví dụ, với danh sách là {12, 13, 5, 6, -8, 9, 7, -2, 5, -1, 6, -3} và số nguyên nhận được là 5 thì danh sách kết quả là {12, 13, -8, 9, -2, -1, -3}, tức là các phần tử 5,6,7 bị xóa đi.

Đáp áp:

algorithm remove_in_range (**val** x <**int**>)
    Post Các phần tử có data y sao cho (y-x) là 0,1,2 bị xóa đi

1. pre = null, tmp = head
2. **loop** (tmp is not null)
    1. **if** ((tmp->data == x) or (tmp->data == x+1) or (tmp->data == x+2))
        1. **if** (pre is null) //delete first
            1. head = head->link
            2. delete tmp
            3. tmp = head
        2. **else** //delete the element after pre
            1. pre->next = tmp->link
            2. delete tmp
            3. tmp = pre->link
        3. **end if**
    2. **else**
        1. pre = tmp
        2. tmp = tmp->link
    3. **end if**
3. **end loop**

end remove_in_range

Câu 3 (2 điểm): Viết một hàm toàn cục (global function) bằng pseudocode nhận vào một queue và đảo ngược queue đó. Giả sử rằng các phương thức của queue và stack được cho theo đặc tả của hình bên cạnh. Chú ý: không được viết và dùng thêm các hàm phụ trợ nào khác.

Stack ADT
<void> Create()
<ErrorCode> Push (val DataIn <DataType>) //Thêm 1 phần tử vào đỉnh stack
<ErrorCode> Pop () //Bỏ phần tử trên đỉnh stack
<ErrorCode> Top (ref DataOut <DataType>) //Xem phần tử trên đỉnh stack
<boolean> isEmpty ()

Queue ADT
<void> Create()
<ErrorCode> EnQueue (val DataIn <DataType>) //Thêm 1 phần tử vào cuối queue
<ErrorCode> DeQueue () //Bỏ 1 phần tử đầu queue
<ErrorCode> QueueFront (ref DataOut <DataType>) //Xem phần tử đầu queue
<ErrorCode> QueueRear (ref DataOut <DataType>) //Xem phần tử cuối queue
<boolean> isEmpty ()

Đáp áp:

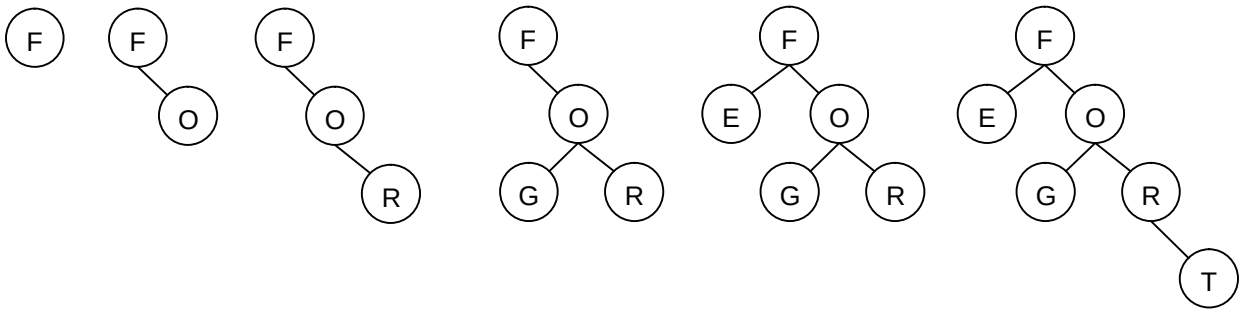
algorithm reverse_queue (**ref** queue <**Queue**>)
Post các phần tử trong queue sẽ bị đảo ngược vị trí

1. stack = Create a stack
2. **loop** (not queue.isEmpty())
    1. queue.QueueFront (x)
    2. stack.Push (x)
    3. queue.DeQueue()
3. **end loop**
4. **loop** (not stack.isEmpty())
    1. stack.Top (x)
    2. queue.EnQueue (x)
    3. stack.Pop()
5. **end loop**

end reverse_queue

Câu 4 (1.5 điểm): Hãy trình bày từng bước quá trình tạo một cây nhị phân tìm kiếm (BST) bằng cách thêm vào trong cây rỗng ban đầu các khóa lần lượt như sau: F,O,R,G,E,T biết rằng giá trị so sánh của các khóa này là thứ tự của chúng trong bảng chữ cái.

Đáp áp:



Câu 5 (1.5 điểm): Trình bày từng bước quá trình tìm kiếm khóa 31 dùng phương pháp tìm kiếm nhị phân binary_search_1 (forgetful version) trên danh sách liên kết (DSLK) đơn có thứ tự như sau: {1, 12, 31, 35, 63, 98 }. Có bao nhiêu lần so sánh trên khóa?

```
<ErrorCode> binary_search_1 (val  target <KeyType>, ref position <int>)

 1. bottom = 0
 2. top = size of the list
 3. loop (bottom < top)
    1. mid = (bottom + top)/2
    2. if (target > data_mid)
       1. bottom = mid + 1
    3. else
       1. top = mid
    4. end if
 4. end loop
 5. if (top < bottom)
    1. return notFound
 6. else
    1. position = bottom
    2. if (target = data_position)
       1. return found
    3. else
       1. return notFound
    4. end if
 7. end if
end binary_search_1
```

Đáp áp:

idx	0	1	2	3	4	5
data	1	12	31	35	63	98

+ bottom = 0, top = 5

bottom < top: true

mid = (0+5)/2 = 2

target=31 > $data_2$ = 31 : false => top = mid = 2 — 1 lần so sánh

+ bottom = 0, top = 2

bottom < top: true

mid = (0+2)/2 = 1

target=31 > $data_1$ = 12: true => bottom = =mid+1 = 2 — 1 lần so sánh

+ bottom = 2, top = 2

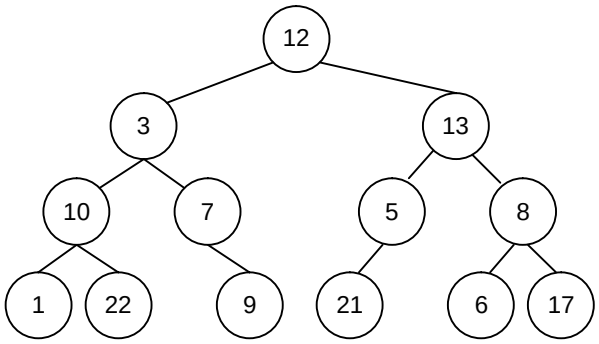
bottom < top: false

+ target=31 = $data_2$: true => found 1 lần so sánh

Vậy có tổng cộng 3 lần so sánh

Câu 6 (1 điểm): Cho cây nhị phân như hình vẽ, hãy cho biết kết quả thực thi của giải thuật sau nếu giải thuật được gọi từ phần tử gốc của cây (nút có giá trị 12).

```
algorithm XYZ (val subroot <pointer>)
1. if (subroot is not null)
   1. print "<"
   2. print (subroot->data)
   3. print ">"
   4. XYZ (subroot->left)
   5. XYZ(subroot->right)
2.end if
end XYZ
```



Đáp áp:

<12><3><10><1>< 22><7><9><13><5><21><8><6><17>

Câu 7 (2 điểm – Dành cho lớp KSTN): Danh sách liên kết đơn vòng (xem đặc tả ở hình bên cạnh) được quản lý như sau:

- Con trỏ current chỉ đến phần tử đầu tiên.
- Nếu danh sách rỗng, current là NULL. Ngược lại, con trỏ link của phần tử cuối chỉ vào phần tử đầu tiên.

```
Node
  data <int>
  link <pointer>
end Node

Circular Linked List
  current <pointer>
end Circular Linked List
```

Viết một **phương thức** bằng pseudocode để đếm số phần tử của danh sách này. Lưu ý, không dùng thêm bất kỳ phương phức hoặc hàm phụ trợ nào (kể cả tự viết lại).

Đáp áp:

```
algorithm circular_list_size ()
        Pre Danh sách liên kết đơn vòng
        Post Số phần tử được đếm
        Return Số phần tử của danh sách

        1. if (current is null)
                1. return 0
        2. else
                1. size=1
                2. tmp = current
                3. loop (tmp->link != current)
                        1. size++
                        2. tmp = tmp->link
                4. end loop
        3. end if
        4. return size
end circular_list_size
```