

队列-3

? a_to_b.py

- 案例

- 案例-3：数字变换

- 概述

对于一对正整数a, b, 对a只能进行加1, 减1, 乘2操作, 问最少对a进行几次操作能得到b?

例如:

a = 3, b = 11: 可以通过 $3 * 2 * 2 - 1$, 3次操作得到 11.

a = 5, b = 8 : 可以通过 $(5 - 1) * 2$, 2次操作得到8.

题目

本题用广度优先搜索, 寻找a到b状态迁移最短路径.

对于每个状态s, 可以转换到状态s+1, s-1, s*2.

1.把初始状态a入队

2.出队一个状态s, 然后把s+1, s-1, s*2入队

3.反复循环(2), 直到状态s为b.

思路

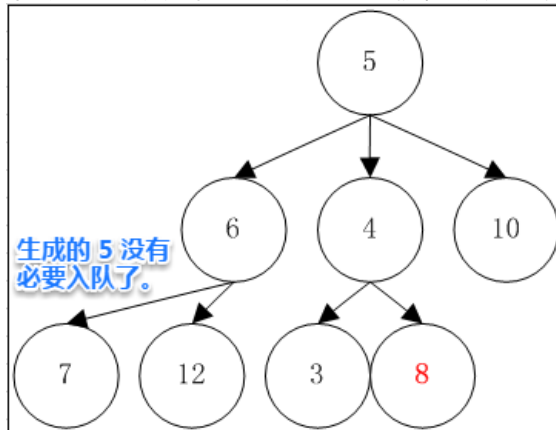
atob.py

源码

- 思路

- 如何记录计算次数? 通过 (数字, 计算次数) 元组这种数据结构实现。

- 为什么要设置集合 (老师实现)? 下图是一个 5 to 8 的过程, 5 之所以要 +1, -1, *2 就是说明 5 不等于 8, 不满足条件。所以当 6 - 1 又得到 5 的时候, 5 就没有必要入队再次进行与 8 的比较了。



- 代码-0：自己实现 (有问题)

```
# coding:utf-8
```

```
from collections import deque
```

```
def a_to_b(a, b):
```

```
    i = 0
```

```
    q = deque()
```

```
    cur = (a, i)
```

```
    while not cur[0] == b:
```

```
        q.append((cur[0] + 1, i + 1))
```

```
        q.append((cur[0] - 1, i + 1))
```

```
        q.append((cur[0] * 2, i + 1))
```

```
        cur = q.popleft()
```

```
    return cur[1]
```

```
if __name__ == "__main__":
```

```
    print a_to_b(3, 11)
```

1 # 输出结果永远是 1。问题出在高亮处, i 的初值永远是 0 所以不管多少次循环 + 1 后永远是 1

```
-----
-----
# coding:utf-8

from collections import deque

def a_to_b(a, b):
    i = 0
    q = deque()
    cur = (a, i)

    while not cur[0] == b:
        i = i + 1
        q.append((cur[0] + 1, i))
        q.append((cur[0] - 1, i))
        q.append((cur[0] * 2, i))
        cur = q.popleft()
    return cur[1]

if __name__ == "__main__":
    print a_to_b(3, 11)
```

13 # 从 3 计算到 11 只需要 3 次计算这里却是 13。问题出在高亮处

队列: [(4, 1), (2, 1), (6, 1)]

出队: (4, 1)

队列: [(2, 1), (6, 1), (5, 2), (3, 2), (8, 2)]

出队: (2, 1)

队列: [(6, 1), (5, 2), (3, 2), (8, 2), (3, 3), (1, 3), (4, 3)]

2 这个数字到这一步骤应该计算 2 次但是这里却显示了 3 错误原因在于, 所有“结构”公用 i 这一个变量记录计算次数

■ 代码-1: 自己实现

```
# coding:utf-8

from collections import deque

def a_to_b(a, b):
    # 用元组这种数据结构表示 (计算结果, 层数)
    # 通过层数得知通过几次计算由 a 得到 b
    q = deque()
    cur = (a, 0)

    # 如果计算结果不等于 b, 再次计算 计算结果的+1、-1、*2 入队
    while not cur[0] == b:
        # 使用各自的变量存储计算次数
        q.append((cur[0] + 1, cur[1] + 1))
        q.append((cur[0] - 1, cur[1] + 1))
        q.append((cur[0] * 2, cur[1] + 1))
        cur = q.popleft()
    return cur[1]

if __name__ == "__main__":
    print u'通过', a_to_b(3, 11), u'次计算实现!'
```

■ 代码-2: 老师实现

```
# coding:utf-8
```

```

from collections import deque

def a_to_b(a, b):

    q = deque([(a, 0)])
    # 通过集合 check 判断, 如果通过 +1、-1、*2 生成的元素之前入队过
    # ( 经过与 b 的判断, 入队过现在还在生成新数, 说明该数不等于 b )
    # 就不要再次入队
    check = {a}

    # 队列出队元素是元组, 将元组拆包
    # num 获得元组第一元素
    # count 获得元组第二个元素
    num, count = q.popleft()

    while not num == b:
        # 新生成的数不在集合中, 才添加到队列
        if not num + 1 in check:
            q.append((num + 1, count + 1))
            check.add(num + 1)
        # 新生成的数不在集合中, 才添加到队列
        if not num - 1 in check:
            q.append((num - 1, count + 1))
            check.add(num - 1)
        # 新生成的数不在集合中, 才添加到队列
        if not num * 2 in check:
            q.append((num * 2, count + 1))
            check.add(num * 2)
        num, count = q.popleft()

    return count

if __name__ == "__main__":
    print a_to_b(3, 11)

# 没有集合判断
deque([(11, 3), (24, 3), (7, 4), (5, 4), (12, 4), (5, 4), (3, 4), (8, 4), (11, 4), (9, 4), (20, 4), (5, 4), (3, 4), (8, 4), (3, 4), (1, 4), (4, 4), (7, 4),
(5, 4), (12, 4), (10, 4), (8, 4), (18, 4), (8, 4), (6, 4), (14, 4), (17, 4), (15, 4), (32, 4), (5, 4), (3, 4), (8, 4), (3, 4), (1, 4), (4, 4), (7, 4), (5, 4),
(12, 4), (3, 4), (1, 4), (4, 4), (1, 4), (-1, 4), (0, 4), (3, 4), (1, 4), (4, 4), (6, 4), (4, 4), (10, 4), (4, 4), (2, 4), (6, 4), (9, 4), (7, 4), (16, 4), (9, 4),
(7, 4), (16, 4), (7, 4), (5, 4), (12, 4), (15, 4), (13, 4), (28, 4), (7, 4), (5, 4), (12, 4), (5, 4), (3, 4), (8, 4), (11, 4), (9, 4), (20, 4), (14, 4), (12,
4), (26, 4)])
# 有集合判断, 可以看出队列中元素显著的少
deque([(11, 3), (24, 3), (20, 4), (18, 4), (17, 4), (15, 4), (32, 4), (-1, 4), (28, 4), (26, 4)])

```

o

- And So On