数据结构和算法
（Python描述）

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

队列和广度优先搜索

队列的概念和实现

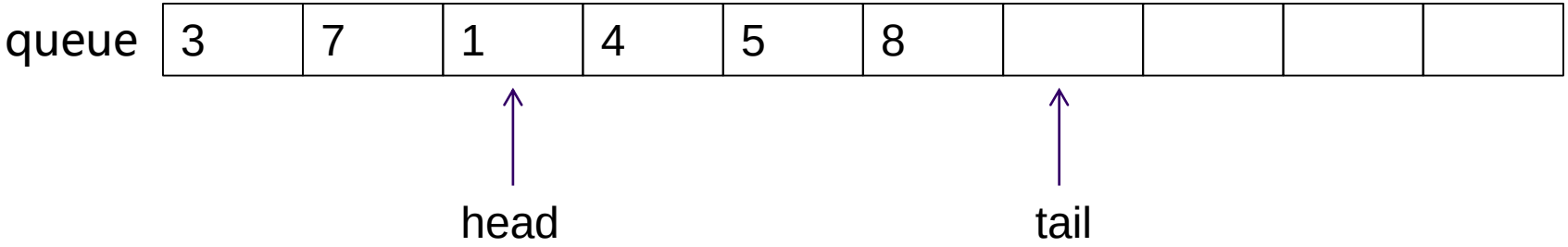
美国黄石公园

队列的概念

- 即排队的队列。只能一头进(push)，另一头出(pop)。先进先出
- 要求进出的复杂度都是O(1)
- 如果用列表的append进，pop(0)出，则出的复杂度为O(n)

队列的实现方法一

用足够大的列表实现，维护一个队头指针和队尾指针，初始：head=tail = 0



- head指向队头元素，tail指向队尾元素的后面
- push(x)的实现：

```
queue[tail] = x
tail+=1
```

- pop()的实现：

```
head += 1
```

- 判断队列是否为空：

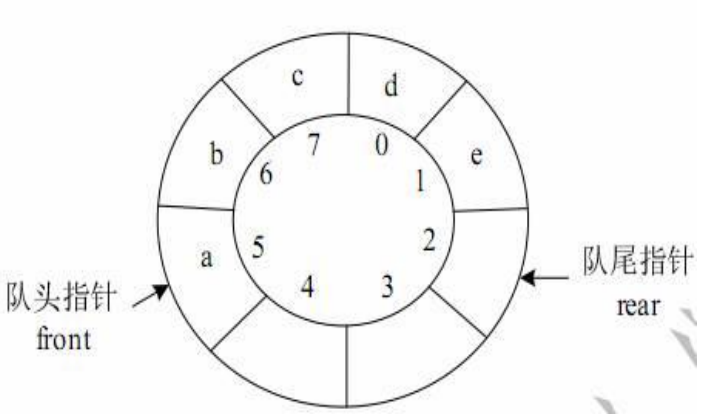
```
head == tail
```

队列的实现方法二

如果不想浪费空间开足够大的列表，而是想根据实际情况分配空间，则可以用列表+头尾循环法实现队列

1) 预先开设一个capacity个空元素的列表queue，head = tail = 0
2) 列表没有装满的情况下：

- push(x)的实现：
  queue[tail] = x
  tail = (tail+1) % capacity
- pop()的实现：
  head = (head+1) % capacity
- 判断队列是否为空：
  head == tail



capacity可以是4,8,16.....

队列的实现方法二

3) 若一个push操作后导致列表满：

1. 建一个大小是原列表k倍大的新列表(k>1,可以取1.5,2.....)
2. 将原列表内容全部拷贝到新列表，作为新队列
3. 重新设置新列表的head和tail
4. 原列表空间自动被Python解释器回收

- 导致队列满的push的时间复杂度是O(n)。平均push操作是O(1)
- Python列表append做到O(1)的实现也是这种原理,且k取1.125，空间换时间
- 若每次增加空间只增加固定数量，比如20个单元，则push平均复杂度还是O(n)

队列的实现方法二

```
class queue:
    initC = 4       #队列初始容量
    def __init__(self):
        self.q = [0 for i in range(queue.initC)]
        self.capacity = queue.initC   #容量
        self.size = 0                 #有效元素个数
        self.head = self.tail = 0
    def empty(self):
        return self.head == self.tail
    def front(self):   #看队头。空队列导致re
        return self.q[self.head]
    def back(self):    #看队尾空队列导致re
        if self.tail > 0:
            return self.q[self.tail - 1]
        else:
            return self.q[-1]
```

```
def push(self,x):
    self.q[self.tail] = x
    self.size += 1
    if self.size == self.capacity:
        tmp = [0 for i in range(self.capacity*2)]
        k = 0
        while self.head != self.tail:
            tmp[k] = self.q[self.head]
            self.head += 1
            if self.head == self.capacity :
                self.head = 0
            k += 1
        self.q = tmp
        self.q[k] = x
        self.head,self.tail = 0,k+1
        self.capacity *=2
    else:
        self.tail += 1
        if self.tail == self.capacity:
            self.tail = 0
```

队列的实现方法二

```
    def pop(self):
        if self.size == 0:
            return None          #也可以不要这两行，让空队列导致re
        result = self.q[self.head]
        self.head += 1
        if self.head == self.capacity:
            self.head = 0
        self.size -= 1
        if self.size == 0:
            self.q = [0 for i in range(queue.initC)]
            self.capacity = queue.initC
            self.head = 0
            self.tail = 0
        return result
q = queue()
for i in range(1,14):
    q.push(i)
a = q.pop()
```

Python中的队列

collections库中的deque是双向队列，可以像普通列表一样访问，且在两端进出，复杂度都是O(1)

```
import collections
dq = collections.deque()
dq.append('a') #右边入队
dq.appendleft(2) #左边入队
dq.extend([100,200]) #右边加入100,200
dq.extendleft(['c','d']) #左边依次加入 'c','d'
print(dq.pop()) #>>200 右边出队
print(dq.popleft()) #>>d 左边出队
print(dq.count('a')) #>>1
dq.remove('c')
print(dq)       #>>deque([2, 'a', 100])
dq.reverse()
print(dq)       #>>deque([100, 'a', 2])
print(dq[0],dq[-1],dq[1]) #>>100 2 a
print(len(dq)) #>>3
```

广度优先搜索
例题: 抓住那头牛

美国黄石公园

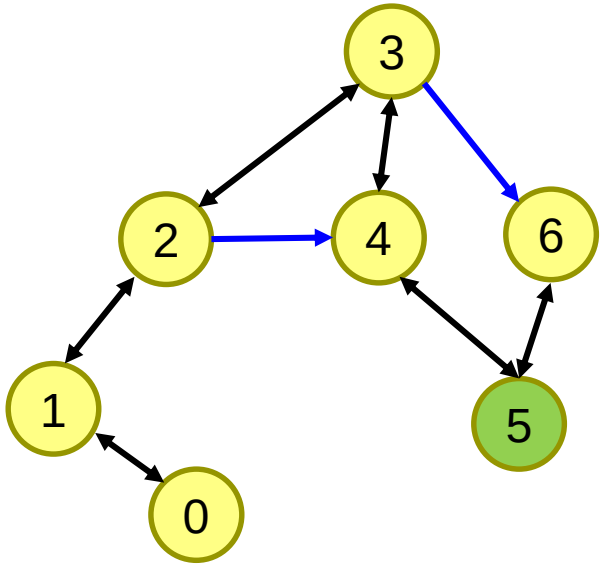
抓住那头牛(百练习4001)

农夫知道一头牛的位置，想要抓住它。农夫和牛都位于数轴上，农夫起始位于点N(0<=N<=100000)，牛位于点K(0<=K<=100000)。农夫有两种移动方式：

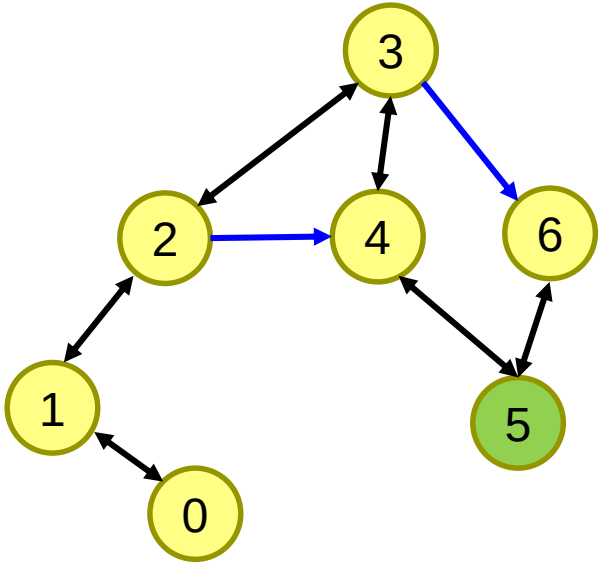
1、从X移动到X-1或X+1，每次移动花费一分钟
2、从X移动到2*X，每次移动花费一分钟

假设牛没有意识到农夫的行动，站在原地不动。农夫最少要花多少时间才能抓住牛？

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?

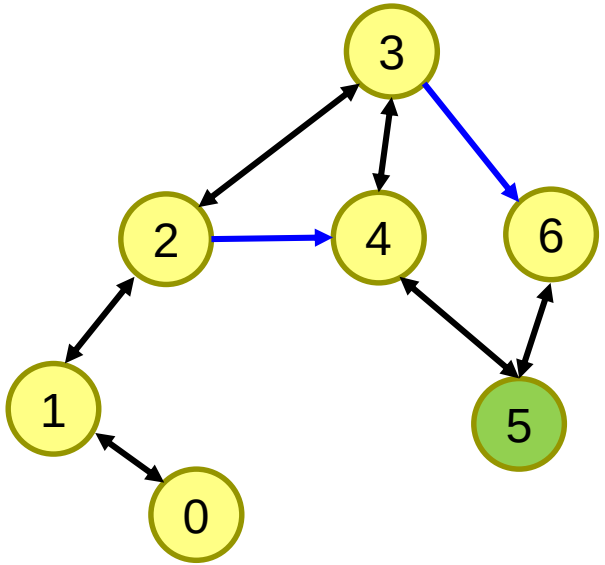


假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



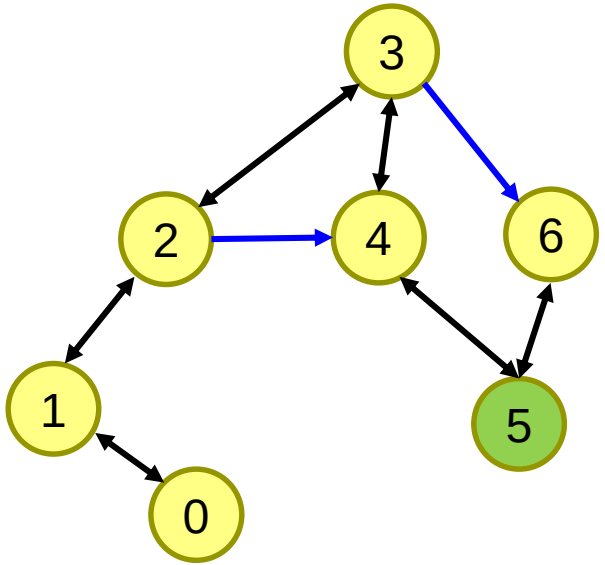
策略1）深度优先搜索：从起点出发，随机挑一个方向，能往前走就往前走(扩展），走不动了则回溯。不能走已经走过的点(要判重）。

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



运气好的话:
3->4->5
或
3->6->5
问题解决!

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?

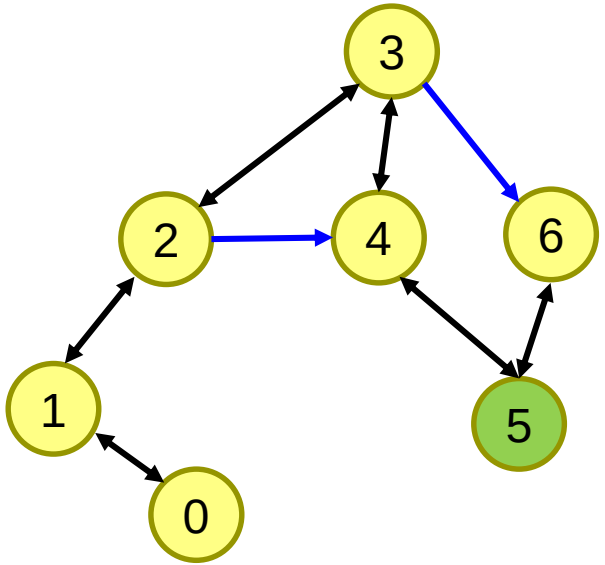


运气不太好的话：
3->2->4->5
运气最坏的话：
3->2->1->0->4->5

要想求最优(短）解，则要遍历所有走法。可以用各种手段优化，比如，若已经找到路径长度为n的解，则所有长度大于n的走法就不必尝试。

运算过程中需要存储路径上的节点，数量较少。
用栈存节点。

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?

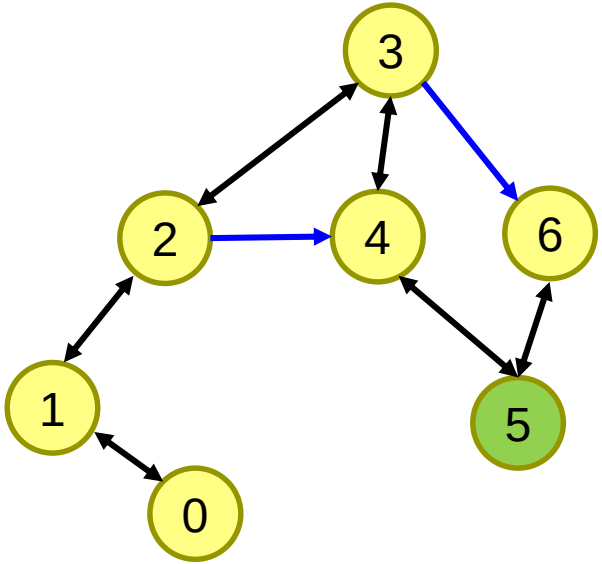


策略2）广度优先搜索：

给节点分层。起点是第0层。从起点最少需n步就能到达的点属于第n层。

第1层：2,4,6
第2层：1,5
第3层：0

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?

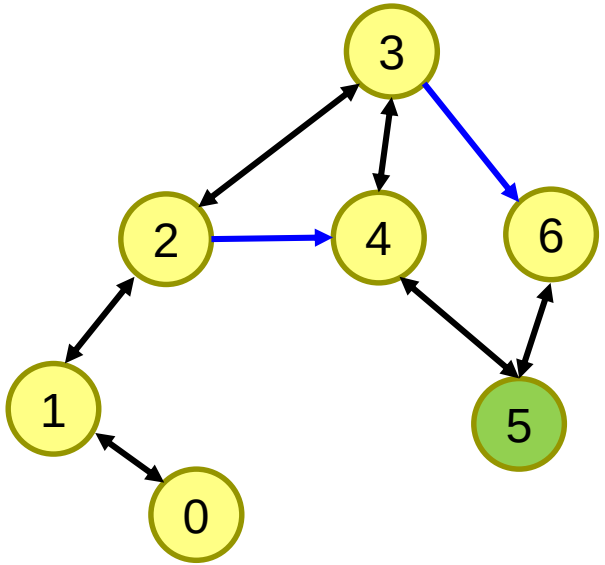


策略2）广度优先搜索：

给节点分层。起点是第0层。从起点最少需n步就能到达的点属于第n层。

依层次顺序，从小到大扩展节点。把层次低的点全部扩展出来后，才会扩展层次高的点。

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



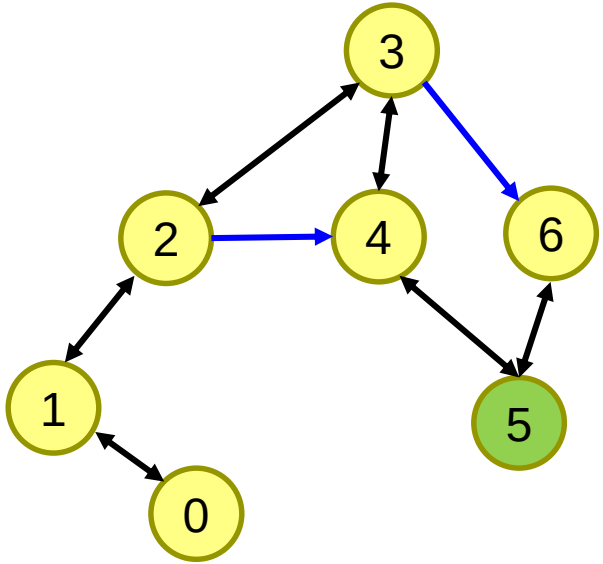
策略2）广度优先搜索：

搜索过程（节点扩展过程）：

3
2 4 6
1 5

问题解决。
扩展时，不能扩展出已经走过的节点(要判重) 。

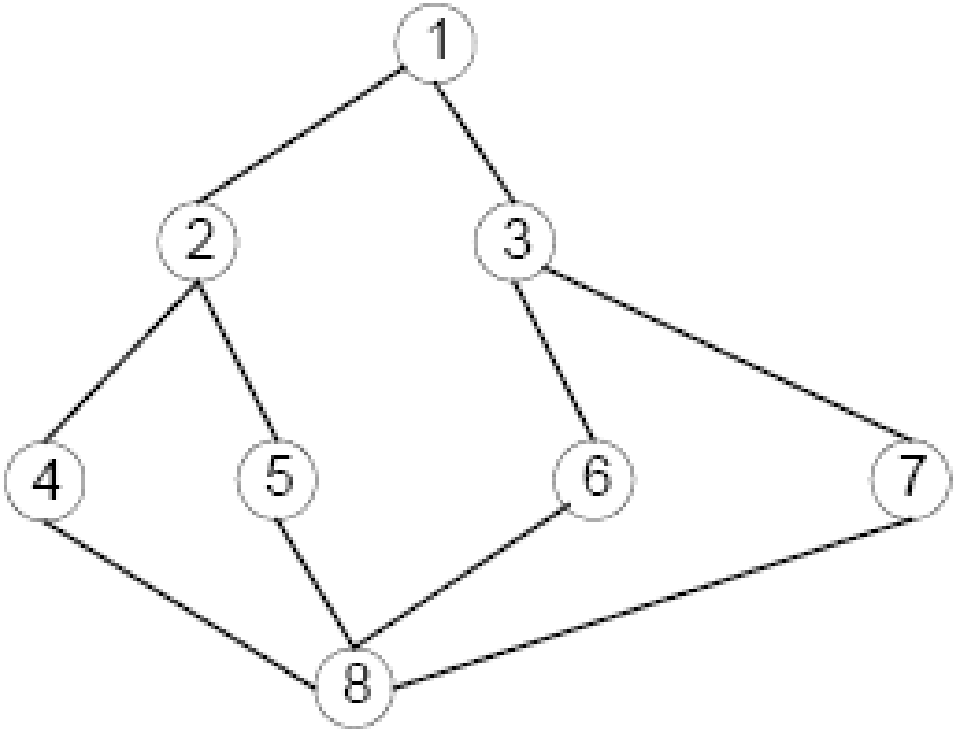
假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



策略2）广度优先搜索：

可确保找到最优解，但是因扩展出来的节点较多，且多数节点都需要保存，因此需要的存储空间较大。
用队列存节点。

深搜 vs. 广搜



若要遍历所有节点：

- 深搜

1-2-4-8-5-6-3-7

- 广搜

1-2-3-4-5-6-7-8

广搜算法

□广度优先搜索算法如下：（用QUEUE）

(1) 把初始节点S0放入Open表中；

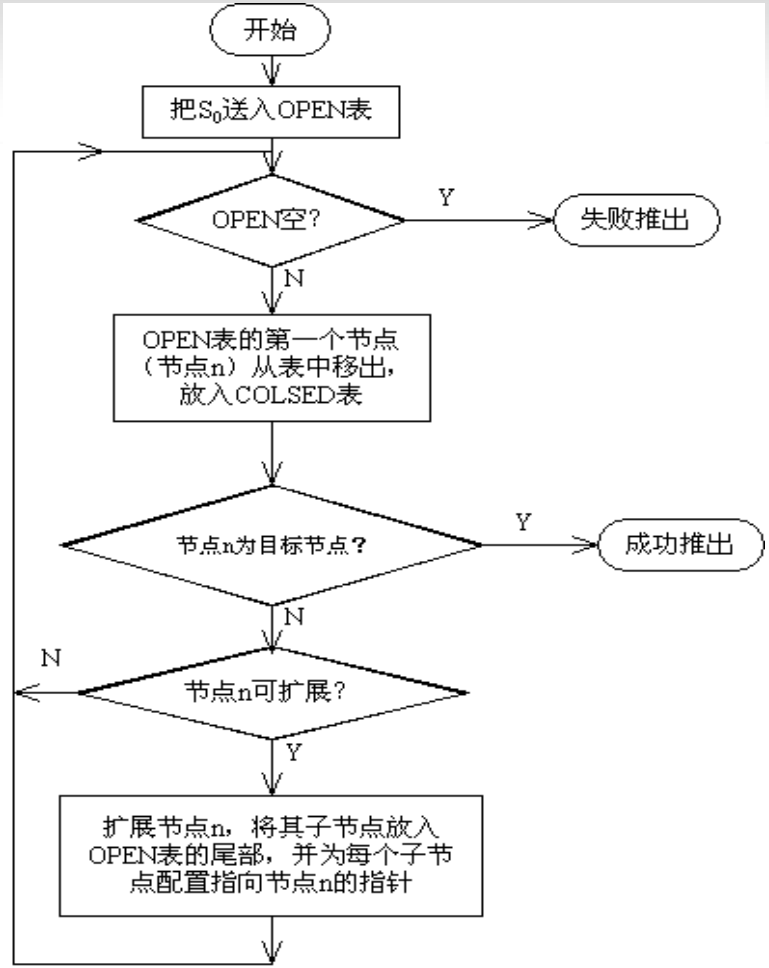
(2) 如果Open表为空，则问题无解，失败退出；

(3) 把Open表的第一个节点取出放入Closed表，并记该节点为n；

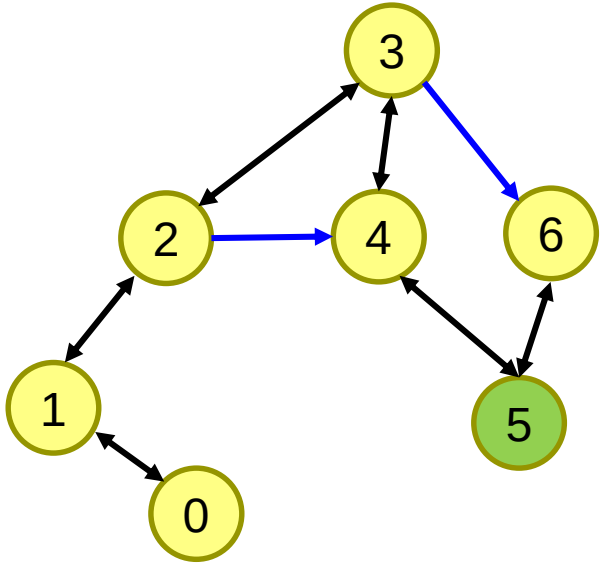
(4) 考察节点n是否为目标节点。若是，则得到问题的解，成功退出；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，将其不在Closed表和Open表中的子节点(判重）放入Open表的尾部，并为每一个子节点设置指向父节点的指针(或记录节点的层次），然后转第(2)步。



假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



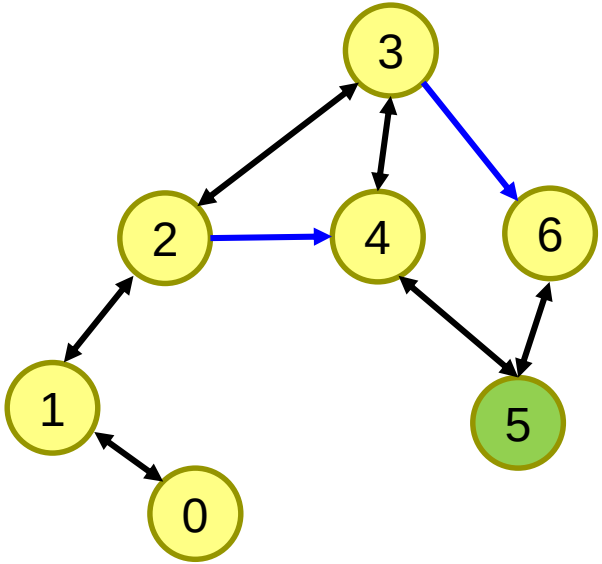
广度优先搜索队列变化过程：

Closed

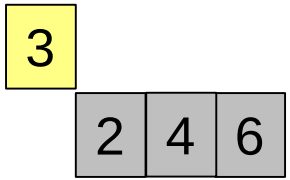
Open

3

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



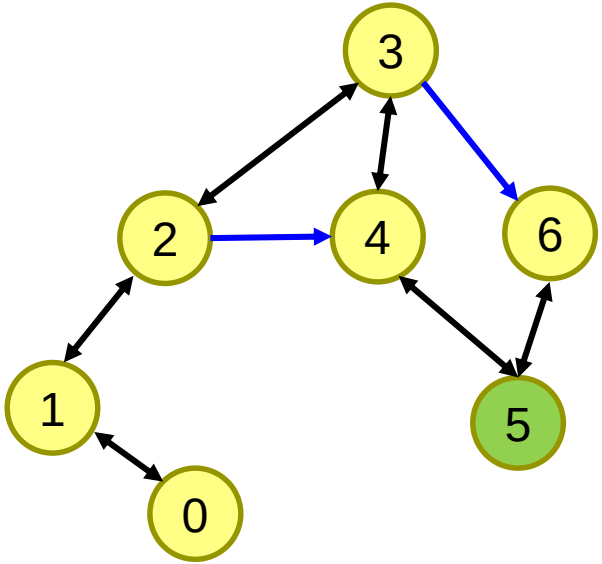
广度优先搜索队列变化过程：



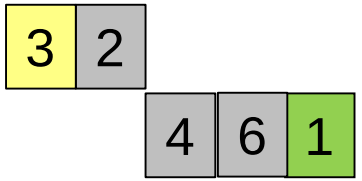
Closed

Open

假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



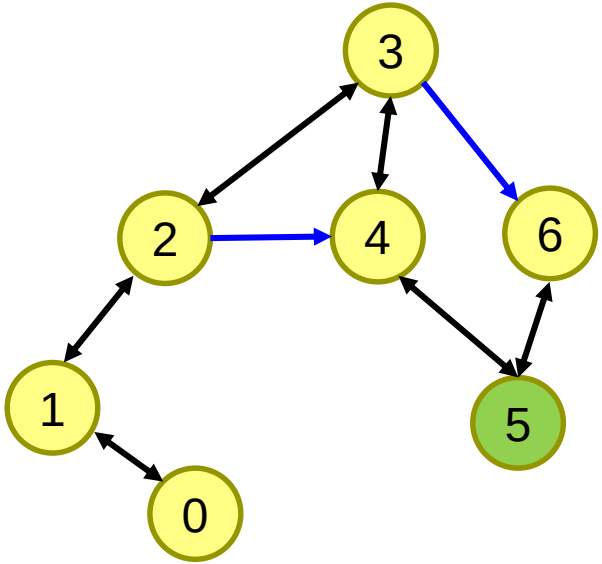
广度优先搜索队列变化过程：



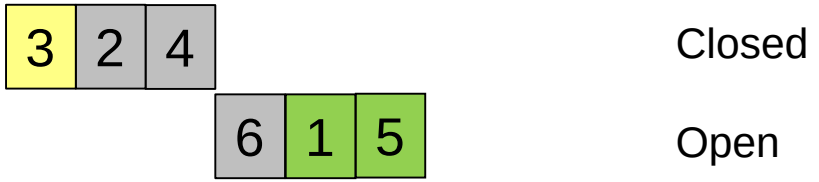
Closed

Open

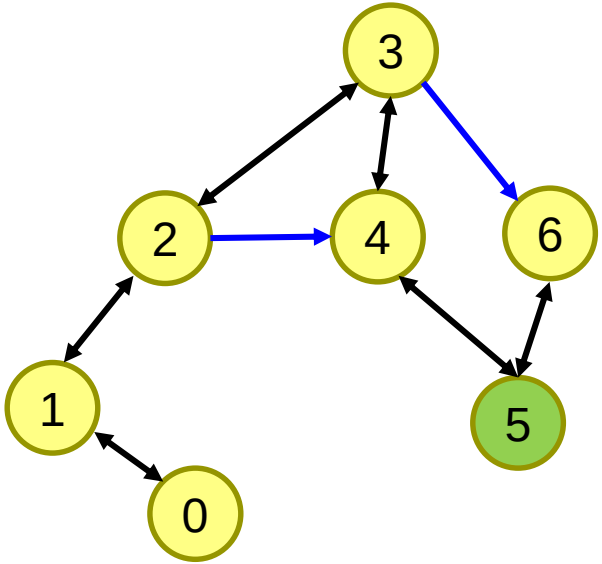
假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



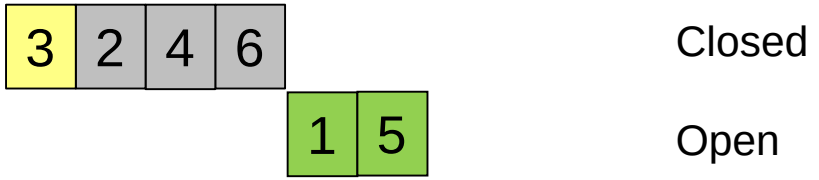
广度优先搜索队列变化过程：



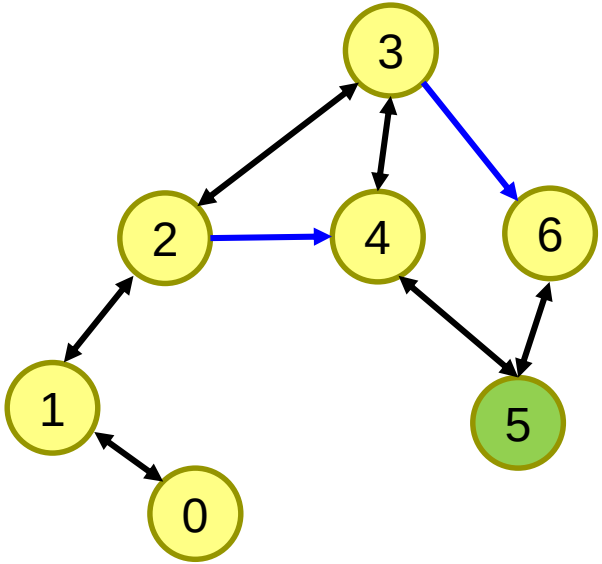
假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



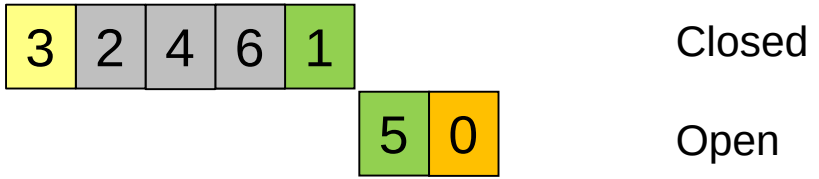
广度优先搜索队列变化过程：



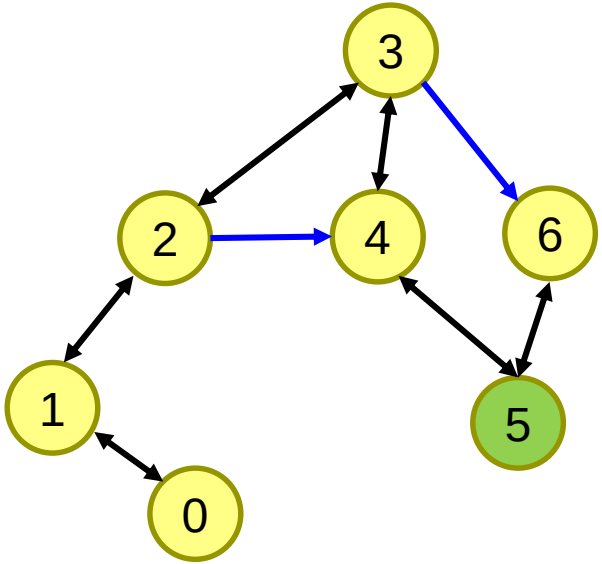
假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



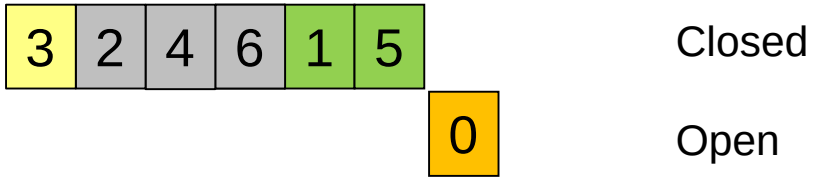
广度优先搜索队列变化过程：



假设农夫起始位于点3，牛位于5
N=3,K=5，最右边是6。
如何搜索到一条走到5的路径?



广度优先搜索队列变化过程：



目标节点5出队列，问题解决！

#poj3278 Catch That Cow

```
import collections
class step:
    def __init__(self,x,steps):
        self.x = x          #位置
        self.steps = steps    #到达x所需的步数
MAXN = 100000
N,K = map(int,input().split())
q = collections.deque()   #队列,即Open表
visited = [False] * (MAXN+10)
q.append(step(N,0))
visited[N] = True
```

```
while len(q) > 0:
    s = q.popleft()
    if s.x == K:   #找到目标
        print(s.steps)
        break
    else:
        if s.x - 1 >= 0 and not visited[s.x-1]:
            q.append(step(s.x-1,s.steps+1))
            visited[s.x-1] = 1
        if s.x + 1 <= MAXN and not visited[s.x+1]:
            q.append(step(s.x+1,s.steps+1))
            visited[s.x+1] = 1
        if s.x * 2 <= MAXN and not visited[s.x*2]:
            q.append(step(s.x*2,s.steps+1))
            visited[s.x*2]  = 1
```

广度优先搜索
例题: 迷宫问题

美国黄石公园

迷宫问题 (百练4127)

定义一个矩阵：

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的最短路线。

迷宫问题

基础广搜。先将起始位置入队列

每次从队列拿出一个元素，扩展其相邻的4个元素入队列(要用二维标志列表判重），直到队头元素为终点为止。队列里的元素记录了指向父节点（上一步）的指针

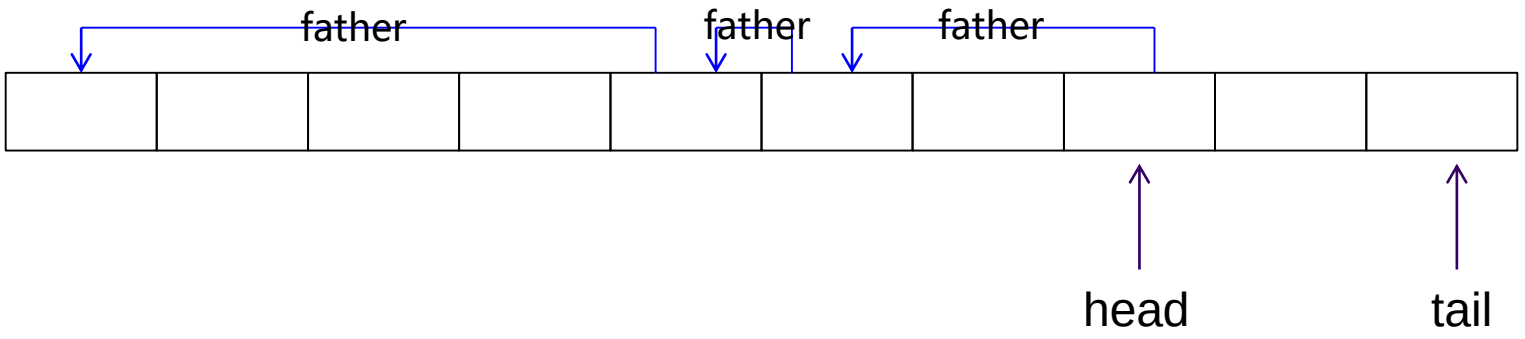
队列元素：(r,c,father)

r,c：节点的坐标
father： 父节点在队列中的下标(从a走到b,则a是b的父节点)

判重的二维列表：flags[i][j]表示 (i,j)那个位置是否走过，即是否入过队列

迷宫问题

- 队列不能用collections.deque，要自己写。用一个足够大的列表实现，维护一个队头指针和队尾指针
- 足够大：能放下所有节点



当队头元素为目标时，沿father指针链取出行走过程中的每个节点的倒序

广度优先搜索
例题: 鸣人和佐助

美国黄石公园

鸣人和佐助(百练6044)

已知一张地图（以二维矩阵的形式表示）以及佐助和鸣人的位置。地图上的每个位置都可以走到，只不过有些位置上有大蛇丸的手下(#)，需要先打败大蛇丸的手下才能到这些位置。

鸣人有一定数量的查克拉，每一个单位的查克拉可以打败一个大蛇丸的手下。假设鸣人可以往上下左右四个方向移动，每移动一个距离需要花费1个单位时间，打败大蛇丸的手下不需要时间。如果鸣人查克拉消耗完了，则只可以走到没有大蛇丸手下的位置，不可以再移动到有大蛇丸手下的位置。

佐助在此期间不移动，大蛇丸的手下也不移动。请问，鸣人要追上佐助最少需要花费多少时间?

```
4 4 1
#@##
**##
###+
****
```

鸣人和佐助

状态定义为：

（r,c,k），鸣人所在的行，列和查克拉数量

如果队头节点扩展出来的节点是有大蛇手下的节点，则其 k 值比队头的k要减掉 1。如果队头节点的查克拉数量为0,则不能扩展出有大蛇手下的节点。

```
4 4 1
#@##
**##
###+
****
```

求钥匙的鸣人

不再有大蛇丸的手下。

但是佐助被关在一个格子里，需要集齐k种钥匙才能打开格子里的门门救出他。

K种钥匙散落在迷宫里。有的格子里放有一把钥匙。一个格子最多放一把钥匙。走到放钥匙的格子，即得到钥匙。

鸣人最少要走多少步才能救出佐助。

求钥匙的鸣人

状态：

（r,c,keys）： 鸣人的行，列，已经拥有的钥匙种数

目标状态 (x,y,K) (x,y)是佐助呆的地方

如果队头节点扩展出来的节点上面有不曾拥有的某种钥匙，则该节点的 keys比队头节点的 keys要加1