Đại học Quốc Gia TP. Hồ Chí Minh
Trường đại học Bách Khoa
Khoa: **Khoa học & Kỹ thuật Máy tính**
Bộ môn: **Khoa học Máy tính**

ĐỀ KIỂM TRA GIỮA HỌC KỲ 1
Năm học: 2011 – 2012
Môn: **Cấu trúc dữ liệu & Giải thuật**
MSMH: 503001
Ngày thi: --/10/2011 - Thời gian: 70 phút
(*Được sử dụng tài liệu*)

Lưu ý: Đề kiểm tra gồm 4 câu với thang điểm 11/10. Sinh viên làm đúng trên 10 điểm sẽ được làm tròn thành 10.

Câu 1: (2 điểm)

Cho một danh sách các số nguyên *a* gồm *n* phần tử. Hãy viết hàm *func(int* a, int n)* có độ phức tạp (theo Big-O Notation) như sau:

a) trong trường hợp xấu nhất $O(n^2)$, và trong trường hợp tốt nhất O(n);

```
void func(int* a, int n) {
    for(int i = 0;i < n;i++) {
        for(int j = i; j< n; j++) {
            if(a[i] > a[j]) {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
}
```

Trường hợp xấu nhất, danh sách đã có thứ tự giảm dần, độ phức tạp O(n^2).
Trường hợp tốt nhất, danh sách đã có thứ tự tăng dần, độ phức tạp O(n).

b) trong trường hợp xấu nhất O(n *log* n), và trong trường hợp tốt nhất O(n).

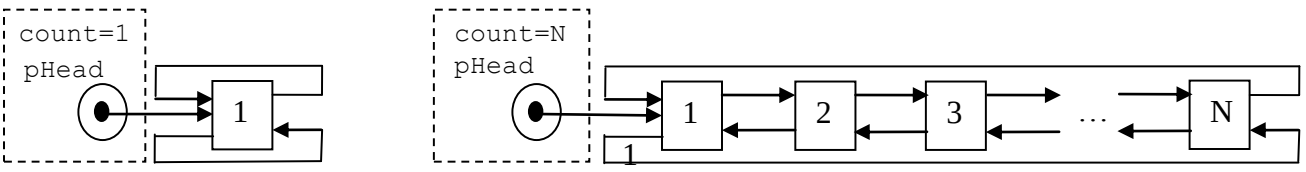
```
void func(int* a, int n) {
    for(int i = 0; i< n;i++) {
        if(a[i]>=10) {
            for(int j = 0; j< n;j = j * 2) {
                println("func");
            }
        }
    }
}
```

Trường hợp xấu nhất, danh sách có mọi phần tử >=10. Độ phức tạp là O(nlogn)
Trường hợp tốt nhất, danh sách không có phần tử nào >=10. Độ phức tạp là O(n).

Gợi ý: chia thành nhiều trường hợp để giải với độ phức tạp khác nhau, như là xét trường hợp n chẵn và n lẻ.

Câu 2: (4 điểm)

Cho một cấu trúc danh sách liên kết vòng đôi được mô tả trong Hình 1.



```
//just an entry in the list, a "struct++" in fact
class Node {
public:
   int data;
   Node* next;
   Node* back;
};

//interface part
class List {
private:
   int count;
   Node* pHead;
public:
   List();
   void add(int data, int index);
   Node* remove(int index);
   void display();
   ~List();
};
```

Hình 1. Đặc tả cấu trúc của danh sách liên kết vòng đôi

Method *remove* sẽ thực hiện các tác vụ sau:

a) rút phần tử ở vị trí thứ *index* ra khỏi danh sách liên kết vòng đôi này (giả sử phần tử bắt đầu của danh sách có chỉ số là 1 và được tham khảo bởi `pHead`),

b) `pHead` sẽ được cập nhật lại bằng cách tham khảo đến phần tử đứng sau, liền kề với phần tử được xóa,

c) *remove* sẽ trả về tham khảo của phần tử vừa được rút ra khỏi danh sách (lưu ý về tính an toàn của danh sách: cần phải gán các tham khảo *next*/*back* của phần tử này về NULL).

Ví dụ : Giả sử danh sách *list* đang là (1,2,3,4,5). Sau khi gọi *list*.*remove*(2) thì *list* sẽ trở thành (3,4,5,1). Nếu gọi tiếp *list*.*remove*(6) thì *list* sẽ trở thành (5,1,3).

Hãy hiện thực method *remove* theo hai cách: (i) không đệ quy và (ii) đệ quy.

Solution:

Đệ qui:

```
Node* remove(int index) {
    Node* pTemp=NULL;
    if  ((index > count) || (count<=0)) return NULL;
    if  (index == 1) {
        pTemp = pHead;
        if (count==1) {
            pHead = NULL;
        }
    } else {
        pTemp =  search (index) ;
    }
    count--;
    if (pHead != NULL){
        pHead = pTemp ->next; // phan thu ngay sau ptu se xoa
```

```
    }
    pTemp->back->next=pTemp->next ;
    pTemp->next->back=pTemp->back;
    pTemp->next=NULL ;
    pTemp->back=NULL ;
    return pTemp;
}
Node* search(int index, Node* pTemp=pHead) {
    if ((index > 1) && (pTemp != NULL)) {
        index-- ;
        pTemp=search (index, pTemp -> next);
    }
    return pTemp ;
}
```

Không đệ qui:

```
Node*  remove(int index) {
    if (index > count) return NULL;
    int k = 1;
    // lap toi phan tu muon xoa
    while (k != index) {
        pHead = pHead->next;
        k++;
    }

    Node* pTemp = pHead;
    pHead = pHead -> next; // phan thu ngay sau ptu se xoa
    if (pHead != null) { // index < count
        pHead->back = pTemp->back;
    }
    if (pTemp->back != null) { // khong phai xoa phan tu dau tien
        (pTemp->back)->next = pHead;
    }
    count--;
    pTemp->next=NULL ;
    pTemp->back=NULL ;
    return pTemp;
}
```

<u>Câu 3</u>: (2 điểm)

Xét trò chơi *Josephus* như sau: *N* cá nhân hình thành một vòng tròn và một số nguyên *m*. Trò chơi sẽ từng bước loại bỏ người thứ *m* trong vòng tròn và lặp lại cho đến khi không còn người nào trong vòng tròn.

Ví dụ, đối với "*N* = 9, *m* = 4", danh sách ban đầu gồm các số và theo thứ tự : 1,2, 3,4,5,6,7,8, 9. Thứ tự loại bỏ theo trò chơi *Josephus* là: 4,8,3,9,6,5,7,2,1.

Để hiện thực bài toán này, giả sử danh sách ban đầu được lưu trữ bằng một danh sách liên kết vòng đôi (được hiện thực bằng *class List* như trong hình 1). Do vậy, chúng ta có thể hàm method *remove* như trong câu 2.

Giả sử chúng ta đã có các cấu trúc dữ liệu *stack* và *queue* đã được hiện thực cùng với các hàm sau:

boolean isEmpty(*stack s*) // kiểm tra xem *s* có rỗng hay không
Node top*(*stack s*) // trả về tham khảo của phần tử trên đỉnh của *s*
void push(*Node* x, stack s*) // đẩy phần tử *x* vào *s*
Node pop*(*stack s*) // lấy phần tử đầu tiên ra khỏi *s* và trả về tham khảo phần tử này

boolean isEmpty(*queue q*) // kiểm tra xem *q* có rỗng hay không

Node queueFront(queue q) // trả về tham khảo của phần tử đầu của q*
Node queueRear*(*queue q*) // trả về tham khảo của phần tử cuối của *q*
void enQueue(*Node* x, queue q*) // thêm phần tử *x* vào cuối hàng đợi *q*
Node deQueue*(*queue q*) // lấy phần tử đầu tiên ra khỏi *q* và trả về tham khảo đến phần tử này

a) Theo bạn nên chọn cấu trúc dữ liệu *queue* hay *stack* để nhằm phục vụ cho ngõ ra của bài toán : lưu trữ danh sách các cá nhân theo thứ tự loại bỏ trong trò chơi. Tại sao?
b) Viết thuật giải để giải bài toán này với cấu trúc dữ liệu mà bạn chọn ở câu (a). *Josephus(List l, queue q, int m)* hay *Josephus(List l, stack s, int m)*

Solution

a. Nên chọn cấu trúc dữ liệu queue, bởi vì:
   Khi lặp đi lặp lại để loại bỏ cá nhân ra khỏi vòng tròn, cá nhân nào gặp trước sẽ bị loại bỏ ra trước, tương tự tính chất FIFO của queue nên dựa vào queue kết quả, ta có thể biết được và truy xuất nhanh chóng cá nhân bị loại bỏ đầu tiên.
b. Giải thuật:

```
void Josephus(List l, queue q, int m) {
    queue tempQueue = new Queue();

    while(!l.isEmpty()) {
        enQueue(l.remove(0),tempQueue);
    }

    int i = 1;
    while(!isEmpty(tempQueue)) {
        if(i == m) {
            enQueue(tempQueue.deQueue(),q);
            i = 1;
        } else {
            i++;
            enQueue(tempQueue.deQueue(),tempQueue);
        }
    }

}
```

Câu 4: (2 điểm)

Giả sử chúng ta dùng một *queue* để lưu trữ kết quả của bài toán Josephus. Hãy viết thuật giải để khôi phục lại danh sách các phần tử theo thứ tự ban đầu (nghĩa là theo thứ tự tăng dần của giá trị *data*).
void restore(queue q) // danh sách trả về cũng được lưu trữ trong *q*
Sinh viên lớp thường có thể khai báo các biến tạm tuỳ ý khi hiện thực hàm này, sinh viên lớp KSTN **chỉ được phép khai báo thêm 1 biến tạm thuộc kiểu *Node** và các biến tạm khác phải thuộc kiểu queue**.

Solution: Bản chất là việc sắp xếp tăng dần 1 queue
Không giới hạn biến tạm.

```
void restore(Queue q) {
```

```
    Node* temp = queueFront(q);
    While(temp != queueRear(q)) {
        If(temp-> next != null) {
            Node* temp1 = temp->next;
            While(temp1 != queueRear(q)) {
                if(temp->data > temp1->data) {
                    int tempData = temp->data;
                    temp -> data = temp1->data;
                    temp1->data = tempData;
                }
            }
        }
        Temp = temp-> next;
    }
}
```

Giới hạn:

```
Void recursiveRestore(Queue q, Queue& result) {
    Queue queueTemp = new Queue();
    Node* min = enQueue(q);
    If(min == null) return ; //finished
    While(!q.isEmpty()) {
        If(queueFront(q)->data < min->data) {
            // update min
            enQueue(min,queueTemsp);
            min = deQueue(q);
        } else {
            enQueue(deQueue(q),queueTemp);
        }
    }
    // insert min to result
    enQueue(min, result);
    recursiveRestore(queueTemp,result);
}
void restore(Queue q) {
    Queue result = new Queue;
    recursiveRestore(q,result);
}
```

<u>Câu 5</u>: (1 điểm)

Hãy so sánh hai loại cấu trúc: liên kết đơn vòng và mảng vòng (*circular array*).
(Ví dụ: độ phức tạp trong trường hợp xấu nhất của việc thêm/xóa một phần tử, …)
Sinh viên lớp thường chỉ cần nêu và giải thích đúng 2 (ưu/khuyết) điểm, sinh viên lớp KSTN cần nêu và giải thích đúng 4 (ưu/khuyết) điểm.

Liên kết đơn vòng	Mảng vòng
Hiệu quả bộ nhớ vì dùng đến đâu cấp phát đến đó	Không hiệu quả vì sẽ có độ dài mảng nhất đinh mặc dù chưa có nhiều phần tử
Hiệu quả khi cấp phát bộ nhớ (Không cần resize)	Khi mảng bị đầy sẽ tốn thao tác điều chỉnh kích thước mảng
Thời gian truy xuất một phần tử trường hợp xấu nhất là 0(n) (Vì phải duyệt qua từng phần tử)	Thời gian truy xuất hằng số (dùng phép mod để truy xuất phần tử)
Thời gian xóa một phần tử nhanh hơn (chỉ thay đổi liên kết khi đã biết vị trí cần xóa)	Sau khi xóa phần tử sẽ tốn thời gian dịch chuyển phần tử trong mảng

– Hết –