BÀI 16

KIỂM THỬ VÀ GỠ LỖI CHƯƠNG TRÌNH

Học xong bài này, em sẽ:

- Biết và khắc phục được một số lỗi thường gặp khi viết chương trình.
- Bước đầu thực hiện được một số truy vết đơn giản để tìm và gỡ lỗi cho chương trình Python.

Có những chương trình còn lỗi vì khi thực hiện cho ra kết quả sai. Theo em, việc biết giá trị của một số biến ngay sau khi mỗi câu lệnh được thực hiện có thể giúp tìm ra lỗi của chương trình hay không?

1 Lỗi trong chương trình và kiểm thử

1

Trong những phần trước, các bài tập và bài thực hành không quá phức tạp. Đã lần nào em soạn chương trình và thực hiện được ngay từ lần chạy đầu tiên chưa?

Chương trình chúng ta viết ra rất có thể có lỗi. Ngay cả những người lập trình giàu kinh nghiệm cũng có thể viết ra những chương trình còn lỗi. Quá trình xác định lỗi và sửa lỗi được gọi là *gỡ lỗi*. Người lập trình thường gặp các loại lỗi như sau:

- *Lỗi cú pháp* là lỗi câu lệnh viết không theo đúng quy định của ngôn ngữ, ví dụ như thiếu hoặc thừa ngoặc trong biểu thức, tên biến sai quy cách,... Loại lỗi này được môi trường lập trình phát hiện và thông báo khá cụ thể, rõ ràng, cả về đặc trưng lỗi và nơi xảy ra lỗi.
- *Lỗi ngoại lệ* (Exceptions Error) còn gọi là lỗi Runtime, là lỗi xảy ra khi chương trình đang chạy, một lệnh nào đó không thể thực hiện được. Lỗi này sẽ được thông báo ngay trên màn hình.
- *Lỗi ngữ nghĩa* (còn gọi là *lỗi logic*) là lỗi mặc dù các câu lệnh viết đúng quy định của ngôn ngữ nhưng sai trong thao tác xử lí nào đó, ví dụ như viết nhầm dấu phép tính, nhầm tên biến, gọi hàm có tham số không đúng kiểu, chỉ sai phạm vi duyệt, thiếu câu lệnh cần thiết,... Đây là loại lỗi rất khó phát hiện vì trong rất nhiều trường hợp chương trình vẫn được thực hiện trọn vẹn, nhưng kết quả đưa ra không phù hợp hoặc kết quả chỉ sai với một số bộ dữ liệu vào nào đó.

Ví dụ, xét chương trình ở *Hình 1a*, chương trình này thực hiện yêu cầu nhập vào hai số nguyên p, q và danh sách a gồm các số nguyên, sau đó đưa ra $\max\{|a_i|, i = p, p+1,..., q\}$. Biết rằng các phần tử của danh sách a được đánh chỉ số bắt đầu từ 0 và $0 \le p \le q < \texttt{len(a)}$.

File Edit Format Run Options Window Help

```
p = int(input("p = "))
q = int(input("q = "))
a = [int(i) for i in input("Dãy số: ").split()]
max = 0
for i in range(p,q):
        if abs(a[i]) > max:
            max = abs(a[i])
print("Kết quả: ", max)
```

Hình 1a. Chương trình cần kiểm thử và tìm lỗi

File Edit Shell Debug Options Window Help

```
p = 1
q = 4
Dãy số: 8 7 1 5 -6 4
Kết quả: 7
>>>
```

Hình 1b. Kết quả đúng

File Edit Shell Debug Options Window Help

```
p = 2
q = 4
Dãy số: 8 7 1 5 -6 4
Kết quả: 5
>>>
```

Hình 1c. Kết quả sai

Với đầu vào p = 1, q = 4 (*Hình 1b*), dãy con được xem xét là 7 1 5 –6, nên 7 là đáp án đúng, là số có giá trị tuyệt đối lớn nhất trong dãy con đó. Với đầu vào p = 2, q = 4 (*Hình 1c*), dãy con được xét là 1 5 –6, đáp án đúng phải là 6.

Việc đọc kĩ lại chương trình để tìm lỗi chỉ thích hợp với các chương trình ngắn, đơn giản và ngay cả trong trường hợp này cũng mất khá nhiều thời gian, công sức.

Môi trường lập trình của những ngôn ngữ lập trình bậc cao nói chung và Python nói riêng có công cụ hỗ trợ cho người dùng tìm lỗi.

Các lỗi ngữ nghĩa thì khó phát hiện hơn, chỉ có thể đoán nhận và tìm thấy thông qua quan sát kết quả thực hiện chương trình với các bộ dữ liệu vào (các bộ test) khác nhau. Những kết quả kiểm thử như vậy có thể dẫn đến việc chỉnh lí, bổ sung hoặc thay đổi thuật toán.

Khi không phát hiện thêm lỗi, ta có thể đưa chương trình vào khai thác, phục vụ mục đích thực tiễn của bài toán.

Để kiểm tra tính đúng đắn của chương trình so với yêu cầu của đề bài, trước hết cần chuẩn bị các bộ dữ liệu vào. Dữ liệu kiểm thử phải phù hợp với các ràng buộc đã cho và chia thành ba nhóm:

- Kiểm thử những trường hợp thường gặp trong thực tế.
- Kiểm thử những trường hợp đặc biệt (ví dụ, khi danh sách chỉ bao gồm một phần tử).
- Kiểm thử những trường hợp các tham số nhận giá trị lớn nhất có thể.

Dữ liệu kiểm thử ở hai nhóm đầu cần có kích thước đủ nhỏ để ta có thể kiểm chứng các kết quả do chương trình đưa ra. Dữ liệu ở nhóm thứ hai là để kiểm tra tính trọn vẹn của thuật toán trong thực hiện chương trình. Dữ liệu ở nhóm thứ ba nhằm kiểm tra tính hiệu quả của chương trình và tính hợp lí trong tổ chức dữ liệu. Có thể chương trình viết ra đã lưu quá nhiều dữ liệu trung gian nên khi dữ liệu vào có kích thước lớn thì không đủ bộ nhớ để thực hiện. Kiểm thử với dữ liệu thuộc nhóm thứ ba, ta chỉ có thể đánh giá được tính hợp lí của kết quả.

2 Truy vết với cách bổ sung câu lệnh theo dõi kết quả trung gian

2 Tại sao rất khó phát hiện lỗi nếu chỉ dùng biện pháp đọc kĩ lại chương trình?

Một cách tìm lỗi ngữ nghĩa rất hay được dùng là bổ sung vào chương trình những câu lệnh đưa ra các kết quả trung gian nhằm truy vết các xử lí của chương trình. Với cách đó, ta có thể dự đoán và khoanh vùng được phần chương trình chứa các câu lệnh đưa đến kết quả sai.

Sau khi đã chỉnh sửa xong chương trình, ta cần xoá đi các câu lệnh đã thêm để truy vết hoặc biến chúng thành dòng thông tin chú thích.

Các sai sót có thể xảy ra ngay khi nhập dữ liệu vào, vì vậy đây cũng là chỗ cần quan tâm khi tìm lỗi.

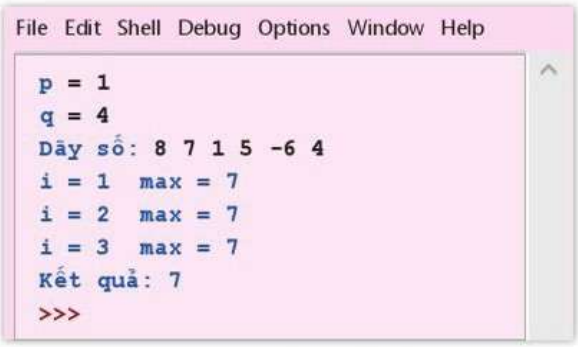
Trong ví dụ ở mục 1, hiện tượng có lúc chương trình cho kết quả sai có thể do nguyên nhân ở khâu nhập dữ liệu hoặc ở phạm vi tìm kiếm max. Để tìm xem lỗi ở đâu, ta có thể thêm câu lệnh đưa ra các phần tử tham gia tìm kiếm max, (câu lệnh `print("i = ",i,"max= ",max)`. Câu lệnh này có thể đặt trước hay sau câu lệnh đưa ra kết quả của chương trình.

File Edit Format Run Options Window Help

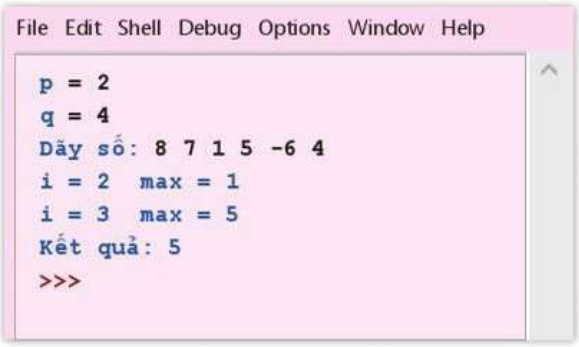
```
p = int(input("p = "))
q = int(input("q = "))
a = [int(i) for i in input("Dãy số: ").split()]
max = 0
for i in range(p,q):
        if abs(a[i]) > abs(max):
            max = abs(a[i])
        print("i =", i, "max = ", max)
print("Kết quả: ", max)
```

Câu lệnh mới thêm vào

Hình 2a. Chương trình ở Hình 1a đã thêm câu lệnh để truy vết



Hình 2b. Kết quả đúng



Hình 2c. Kết quả sai

Kết quả kiểm thử đó cho ta thấy có lỗi ở việc xác định miền cần tìm max và cần phải sửa lại câu lệnh `for i in range(p,q)` thành `for i in range(p,q + 1)`.

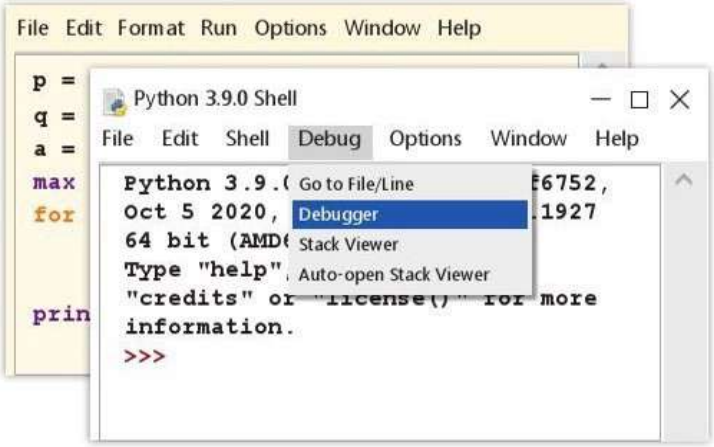
3 Truy vết với công cụ gỡ lỗi của ngôn ngữ lập trình

Phương pháp truy vết đã nêu ở mục 2 đòi hỏi phải can thiệp trực tiếp vào chương trình nguồn, thêm các câu lệnh mới và sau đó phải xoá các câu lệnh truy vết không còn cần thiết. Mỗi lần thay đổi chương trình nguồn, ta cần lưu chương trình và thực hiện lại từ đầu. Điều này bất tiện vì câu lệnh mới đưa vào cũng có thể có lỗi hoặc đưa nhầm vào vị trí không thích hợp.

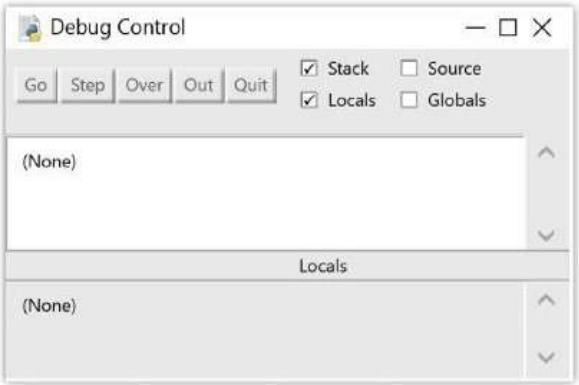
Để người lập trình không cần phải can thiệp vào chương trình nguồn mà vẫn truy vết tìm lỗi được, ngôn ngữ Python cung cấp công cụ **Debug** (Gỡ lỗi).

Để kích hoạt chế độ gỡ lỗi, cần thực hiện lần lượt các thao tác sau:

- Trên cửa sổ Shell, mở file chương trình cần gỡ lỗi, kết quả là chương trình này xuất hiện (trong cửa sổ Code).
- Chọn **Debug** trên cửa sổ Shell và sau đó chọn **Debugger** (*Hình 3*). Kết quả là cửa sổ Debug Control sẽ xuất hiện (*Hình 4*).



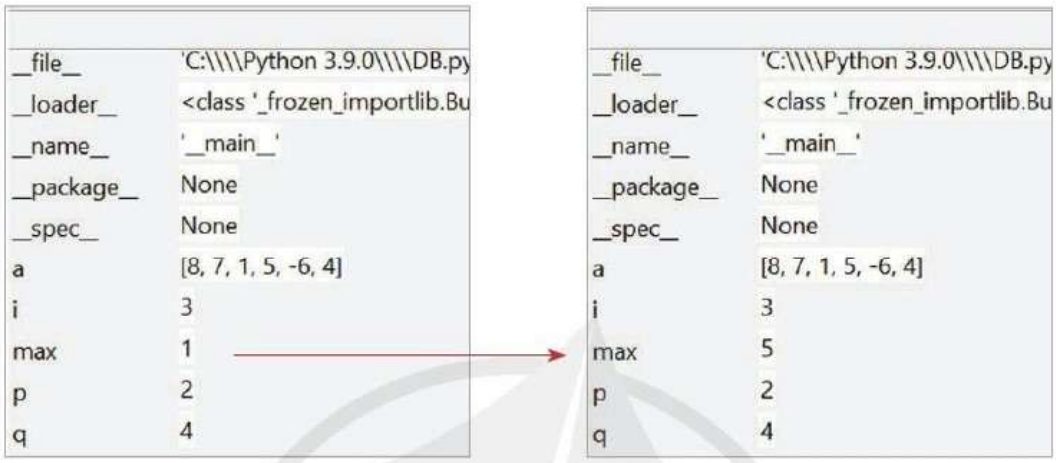
Hình 3. Kích hoạt chế độ gỡ lỗi



Hình 4. Cửa sổ Debug Cotrol của công cụ gỡ lỗi đang ở trạng thái chờ làm việc

Việc bắt đầu thực hiện chương trình vẫn được tiến hành như bình thường bằng cách chọn **Run Module** (hoặc nhấn phím **F5**) trong cửa sổ Code.

Chọn **Step** để thực hiện câu lệnh hiển thị phía dưới. Riêng các lệnh vào – ra dữ liệu ta có thể phải nháy chuột một số lần (hoặc chọn **Over** để vượt qua nhanh không cần nháy chuột nhiều lần).



Hình 5. Sự thay đổi của các biến khi một câu lệnh được thực hiện

Với chương trình có lỗi đã nói đến ở mục 1, dùng công cụ **Debug** của Python để chạy từng bước và quan sát sự thay đổi của các biến (trong cửa sổ Debug Control), ta sẽ phát hiện ra lỗi. Cụ thể là biến i chỉ chạy đến 3 là kết thúc việc tìm max, nên chương trình đưa ra max = 5 (trong khi giá trị cần đưa ra phải là 6).

Công cụ **Debug** của môi trường lập trình Python cũng cho ta đặt các điểm dừng (Breakpoint) để tránh các đoạn chương trình không cần truy vết, thực hiện nhanh đến câu lệnh ta quan tâm. Các mục còn lại trên dải lệnh của cửa sổ Debug Control cho phép bỏ qua truy vết các các câu lệnh trong vòng lặp hoặc hàm.

Nói chung truy vết để tìm lỗi là một quá trình khá khó khăn và phức tạp, đôi khi mất khá nhiều thời gian. Hỗ trợ cho những người lập trình, Python còn trang bị một thư viện riêng cung các dịch vụ gỡ lỗi, đó là thư viện PDB.

4 Thực hành gỡ lỗi cho chương trình

Bài toán: Cho a là danh sách các số nguyên. Em hãy tạo danh sách b có các phần tử ở vị trí lẻ bằng phần tử ở vị trí tương ứng của a, các phần tử ở vị trí chẵn bằng phần tử ở vị trí tương ứng của a cộng thêm 1, tức là:

$$b_i = \begin{cases} a_i + 1, & i = 0, 2, 4, \ldots \\ a_i, & i = 1, 3, 5, \ldots \end{cases}$$

So sánh số lượng các phần tử giá trị chẵn ở a với số lượng các phần tử giá trị chẵn ở b, đưa ra thông báo. Gọi p là số lượng các phần tử giá trị chẵn ở a, q là số lượng các phần tử giá trị chẵn ở b và đưa ra thông báo "a ít hơn" nếu p < q, "b ít hơn" nếu p > q và "Bằng nhau" trong trường hợp còn lại.

Nhiệm vụ: Chương trình ở *Hình 6* giải bài toán đã nêu nhưng còn có lỗi và cần được gỡ lỗi. Em hãy áp dụng các phương pháp truy vết để xác định lỗi và đề xuất cách sửa một số ít nhất các câu lệnh để có chương trình đúng.

File Edit Format Run Options Window Help

```
a = [int(i) for i in input().split()]
b = a
n = len(a)
for i in range(0,n,2):
    b[i] = b[i] + 1
p = 0, q = 0
for i in range(n):
    if a[i]%2 == 0: p = p + 1
    if b[i]&1 == 0: q = q + 1
if p < q: print("a ít hơn.")
elif p == q: print("Bằng nhau")
else:print("b ít hơn")
```

Hình 6. Chương trình cần được gỡ lỗi

Hướng dẫn:

– *Phương pháp dùng công cụ Gỡ lỗi (Debug)*

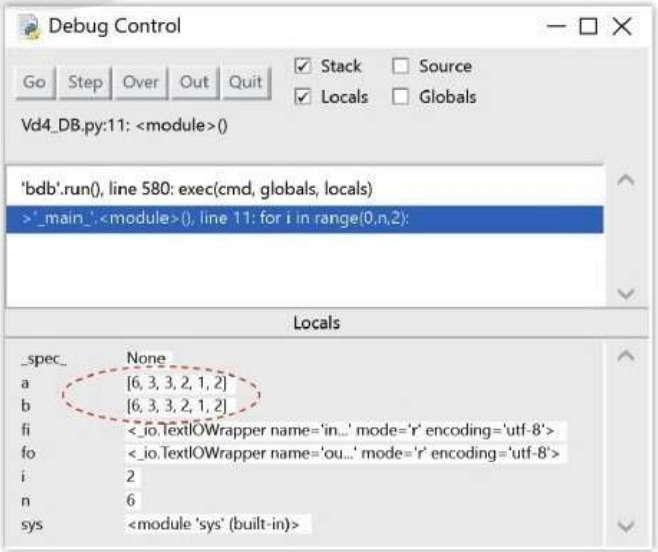
Để gỡ lỗi cần chuẩn bị một danh sách số nguyên, ví dụ: [5, 3, 2, 2, 1, 2]. Chọn **Debugger** sau đó chọn **Step** để thực hiện từng bước các câu lệnh, quan sát giá trị hai danh sách a và b. Sau một vài lần thực hiện câu lệnh trong vòng lặp:

```
for i in range(0,n,2):
b[i] = b[i] + 1
```

Ta thấy a và b đồng thời thay đổi giá trị, mặc dù trong vòng lặp chỉ chứa câu lệnh thay đổi giá trị của danh sách b.

Ví dụ sau khi i = 2 ta có kết quả như ở *Hình 7*.

Điều này nói lên rằng chương trình chưa tạo ra bản sao của danh sách a mà chỉ tạo ra một tên mới cho cùng một danh sách a. Vậy câu lệnh sai trong chương trình là `b = a`. Cần thay nó bằng câu lệnh



Hình 7. Minh hoạ kết quả của phương pháp Gỡ lỗi

`b = [] + a`. Đây là một lỗi mà nếu chỉ đọc chương trình sẽ rất khó phát hiện.

– *Phương pháp bổ sung vào chương trình các câu lệnh truy vết*

Ta có thể thêm các câu lệnh `print(a)` và `print(b)` để xuất ra giá trị các danh sách a và b sau mỗi vòng lặp. Dễ dàng nhận thấy a và b cùng đồng thời thay đổi, từ đó rút ra được kết luận như đã nêu ở phương pháp dùng công cụ Gỡ lỗi.

Em hãy soạn thảo và thực hiện từng bước chương trình ở hình sau:

File Edit Format Run Options Window Help

```
s = 0
for i in range(1,4):
    s = s + i*i
print("s = ",s)
```

Câu 1. Em hãy nêu một vài lỗi thuộc nhóm lỗi cú pháp và một vài lỗi thuộc nhóm lỗi ngữ nghĩa.

Câu 2. Tại sao phải tạo nhiều bộ dữ liệu vào khác nhau để kiểm thử chương trình?

Câu 3. Có bao nhiêu nhóm dữ liệu khác nhau cần tạo ra để kiểm thử chương trình?

Câu 4. Có thể xem giá trị các biến sau khi thực hiện một câu lệnh ở đâu?

Tóm tắt bài học

- Có ba loại bộ dữ liệu vào cần tạo để kiểm tra, đánh giá chương trình.
- Lỗi ngữ nghĩa khó phát hiện.
- Để tìm và sửa lỗi ngữ nghĩa cần dùng biện pháp truy vết.
- Muốn truy vết để tìm lỗi:
  - Có thể đưa thêm các câu lệnh xuất ra kết quả trung gian của quá trình tính toán.
  - Có thể sử dụng công cụ gỡ lỗi của môi trường lập trình.
- Trên cửa sổ Debug Control có phần hiển thị thông tin về giá trị các biến trong chương trình.