BÀI TẬP TUẦN 3

Môn học: Cấu Trúc Dữ Liệu và Giải Thuật.
GV hướng dẫn thực hành: Nguyễn Khánh Toàn (ktoan271199@gmail.com).
Nội dung chính:

- Ôn tập ngăn xếp và hàng đợi.
- Tính toán độ phức tạp thuật toán.

Thời gian thực hiện: **2** tuần.

Bài tập được biên soạn lại và tham khảo từ tài liệu thực hành môn Cấu Trúc Dữ Liệu và Giải Thuật của bộ môn Công Nghệ Tri Thức, trường Đại Học Khoa Học Tự Nhiên TPHCM. Trân trọng cảm ơn quý Thầy Cô của bộ môn.

1 Ngăn xếp / Hàng đợi

1.1 Cài đặt

Các bạn sử dụng cấu trúc dữ liệu danh sách liên kết đơn (lưu ý là danh sách phải lưu lại con trỏ head và tail) đã thực hiện ở bài trước để cài đặt cấu trúc dữ liệu ngăn xếp và hàng đợi.

- Ngăn xếp hoạt động theo nguyên lý LIFO (Last in First out) - phần tử nào vào sau thì sẽ được rút ra trước.
- Hàng đợi hoạt động theo nguyên lý FIFO (First in First out) - phần tử nào vào trước thì sẽ được rút ra trước, giống như thứ tự ưu tiên khi chúng ta xếp hàng.

Các hàm số cần cài đặt:

- Khởi tạo ngăn xếp / hàng đợi từ một giá trị cho trước.
- Thêm phần tử mới (**push** hoặc **enqueue**) vào ngăn xếp và hàng đợi.
- Xóa phần tử cuối cùng (**pop** hoặc **dequeue**) của ngăn xếp và hàng đợi và trả về giá trị của phần tử đó.
- Hàm đếm (**count**) trả về số lượng phần tử trong ngăn xếp và hàng đợi.
- Hàm kiểm tra ngăn xếp / hàng đợi có đang rỗng hay không?

1.2 Bài tập vận dụng

Các bạn sử dụng cấu trúc dữ liệu ngăn xếp và hàng đợi đã cài đặt để thực hiện các bài tập sau đây.

Bài tập 1. Chuyển đổi từ trung tố (infix) sang hậu tố (postfix)

Các bạn sử dụng cấu trúc dữ liệu *ngăn xếp* để chuyển biểu thức tính toán đang ở dạng trung tố sang hậu tố, ví dụ:

$3 + 4 * (2 \wedge 5 - 1) \wedge (7 + 4/2) - 1 \rightarrow 3425 \wedge 1 - 742/ + \wedge * + 1 -$

Sở dĩ cần phải làm điều này vì biểu thức trung tố chỉ phù hợp với cách sử dụng của con người vì chúng ta có độ ưu tiên khác nhau cho từng loại toán tử. Ví dụ với biểu thức $1 - 4 * 2$, ở đây phép nhân sẽ được thực hiện trước vì ưu tiên cao hơn phép trừ. Để máy tính hiểu được điều này ta phải chuyển toán tử sang dạng hậu tố (hoặc tiền tố). Ví dụ trong biểu thức dạng hậu tố sẽ là $142 * -$, lúc này khi đọc biểu thức trái sang phải máy tính sẽ bắt gặp toán tử $*$ đầu tiên và thực hiện $4 * 2 = 8$ trước. Sau đó khi gặp toán tử $-$, máy tính sẽ thực hiện $1 - 8 = -7$.

Có thể để ý rằng, khi viết ở dạng hậu tố, toán tử cần thực hiện sẽ ở sau các toán hạng, ví dụ '84/' nghĩa là '8/4'.

Bài tập 2. Tương tự, người ta cũng định nghĩa biểu thức dạng tiền tố (prefix) như sau (ví dụ dưới đây là biểu thức chuyển từ dạng trung tố sang tiền tố):

$3 + 4 * (2 \wedge 5 - 1) \wedge (7 + 4/2) - 1 \rightarrow - + 3 * 4 \wedge - \wedge 251 + 7/421$

$1 - 4 * 2 \rightarrow -1 * 42$

Các bạn viết hàm chuyển đổi từ biểu thức dạng tiền tố (prefix) sang hậu tố (postfix) và ngược lại từ hậu tố sang tiền tố.

Ở bài tập 1 và 2, các bạn chỉ thao tác với toán hạng có 1 chữ số (ví dụ 0, 1, 2, ..., 9), và trong bài tập không xem xét đến các số âm. Toán tử chỉ xem xét là các phép toán $\wedge$, +, -, *, /.

Bài tập 3. Cho trước một xâu chỉ bao gồm các ký tự '(', ')', '{', '}', '[', ']'. Các bạn viết hàm xác định xem xâu có hợp lệ hay không? Một xâu hợp lệ sẽ phải thỏa các điều kiện sau:

- Ngoặc mở phải được đóng bằng ngoặc đóng cùng loại, ví dụ '(){}'.
- Ngoặc mở phải được đóng theo thứ tự, ví dụ '[(])' là không hợp lệ, tuy nhiên '[()], [{()}]' lại là hợp lệ.

2 Tính toán độ phức tạp thuật toán

Các bạn xem lại lý thuyết về đánh giá độ phức tạp thuật toán (dựa trên số thao tác cơ sở được thực hiện) và giải cái bài tập sau. Lưu ý rằng thao tác cơ sở bao gồm cả các phép tính cơ bản (+, -, *, /, $\wedge$, ...) nhưng ở đây chỉ yêu cầu các bạn thực hiện với các thao tác so sánh và gán.

Bài tập 1. Dưới đây là mã nguồn của hàm tính tổng bình phương của các số nguyên dương từ 1 đến n. đã được thêm các biến **count_assign** và **count_compare** để đếm số lượng thao tác cơ sở được thực hiện. Từ tổng số

lượng thao tác cơ sở có thể ước lượng được độ phức tạp thuật toán theo kích thước input (n), ví dụ với mọi $n \geq n_0$ thì tổng số lượng thao tác thực hiện là $3n + 1$. Lúc này ta có thể hiểu thuật toán có độ phức tạp là $O(n)$ vì số lượng thao tác cơ sở thực hiện tuyến tính với n.

```
int squaresum(int n, int &count_assign , int &count_compare) {
    count_assign = 0;
    count_compare = 0;

    int i = 1; ++count_assign;
    int sum = 0; ++count_assign;

    while (++count_compare && i <= n) {
        sum += i*i; ++count_assign ;
        i += 1; ++count_assign;
    }
    return sum;
}
```

Các bạn chạy lại hàm trên kiểm tra với nhiều kích thước n khác nhau, $n = \{10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000\}$. và dự đoán độ phức tạp về thời gian của hàm tính tổng trên.

Sau đó các bạn thực hiện tính toán số lượng thao tác thực hiện bằng biểu thức toán học để kiểm chứng điều này.

Bài tập 2.

```
int sum = 0;
for (int i = 1; i < n; i *= 2)
    for (int j = n; j > 0; j /= 2 )
        for (int k = j; k < n; k += 2)
            sum += (i + j * k);
```

a. Các bạn thêm 3 biến số là **count_assignment** để đếm số thao tác gán, **count_comparison** để đếm số phép so sánh thực hiện trong đoạn chương trình tính tổng trên.

Với $n = \{10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000\}$, các bạn tính tổng số thao tác cơ bản thực hiện và đưa ra kết quả dự đoán về độ phức tạp về mặt thời gian của chương trình.

b. Giả sử $n = 2^k$ với k là một số nguyên dương. Các bạn tính toán số lần thực hiện các vòng lặp bằng biểu thức toán học theo n để kiểm chứng dự đoán của mình (nếu cụ thể hơn các bạn có thể tính số lần thực hiện các thao tác cơ sở: phép so sánh, phép gán).

Bài tập 3.

```
int somesum(int n) {
    int sum = 0, i = 1, j;
    while (i <= n) {
        j = n - i;
        while (j <= i*i) {
            sum = sum + i*j;
            j += 1;
        }
        i += 1;
    }
```

```
    return sum;
}
```

a. Các bạn thêm 2 biến số là **count_assignment** để đếm số thao tác gán, **count_comparison** để đếm số phép so sánh thực hiện trong đoạn chương trình tính tổng trên.

Với $n = \{10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000\}$, các bạn tính tổng số thao tác cơ bản thực hiện và đưa ra kết quả dự đoán về độ phức tạp về mặt thời gian của chương trình.

b(*). Các bạn tính toán số lần thực hiện các thao tác gán và so sánh bằng biểu thức toán học theo n để kiểm chứng dự đoán của mình.

Hết