Cấu trúc dữ liệu
Chương 2 - Phần 1

Khoa CNTT, Đại học Kỹ thuật - Công nghệ Cần Thơ
www.ctuet.edu.vn

Content

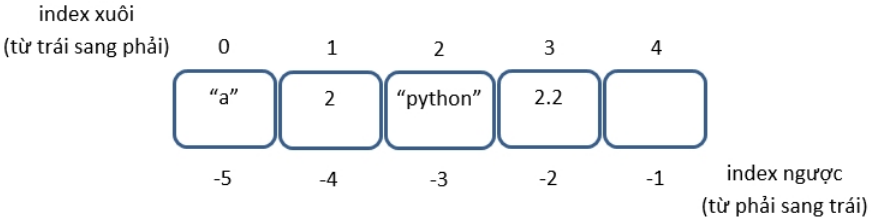
Danh sách dựng sẵn

- Danh sách trong Python có thể bao gồm hỗn hợp số, chuỗi và danh sách.

- Danh sách là đối tượng có tính thay đổi (mutable), tức là chúng ta có thể thay đổi giá trị của danh sách sau khi nó được tạo thành.

Danh sách dựng sẵn

- Một số đặc tính cơ bản của danh sách trong Python như sau:
  - Đối tượng bất kỳ thuộc nhiều kiểu dữ liệu.
  - Kiểu dữ liệu lồng nhau.
  - Ví trí truy xuất là độ lệch (offset) từ bên trái (index = 0) hoặc từ bên phải (index = -1)



Tác vụ thường dùng của danh sách

L = []	L.index(X)
L = [123, "456", "abc", 7.8, {}]	L.count(X)
L = ["a", ["b","c"], "e", 100]	L.sort(X)
L = list("python")	L.reverse(X)
L = list(range(-10,10))	L.copy()
L[i]	L.clear()
L[i][j]	L.pop(i)
L[i:j]	L.remove(X)
len(L)	del L[i]
L1 + L2	del L[i:j]
L * 5	L[i:j] = []

Tác vụ thường dùng của danh sách

L * 5	L[i:j] = []
for x in L: print(x)	L[2] = 3
5 in L	L[i:j] = [1,2,3]
L.append(8)	L = [x**2 for x in range(5)]
L.extend([6,7,8])	list(map(ord, "spam"))
L.insert(i,X)	

List comprehension and mapping

- 2 tác vụ đặc biệt trong Python.
- Tác vụ áp dụng biểu thức trực tiếp lên các đối tượng trong danh sách.
- `lst = [x * 3 for x in "python"]`

List comprehension and mapping

- 2 tác vụ đặc biệt trong Python.
- Tác vụ áp dụng biểu thức trực tiếp lên các đối tượng trong danh sách.
- `lst = [x * 3 for x in "python"]`
  #output: ['ppp', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']

List comprehension and mapping

```
1 lst = []
2 for x in "python":
3     lst.append(x*3)
```

- #output: ['ppp', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']

List comprehension and mapping

- `map` là một hàm dựng sẵn trong Python, có 2 tham số, được dùng để khai báo sử dụng 1 hàm nào (tham số thứ nhất) lên một đối tượng dữ liệu (tham số thứ hai).

```
1 def absolute_value(x):
2     return abs(x)
3
4 if __name__ == '__main__':
5     lst = [-1,-2,0,1,2]
6     lst_new = list(map(absolute_value, lst))
7     print(lst_new)
```

List comprehension and mapping

- map là một hàm dựng sẵn trong Python, có 2 tham số, được dùng để khai báo sử dụng 1 hàm nào (tham số thứ nhất) lên một đối tượng dữ liệu (tham số thứ hai).

```
1 def absolute_value(x):
2     return abs(x)
3
4 if __name__ == '__main__':
5     lst = [-1,-2,0,1,2]
6     lst_new = list(map(absolute_value, lst))
7     print(lst_new)
```

- #output: [1, 2, 0, 1, 2]

List method

- `L = ["algo","learning","pyTHON","LIST"]`
  `L.append("strings")`
  #output: ['algo', 'learning', 'pyTHON', 'LIST', 'strings']

- `L.sort()`
  #output: ['LIST', 'algo', 'learning', 'pyTHON', 'strings']

- `L.sort(reverse=True)`
  #output: ['strings', 'pyTHON', 'learning', 'algo', 'LIST']

List method

- `L.extend([1,2,3])`
  #output: ['algo', 'learning', 'LIST', 'pyTHON', 'strings', 1, 2, 3]
- `L.pop()` #output: 3
- `L`
  #output: ['algo', 'learning', 'LIST', 'pyTHON', 'strings', 1, 2]
- `L.reverse()`
  #output: [2, 1, 'strings', 'pyTHON', 'LIST', 'learning', 'algo']
- `L.index("algo")`
  #output: 6

List method

- `L.insert(2,"integers")`
  #output: [2, 1, 'integers', 'strings', 'pyTHON', 'LIST', 'learning', 'algo']
- `L.remove("integers")`
  #output: [2, 1, 'strings', 'pyTHON', 'LIST', 'learning', 'algo']
- `L.pop(2)`
  #output: 'strings'
- `L`
  #output: [2, 1, 'pyTHON', 'LIST', 'learning', 'algo']

List method

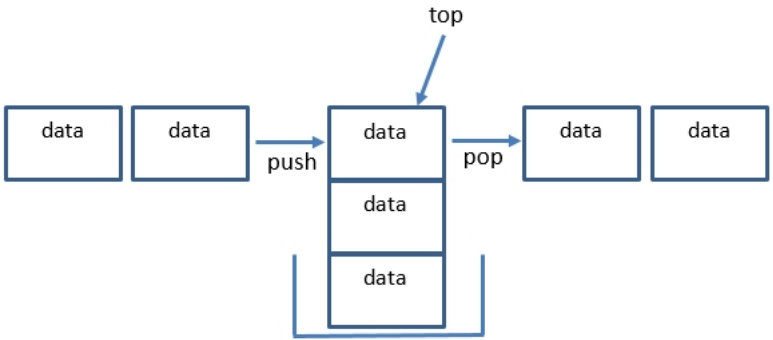
- `L.count(2)`
  #output: 1

- `del L[2:]`
  `L`
  #output: [2, 1]

Ngăn xếp

- Ngăn xếp (stacks) là tập hợp các đối tượng được thêm vào (insert) và bỏ ra (remove) theo nguyên tắc `last-in, first-out (LIFO)`.
- Người dùng có thể thêm vào ngăn xếp vào bất cứ lúc nào, nhưng chỉ có thể truy xuất và bỏ ra các đối tượng đang có trong ngăn xếp tại vị trí đỉnh (top).
  - Ý tưởng 'chồng dĩa' hay đồ vật được xếp chồng lên nhau.

Ngăn xếp



- Tác vụ cơ bản, được dùng như tên của method, là đặt vào (push) và lấy ra (pop).

Ngăn xếp

- Ngăn xếp là một cấu trúc dữ liệu trừu tượng (abstract data type (ADT)).
- Gọi S là một ngăn xếp, ta có các method cơ bản sau:
  - `S.push(e)`: thêm đối tượng e tại đỉnh của S.
  - `S.pop()`: bỏ ra đối tượng tại đỉnh của S. Lỗi phát sinh khi S rỗng.
  - `S.top()`: trả về đối tượng tại đỉnh của S nhưng không xóa đối tượng đó. Lỗi phát sinh khi S rỗng.
  - `S.is_empty()`: trả về `True` nếu danh sách rỗng, ngược lại trả về `False`.
  - `len(S)`: trả về tổng số đối tượng trong ngăn xếp hay còn gọi là kích thước của ngăn xếp.

Cài đặt ngăn xếp

- Quy ước: chúng ta tạo một ngăn xếp rỗng, không giới hạn kích thước, đối tượng thêm vào ngăn xếp thuộc kiểu bất kỳ.

- Tạo 1 file PyDev Module tên là `ListStack.py` trong Eclipse, dạng khởi tạo là `class`.

Cài đặt ngăn xếp

```
1  class ListStack(object):
2      ''' classdocs '''
3      def __init__(self):
4          ''' Constructor. Create an empty Stack '''
5          self.data = []
6
7      def __len__(self):
8          '''return the stack's size or the number of elements in the stack
       '''
9          return len(self.data)
10
11     def is_empty(self):
12         '''return True if the stack is empty otherwise return False '''
13         return len(self.data) == 0
```

Cài đặt ngăn xếp

```
14
15    def push(self, e):
16        ''' Add an element e at the top of the stack '''
17        self.data.append(e)
18
19    def pop(self):
20        ''' Remove and return the element from the top of the stack.
21        Remember the LIFO principle '''
22        if self.is_empty():
23            raise ("Stack is empty")
24        return self.data.pop()
```

Cài đặt ngăn xếp

```
25
26    def top(self):
27        ''' Return but not remove the elemnt from the top of the stack
      '''
28        if self.is_empty():
29            raise ("Stack is empty")
30        return self.data[-1]
31
32    def display(self):
33        ''' Display all elements in the stack '''
34        return self.data
```

Kiểm tra cài đặt ngăn xếp

```
1  import ListStack
2
3  if __name__ == '__main__':
4      S = ListStack.ListStack()
5      print(S.display())
6      print(S.is_empty())
7      S.push(2)
8      S.push("two")
9      S.push([5,6,7])
10     print(S.display())
11     print(S.top())
12     print(S.is_empty())
13     print(len(S))
```

Kiểm tra cài đặt ngăn xếp

```
14
15      print(S.pop())
16      print(S.pop())
17      print(S.pop())
18      print(S.pop())
```

Chuyển đổi danh sách và ngăn xếp

Stack	chuyển đổi từ List dựng sẵn
S.push(e)	L.append()
S.pop()	L.pop()
S.top()	L[-1]
S.is_empty()	len(L) == 0
len(S)	len(L)

Phân tích độ phức tạp thuật toán

Stack	Thời gian thực thi
S.push(e)	$\mathcal{O}(1)$
S.pop()	$\mathcal{O}(1)$
S.top()	$\mathcal{O}(1)$
S.is_empty()	$\mathcal{O}(1)$
len(S)	$\mathcal{O}(1)$

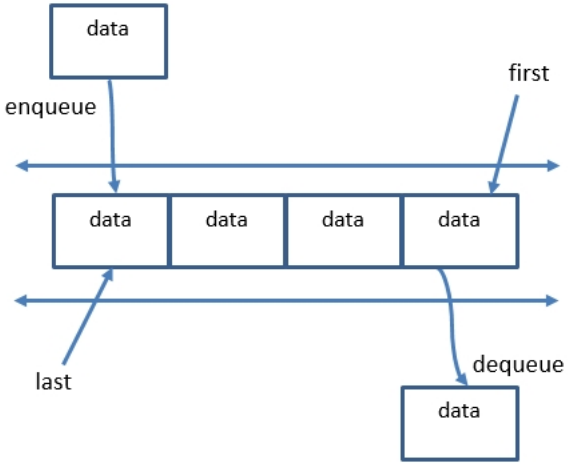
Adapter Design Pattern

- Phương pháp thiết kế cấu trúc dữ liệu trừu tượng phát triển dựa trên cấu trúc dữ liệu dựng sẵn được gọi là Adapter Design Pattern.
- Tận dụng được các tác vụ đã xây dựng sẵn mà người lập trình không cần phải mất thời gian viết lại.
- Không phụ thuộc vào ngôn ngữ lập trình cụ thể nào.
- Tư duy phát triển hướng đối tượng (OPP).

Hàng đợi

- Cấu trúc dữ liệu trừu tượng tiếp theo là hàng đợi (queue) mà trong đó các đối tượng được thêm vào (insert) và bỏ ra (remove) theo nguyên tắc `first-in, first-out (FIFO)`.
  - Người xếp hàng, in qua mạng.
- Người dùng có thể thêm vào hàng đợi bất cứ lúc nào nhưng chỉ có thể bỏ ra khỏi hàng đợi đối tượng ở trong hàng dợi lâu nhất.

Hàng đợi



Hàng đợi

- Gọi `Q` là một hàng đợi, ta có các method cơ bản sau:
  - `Q.enqueue(e)`: thêm đối tượng e vào cuối `Q`.
  - `Q.dequeue()`: bỏ ra đối tượng đầu tiên của `Q`. Lỗi phát sinh khi `Q` rỗng.
  - `Q.first()`: trả về đối tượng đầu tiên của `Q` nhưng không xóa đối tượng đó. Lỗi phát sinh khi `Q` rỗng.
  - `Q.is_empty()`: trả về `True` nếu `Q` rỗng, ngược lại trả về `False`.
  - `len(Q)`: trả về tổng số đối tượng trong hàng đợi hay còn gọi là kích thước của hàng đợi.

Cài đặt hàng đợi

- Theo quy ước, chúng ta tạo một hàng đợi rỗng, không giới hạn kích thước, đối tượng thêm vào hàng đợi thuộc kiểu bất kỳ.

- Chúng ta tạo 1 file PyDev Module tên là `ListQueue.py` trong Eclipse, dạng khởi tạo là `class`

Cài đặt hàng đợi

```
1  class ListQueue(object):
2      ''' classdocs '''
3
4      def __init__(self):
5          ''' Constructor. Create an empty queue '''
6          self.data = []
7
8      def __len__(self):
9          ''' return the stack's size or the number of elements in the
       queue '''
10         return len(self.data)
11
12     def is_empty(self):
13         ''' return True if the queue is empty otherwise return False '''
14         return len(self.data) == 0
```

Cài đặt hàng đợi

```
15
16      def first(self):
17          ''' Return but not remove the element from the top of the queue
        '''
18          if self.is_empty():
19              raise ("Queue is empty")
20          return self.data[0]
21
22      def enqueue(self, e):
23          ''' Add an element to the back of the queue '''
24          self.data.append(e)
```

Cài đặt hàng đợi

```
25
26    def dequeue(self):
27        ''' Remove and return the element from the top of the queue.
28        Remember the FIFO principle '''
29        if self.is_empty():
30            raise ("Queue is empty")
31        element = self.data[0]
32        del self.data[0]
33        return element
34
35    def display(self):
36        ''' Display all elements in the queue '''
37        return self.data
```

Kiểm tra cài đặt hàng đợi

- Tạo 1 file chương trình và thực hiện một số tác vụ để kiểm tra hàng đợi đã xây dựng:

```
1  import ListQueue
2
3  if __name__ == '__main__':
4      Q = ListQueue.ListQueue()
5
6      print(len(Q))
7      print(Q.is_empty())
8
9      Q.enqueue(2)
10     Q.enqueue("two")
11     Q.enqueue(3)
12     print(Q.is_empty())
13     print(Q.display())
```

Kiểm tra cài đặt hàng đợi

```
14
15      print(Q.first())
16      print(Q.display())
17
18      Q.dequeue()
19      print(Q.display())
20
21      Q.enqueue("three")
22      print(Q.display())
23
24      Q.dequeue()
25      print(Q.display())
26      Q.dequeue()
27      print(Q.display())
28      Q.dequeue()
29      print(Q.display())
```

Chuyển đổi giữa Queue ADT là List

Queue	chuyển đổi từ danh sách dựng sẵn
Q.enqueue(e)	L.append()
Q.first()	L[0]
Q.dequeue()	del L[0]
Q.is_empty()	len(L) == 0
len(Q)	len(L)

Phân tích độ phức tạp thuật toán

Queue	Thời gian thực thi
Q.enqueue(e)	$\mathcal{O}(1)$
Q.first()	$\mathcal{O}(1)$
Q.dequeue()	$\mathcal{O}(1)$
Q.is_empty()	$\mathcal{O}(1)$
len(Q)	$\mathcal{O}(1)$

Danh sách liên kết đơn

- Danh sách liên kết đơn (singly linked list) là một tập hợp các nút (node) liên kết với nhau theo mối quan hệ tịnh tiến theo chuỗi (linear sequence).
- Mỗi nút lưu bao gồm dữ liệu và tham chiếu đến một nút khác tiếp theo trong danh sách liên kết đơn.
- Một nút có 2 thành phần: `element` để chưa dữ liệu và `next` để chưa con trỏ hay liên kết đến nút tiếp theo.

Danh sách liên kết đơn

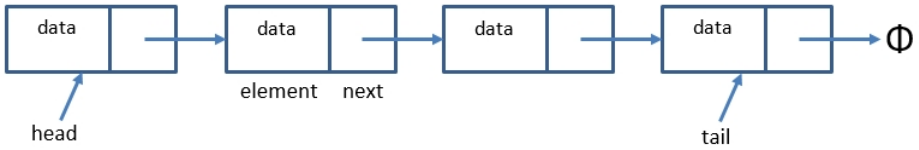
- Nút đầu danh sách được gọi là `head`, nút cuối danh sách được gọi là `tail`.
- Bắt đầu một danh sách liên kết đơn luôn là nút `head`, chúng ta đi tịnh tiến các nút tiếp theo trong danh sách thông qua con trỏ `next` đến nút cuối cùng là `tail`.
  - Đi ngang qua (traversing) một danh sách liên kết.
- Nút cuối danh sách chưa dữ liệu và tham chiếu đến Φ hoặc `None`, hiểu là tập rỗng, hay kết thúc của danh sách.

Cài đặt Node

- Biểu diễn `Node` bằng một `class`, tên file là `Node.py`.

```
1 class Node(object):
2
3     def __init__(self, element, next):
4         ''' Constructor '''
5         self.element = element
6         self.next = next
```

Danh sách liên kết đơn

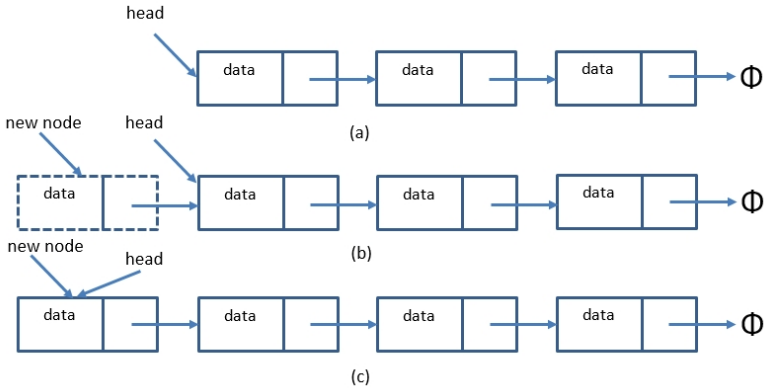


- Một tính chất quan trọng của danh sách liên kết đơn chính là nó không cần hệ thống cung cấp 1 vùng nhớ liên tục khi khởi tạo.
- Thông tin quan trọng nhất là `head`, là nơi bắt đầu của danh sách, và tổng số nút hiện có trong danh sách.

Thêm đối tượng đầu danh sách

1. Khởi tạo một nút mới, đặt tham chiếu của nút mới đến nút `head`.
2. Thay đổi giá trị của nút `head` là nút mới.
3. Tăng kích thước của danh sách lên 1.

Thêm đối tượng đầu danh sách



Hình: (a) trước khi thêm, (b) khởi tạo đối tượng mới, (c) hoàn tất quá trình thêm đối tượng mới.

Thêm đối tượng đầu danh sách

- Gọi L là một danh sách liên kết, các nút trong danh sách là instance của class `Node`.

Algorithm 1 Thêm đối tượng đầu danh sách

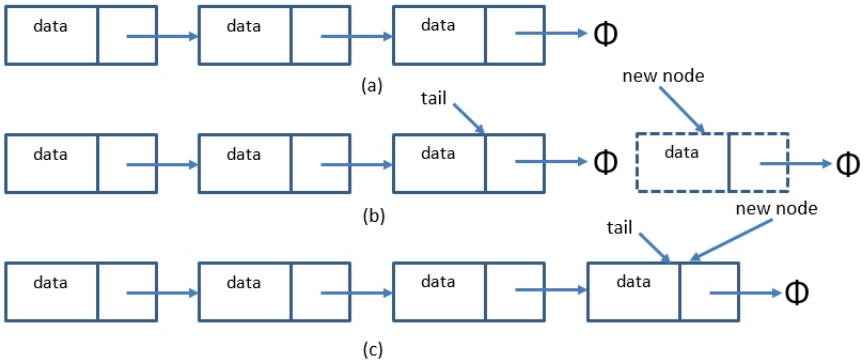
INPUT: L, e
OUTPUT: Thêm e vào đầu của L.

1: new_node = Node(e)
2: new_node.next = L.head
3: L.head = new_node
4: L.size += 1

Thêm đối tượng cuối danh sách

- Khởi tạo một nút mới, đặt tham chiếu của nút mới đến `None`.
- Thay đổi tham chiếu của nút `tail` đến nút mới.
- Tăng kích thước của danh sách lên 1.

Thêm đối tượng cuối danh sách



Hình: (a) trước khi thêm, (b) khởi tạo đối tượng mới, (c) hoàn tất quá trình thêm đối tượng mới.

Thêm đối tượng cuối danh sách

Gọi L là một danh sách liên kết, các nút trong danh sách là instance của class `Node`.

Algorithm 2 Thêm đối tượng vào cuối danh sách

INPUT: L, e

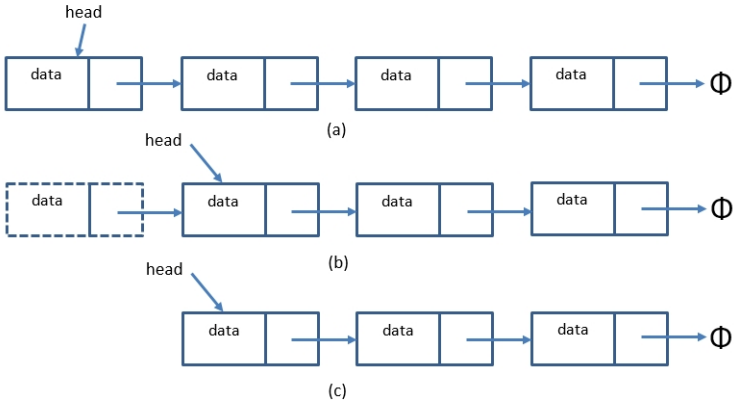
OUTPUT: Thêm e vào cuối của L.

1: new_node = Node(e)
2: new_node.next = `None`
3: L.tail.next = new_node
4: L.tail = new_node
5: L.size += 1

Xóa đối tượng ở đầu danh sách

1. Là tác vụ ngược lại với thêm một đối tượng ở đầu danh sách.
2. Kiểm tra danh sách có rỗng hay không? Nếu danh sách rỗng, tác vụ xóa không cần thực hiện.
3. Giảm kích thước của danh sách đi 1.

Xóa đối tượng ở đầu danh sách



Hình: (a) trước khi xóa, (b) thay đổi giá trị `head`, (c) hoàn tất quá trình xóa đối tượng.

Xóa đối tượng ở đầu danh sách

- Gọi L là một danh sách liên kết, các nút trong danh sách là instance của class `Node`.

Algorithm 3 Xóa đối tượng ở đầu danh sách.

INPUT: L

OUTPUT: L đã xóa đối tượng đầu.

```
1: if L.head == None then
2:     thông báo lỗi: danh sách rỗng
3: end if
4: L.head = L.head.next
5: L.size -= 1
```

Xóa đối tượng ở cuối danh sách

- Về nguyên tắc chúng ta cũng có thể thực hiện tác vụ xóa đối tượng ở cuối dang sách.
- Khó thực hiện hiệu quả ngay cả khi chúng ta có lưu tham chiếu đến `tail` của danh sách.
  - Chúng ta phải biết nút liền trước của nút `tail`.
  - Phải duyệt tuần từ từ `head`.
- Đây là điểm hạn chế của danh sách liên kết đơn.