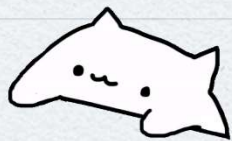


# LUYỆN TẬP THIẾT KẾ THUẬT TOÁN

# NHÓM II XIN CHÀO CÁC BẠN!



- Thành viên:
  - Phan Doãn Thái Bình
  - Nguyễn Trần Tiến



# MỤC LỤC

## X Quiz - Ôn tập



## X Luyện tập



## X Tổng kết



1.  
QUIZ – ÔN TẬP








## NỘI DUNG BÀI QUIZ



**Ôn lại kiến thức đã học qua bài quiz:**

- x Computational Thinking
- x Độ phức tạp
- x Kiểm thử
- x Các phương pháp thiết kế thuật toán

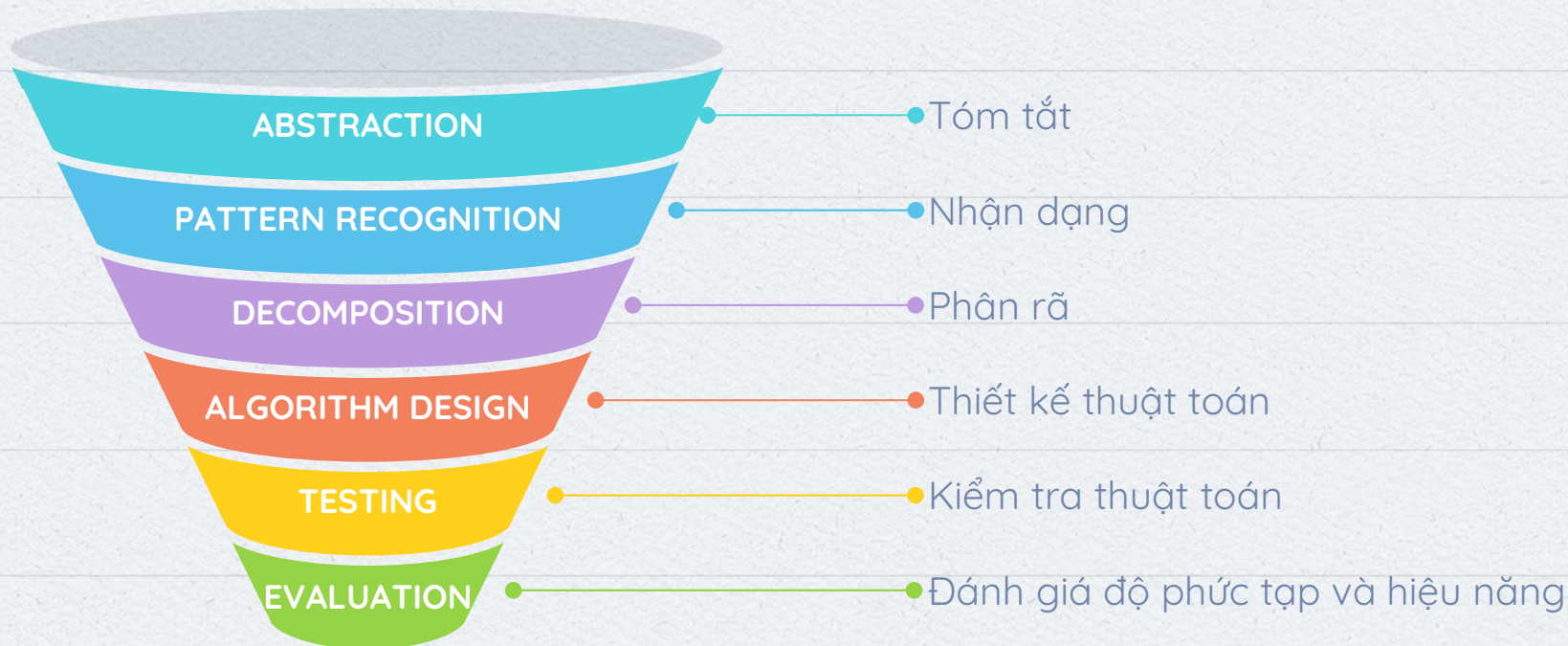


"EVERYONE SHOULD KNOW HOW TO  
PROGRAM A COMPUTER, BECAUSE IT  
TEACHES YOU HOW TO THINK." 





# COMPUTATIONAL THINKING





## MỘT VÀI THUẬT TOÁN ĐÃ HỌC:

- X Brute force
- X Divide & conquer
- X Greedy
- X Backtracking
- X Branch & bound
- X Dynamic programming
- X Geometric Algorithms
- X Graph Algorithms





## 2. LUYỆN TẬP

# BÀI TẬP 1: GIẢI CỨU CÔNG CHÚA

Đề bài:



Ôi không! Công chúa đã bị con Rồng Tà Ác Béo ! Màu Đỏ cướp đi mất rồi!

Bạn – một chàng Hiệp sĩ trong tay có kiếm, quyết tâm lên đường cứu về nàng công chúa của đời mình! Để tới được nơi ở của Rồng Tà Ác Béo ! Màu Đỏ, bạn cần phải trải qua một ngục tối đầy rẫy những cạm bẫy. Trong ngục tối có  $n$  phòng kín, bạn biết được trong phòng thứ  $i$  có  $k[i]$  con quái vật, mỗi con quái vật lần lượt có chỉ số sức mạnh là  $\text{power}[i][j]$  với  $j \in [0; k[i]-1]$ . Bạn phải tiêu diệt tất cả chúng!

Biết tổng số lượng quái vật không quá 10 vạn con!

Bạn – một chàng hiệp sĩ trong tay có kiếm, chiến đấu cùng sự cô độc, chiến thắng kẻ thù dựa trên sức mạnh cây bảo kiếm mình nắm giữ. Bạn sẽ chiến thắng kẻ thù khi sức mạnh của cây kiếm này **lớn hơn** sức mạnh kẻ địch. Cây thần kiếm này có thể **hấp thu 1 sức mạnh** mỗi lần nó tiêu diệt 1 kẻ địch!

BẠN CẦN BAO NHIÊU SỨC  
MẠNH CỦA CÂY KIẾM NÀY?





# BÀI TẬP 1: GIẢI CỨU CÔNG CHÚA

Format:

n

k[1] power[1][0] power[1][1] ... power[1][k[1] - 1]

k[2] power[2][0] power[2][1] ... power[2][k[2] - 1]

...

k[n] power[n][0]...

Input:

2

3 10 15 8

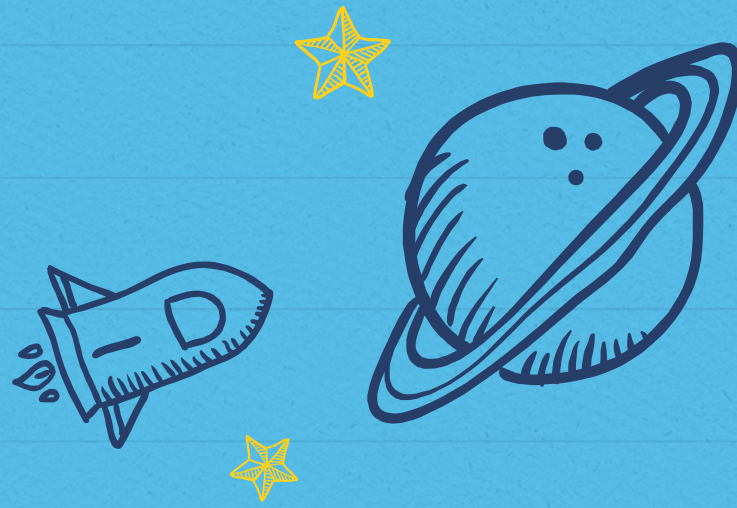
2 12 11

Output:

13



BẠN CẦN BAO NHIÊU SỨC  
MẠNH CỦA CÂY KIẾM NÀY?



# PROCESSING

"MANIFEST PLAINNESS, EMBRACE SIMPLICITY  
REDUCE SELFISHNESS, HAVE FEW DESIRES."





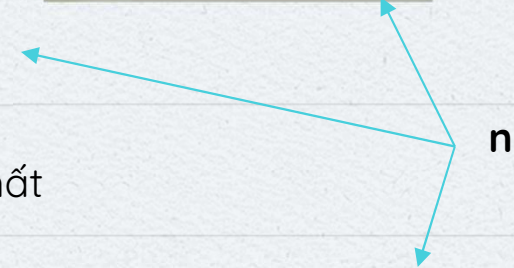
## LỜI GIẢI

### ABSTRACTION

- Cho  $n$  tập hợp có số lượng phần tử khác nhau. Tìm giá trị power nhỏ nhất sao cho có thể hoán vị các phòng và các thứ tự quái vật để:
- Trong từng tập hợp:  

$$X > \text{power}[0], X + 1 > \text{power}[1], X + 2 > \text{power}[2], \dots$$
- Ngoài tập hợp:  

$$X \text{ thoả tập } [0], X + k[0] \text{ thoả tập } [1], X + k[0] + k[1] \text{ thoả tập } [2], \dots$$







## LỜI GIẢI

### DECOMPOSITION

- Nhận thấy ta có thể giải quyết bài toán trong từng phòng: xem mỗi phòng cần bao nhiêu sức mạnh để clear
- => rồi giải quyết xem clear phòng nào trước bằng cách tương tự



n

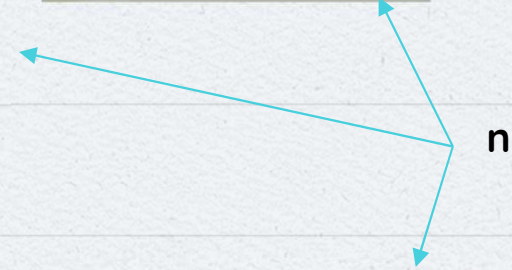




## LỜI GIẢI

### ALGORITHM DESIGN

- Trong một phòng thì ta hoàn toàn có thể chọn con yếu hơn để đánh trước
- $\Rightarrow$  sort lại tăng dần mảng sức mạnh các quái vật
- Tìm min X sao cho X thoả:  
 $X+0 > a[0], X+1 > a[1], X+2 > a[2], X+3 > a[3], \dots$   
 $\Leftrightarrow X > a[0], X > a[1]-1, X > a[2]-2, X > a[3]-3, \dots$   
 $\Leftrightarrow X > \max(a[i] - i)$







LỜI GIẢI

=> GREEDY

## ALGORITHM DESIGN

- Trong một phòng ta **đánh từ con yếu tới con mạnh**
- **Tăng** sức mạnh của mình lên 1 = **Giảm** sức mạnh của tất cả các quái vật đi 1
- Sau khi xác định được chỉ số sức mạnh cần thiết cho mỗi phòng, coi mỗi phòng như một quái vật

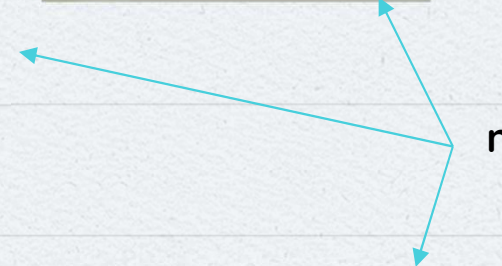
=> Sort lại các quái vật,  $power[i] - i$

=> Vậy sức mạnh cần thiết:

$$need[i] = \max(power[i] - i)$$

=> Vậy mỗi phòng sẽ có  $power = need[i]$

=> Sau khi tiêu diệt: nhận được  $k[i]$  power





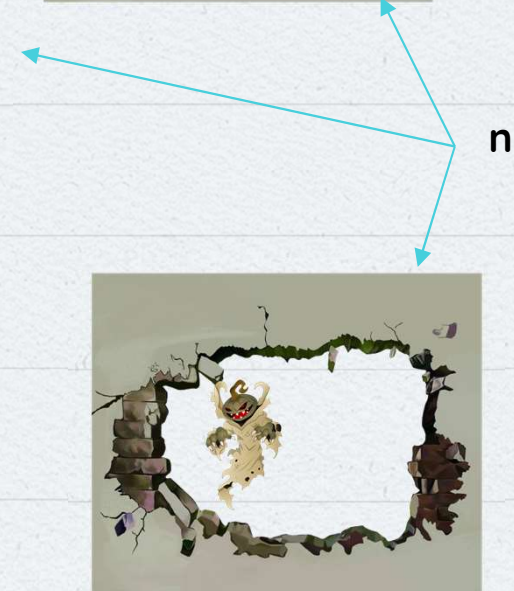


## EVALUATION

- Sort các quái vật trong phòng, sort các phòng  $O(n \log n)$
  - Duyệt tìm max các quái vật trong phòng, các phòng  $O(n)$
- => Vậy ĐPT là  $O(n \log n)$



## LỜI GIẢI





CODE MẪU

main.py ×

```
1 # 1561C - Deep Down Below
2 ts = int(input())
3
4 for _ in range(ts):
5     n = int(input())
6     max_monster = [[0,0] for _ in range(n)]
7     for i in range(n):
8         a = [int(x) for x in input().split()]
9         max_monster[i][0] = 0;
10        max_monster[i][1] = a[0]
11        for j in range(a[0]):
12            max_monster[i][0] = max(max_monster[i][0], a[j+1]-j)
13
14    max_monster.sort(key = lambda x : x[0])
15    hero = 0
16    monster_cleared = 0
17    for i in range(n):
18        max_monster[i][0] -= monster_cleared
19        monster_cleared += max_monster[i][1]
20        hero = max(hero, max_monster[i][0])
21    print(hero + 1)
22
```



## BÀI TẬP 2: HIỆP SĨ DƯỠNG THƯƠNG

Đề bài:

Ôi không, sau trận chiến kinh khủng với bọn quái vật trong hang động tối tăm thì Hiệp sĩ trong tay có kiếm đã kiệt quệ về tinh thần và thể xác!

Thật may mắn rằng trong tay anh ta có thanh thần kiếm có khả năng kích hoạt một lớp lá chắn tường gió dựa trên địa thế.

Vùng bao phủ của lớp lá chắn này có **dạng hình tròn tâm là thanh kiếm**, bán kính hoạt động tùy ý tuy nhiên **vùng này phải chứa một số địa điểm cố định cho trước** (thanh kiếm sẽ tận dụng địa thế của những địa điểm này để lấy năng lượng duy trì lớp bảo hộ)

Ngoài ra vì thời gian dưỡng thương rất lâu nên hiệp sĩ muốn tường gió **tiếp xúc** với con sông tại **đúng 1 điểm** để tùy ý uống nước 😊 Chỉ một điểm thôi vì bọn quái vật hay đi lại gần con sông nên anh không muốn chạm trán với chúng quá nhiều







- 





## BÀI TẬP 2: HIỆP SĨ DƯỠNG THƯƠNG

Format test:

n

x[1] y[1]

...

x[n] y[n]

Input:

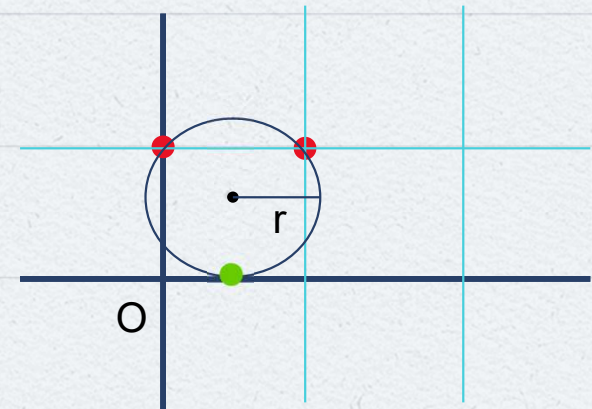
2

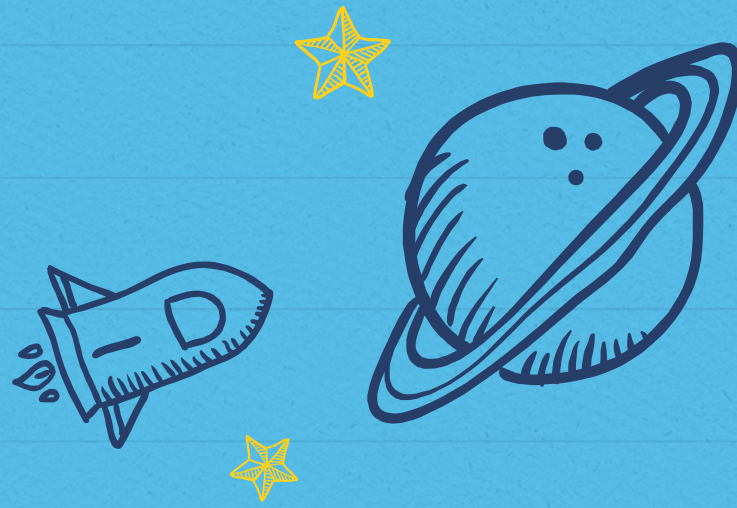
0 1

1 1

Output:

0.625





# PROCESSING

"WITHOUT GEOMETRY, LIFE IS POINTLESS"





## LỜI GIẢI



### ABSTRACTION

- Cho  $N$  điểm trên hệ tọa độ Oxy
- Tìm  $R$  nhỏ nhất sao cho tồn tại đường tròn  $C(I; R)$  chứa  $N$  điểm đã cho và tiếp xúc với trục hoành tại 1 điểm duy nhất





## LỜI GIẢI



### DECOMPOSITION

- Giả sử  $R1$  thoả mãn các yêu cầu thì  $R2 > R1$  cũng thoả mãn
- Với mỗi  $R$ , kiểm tra  $R$  có hợp lệ hay không?







## LỜI GIẢI



### PATTERN RECOGNITION

- Chặt nhị phân bán kính
- Kiểm tra N đoạn thẳng có tồn tại điểm chung không?  
(1D)





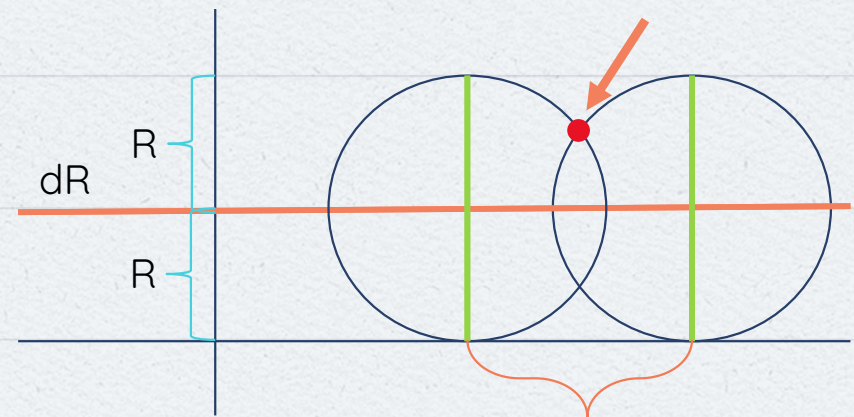
## LỜI GIẢI



### ALGORITHM DESIGN

- Kiểm tra bán kính  $R$  có hợp lệ hay không?
- Với mỗi điểm năng lượng : tìm đoạn  $[a; b]$
- $N$  đoạn  $[a; b]$  không có điểm chung :
  - $\Leftrightarrow \exists b_i, a_j : b_i < a_j$
  - $\Rightarrow \min(b[]) < \max(a[])$
  - $\Rightarrow$  Vậy: Nếu  $\min(b[]) \geq \max(a[]) \Rightarrow R$  hợp lệ

**Lưu ý:** Nếu có 2 điểm nằm khác phía so với trục hoành  $\Rightarrow$  không có đáp án







## LỜI GIẢI



### EVALUATION

- Chặt nhị phân bán kính:  $O(\log C)$  với  $C$  là bán kính tối đa  $\sim 10^{15}$
  - Kiểm tra  $R$  có hợp lệ không tốn  $O(N)$
- $\Rightarrow$  Vậy đpt là  $O(N \log C)$



```

1 # 1059D Nature Reserve
2 import math
3
4 def check(r):
5     global n
6     global x
7     global y
8     mi = 1000000000000000.0
9     mx = -1000000000000000.0
10    for i in range(n):
11        if abs(y[i])>2*r:
12            return False
13        dis = math.sqrt(abs(2*r*abs(y[i]) - y[i]*y[i]))
14        if mi > x[i] + dis: # min Right
15            mi = x[i] + dis
16        if mx < x[i] - dis: # max Left
17            mx = x[i] - dis
18    return mi >= mx

```

```

20 # doc
21 n = int(input())
22 x = []
23 y = []
24 neg = False
25 pos = False
26
27 for _ in range(n):
28     xy = input().split()
29     x.append(int(xy[0]))
30     y.append(int(xy[1]))
31     if int(xy[1])<0:
32         neg = True
33     else:
34         pos = True
35
36 # 2 phía của trục hoành
37 if neg and pos:
38     print(-1)
39     exit()
40
41 # chat nhi phan ket qua
42 l = 0.0
43 r = 1000000000000000.0
44 for _ in range(300):
45     mid = (l+r)/2
46     if check(mid):
47         r = mid
48     else:
49         l = mid
50 print(l)
51

```



## LỜI GIẢI





## BÀI TẬP 3: SNEAKY SNEAKY

Đề bài:

Sau bao khó khăn trắc trở thì cuối cùng Hiệp sĩ trong tay có kiếm cũng đã tới được lâu đài của Rồng Tà Ác Béo ! Màu Đỏ

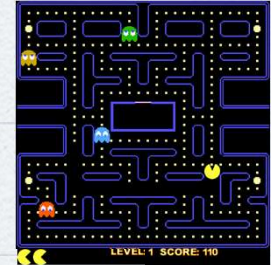
Nhưng trước mặt là  **$n$  căn phòng ( $n \leq 10^5$ )** nối bởi  **$(n-1)$  hành lang**, hai phòng bất kì có thể tới được nhau bằng cách đi qua các hành lang. Thật may mắn rằng trong tay anh ta có tấm bản đồ của lâu đài, và biết được hành lang nào nối phòng nào!

Ngoài ra tấm bản đồ còn cho biết tại căn phòng nào có quái vật ( **$k$  quái vật**)!

Ngay thời điểm hiệp sĩ xâm nhập lâu đài, anh ở căn phòng **1**, nhưng các quái vật đã biết được điều đó và tìm cách bắt anh. Liệu hiệp sĩ có thể chắc chắn vượt qua được các quái vật và tìm **lối ra - phòng mà chỉ có một hành lang nối** (trừ phòng 1)? Biết được hiệp sĩ sẽ bị bắt khi có một quái vật bắt gặp anh tại cùng một phòng, tốc độ hiệp sĩ và quái vật là như nhau.



# BÀI TẬP 3: SNEAKY SNEAKY



Format:

Input:

N K

8 2

r1 r2 r3 ... rk

5 3

u1 v1

4 7

u2 v2

2 5

u3 v3

1 6

...

3 6

7 2

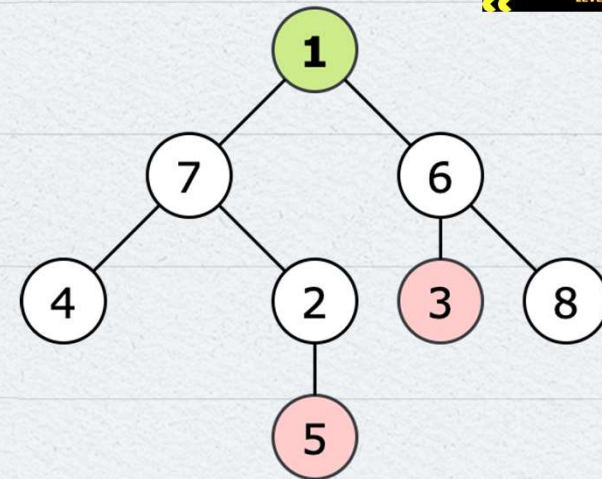
un-1 vn-1

1 7

6 8

Output:

YES



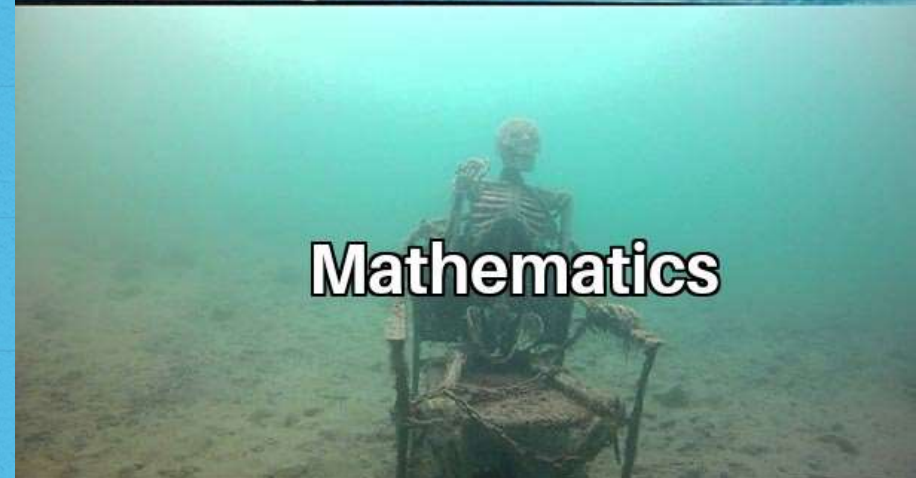




# PROCESSING

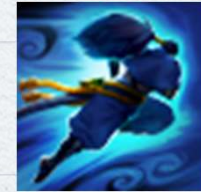
"WITHOUT MATH, LIFE IS EZ"

31





## LỜI GIẢI



### ABSTRACTION

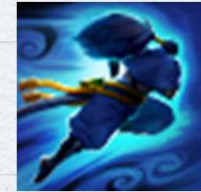
- Cho đồ thị  $N$  đỉnh,  $n-1$  cạnh, giữa 2 đỉnh bất kì luôn có đường đi
- Kiểm tra nếu đỉnh 1 đi tới các nút lá thì có thể tránh không gặp các quái vật xuất phát từ các đỉnh đã cho trước không?







## LỜI GIẢI



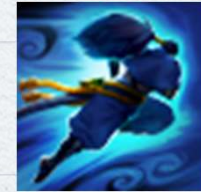
### PATTERN RECOGNITION

- Đồ thị đã cho là 1 cây
  - Tốc độ Hiệp sĩ và quái vật như nhau
- ⇒ giả sử 1 đơn vị thời gian hiệp sĩ và quái vật đều đi được qua một hành lang
- ⇒ **BFS - Loang**



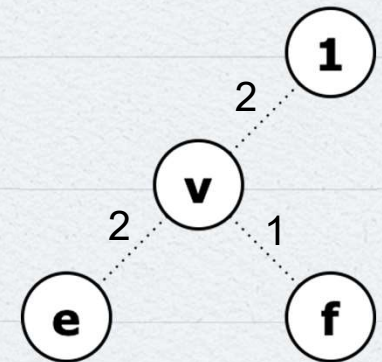


## LỜI GIẢI



### ALGORITHM DESIGN

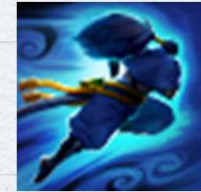
- Ta dễ thấy đối với một nút lá e: thì hiệp sĩ không thể đến khi
- **$\text{Dist}(e, 1) \geq \text{Dist}(e, u)$**  với f là một nút bất kì có quái vật (quái vật đến trước và đợi hiệp sĩ), Dist là khoảng cách giữa 2 đỉnh
- ⇒ **Cần tìm khoảng cách giữa các nút lá tới các phòng có quái vật**
- ⇒ **So sánh với khoảng cách giữa các nút lá tới phòng 1**







## LỜI GIẢI



### ALGORITHM DESIGN

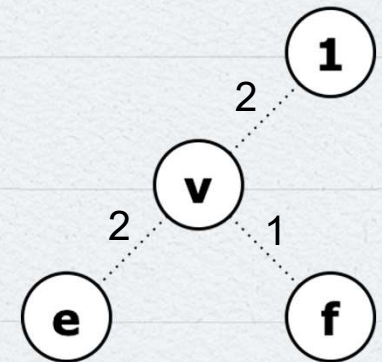
⇒ **Cần tìm khoảng cách giữa các nút lá tới các phòng có quái vật:**

BFS từ các đỉnh có quái vật tới các đỉnh còn lại ->  $d1$

⇒ **So sánh với khoảng cách giữa các nút lá tới phòng 1:**

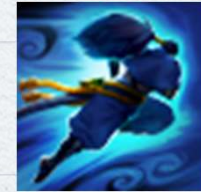
BFS từ đỉnh 1 tới các đỉnh còn lại ->  $d2$

⇒ Với mỗi nút lá: đi được nếu  $d1 > d2$





## LỜI GIẢI



### EVALUATION

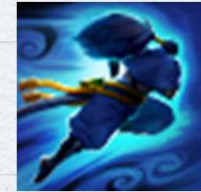
- BFS từ các đỉnh có quái vật tới các đỉnh còn lại
  - ⇒ Đẩy các phòng có quái vật vào queue, bfs tới khi queue đủ n đỉnh
  - ⇒  $O(N)$
- BFS từ đỉnh 1 tới các đỉnh còn lại
  - ⇒ Đẩy phòng 1 vào queue, bfs tới khi queue đủ n đỉnh
  - ⇒  $O(N)$
  - ⇒ Vậy tổng độ phức tạp là  $O(N)$







## CODE MẪU



```
1 def escape_able(n, k, rooms_with_monsters, corridors):
2     # create corridors
3     neighbours = [[] for _ in range(n)]
4     for edge in corridors:
5         neighbours[edge[0]].append(edge[1])
6         neighbours[edge[1]].append(edge[0])
7
8     # monsters
9     queue = []
10    monster_to_this = [n+1]*n
11
12    for monster in rooms_with_monsters:
13        monster_to_this[monster] = 0
14        queue.append(monster)
15
16    while queue:
17        u = queue.pop(0)
18        for v in neighbours[u]:
19            if monster_to_this[v] == n+1:
20                monster_to_this[v] = monster_to_this[u] + 1
21                queue.append(v)
```

```
22
23    # hero
24    queue = []
25    hero_to_this = [n+1]*n
26    hero_to_this[0] = 0
27    queue.append(0)
28
29    while queue:
30        u = queue.pop(0)
31        for v in neighbours[u]:
32            if hero_to_this[v] == n+1:
33                hero_to_this[v] = hero_to_this[u] + 1
34                queue.append(v)
35
36    # check per leaf
37    for leaf in range(1,n):
38        if len(neighbours[leaf]) == 1: # check if this is a leaf
39            if monster_to_this[leaf] > hero_to_this[leaf]:
40                return True
41
42    return False
```

## TỔNG KẾT

## X Vận dụng thuần thục Computational Thinking

- X Luyện tập thường xuyên các phương pháp thiết kế thuật toán



