ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

XẾP LỊCH THI ĐẤU THỂ THAO

Giảng viên: Vương Bá Thịnh

SV thực hiện: Nguyễn Lê Chí Bảo

MSSV: 1610179

MỤC LỤC

1. MỤC TIÊU

- Củng cố các kiến thức của môn học Trí Tuệ Nhân Tạo (AI).

- Rèn luyện thêm về kỹ năng lập trình, đặc biệt là đối với Python

- Rèn luyện cách đọc tài liệu (document)

- Tăng cường khả năng nghiên cứu

2. GIỚI THIỆU BÀI TOÁN

2.1 Đặt vấn đề

Bài toán đặt ra vấn đề là xếp lịch cho cuộc thi đấu môn thể thao X sao cho phù hợp với yêu cầu đặt ra.

2.2 Đặc tả bài toán

- Một giải đấu có n vận động viên (vđv).

- Một trận đấu gồm 2 vđv thi đấu đối kháng.

- Mỗi vđv có 1 điểm số trong bảng xếp hạng của môn thể thao X.

- Mỗi vđv thi đấu chính xác với k vđv khác

- Lịch đấu cần tối ưu hóa mục tiêu sau: **Điểm số trung bình các đối thủ của 2 vđv bất kì (tbinh-i và tbinh-j) không quá chênh lệch**

2.3 Yêu cầu bài toán

- Ngôn ngữ lập trình: Python 3.

- Giải thuật : giải thuật tìm kiếm không có thông tin hoặc heuristic để xếp lịch thi đấu.

- Input: file text với nội dung như sau:

    - Dòng 1: 2 số nguyên n và k, vd: 100 10

    - Dòng 2 -> n+1: Điểm số trong bảng xếp hạng của 1 vđv (dòng 2 là vđv có id là 1, dòng n+1 là vđv có id là n)

- Output: file text với nội dung sau:

- Dòng 1->k: mỗi dòng là đối thủ của vđv có id 1, xác định bằng 1 số nguyên thể hiện id của vđv

- Dòng k+1 -> 2k: đối thủ của vđv có id 2.

- Tương tự cho các dòng tiếp theo.

- Hàm main tuân theo prototype sau: **def main(file_input, file_output)**

3. PHÂN TÍCH BÀI TOÁN

Dễ dàng nhận thấy yêu cầu bài toán là tìm ra cách xếp lịch thi đấu cho n vận động viên, mỗi vận động viên đấu đúng k trận sao cho phù hợp với yêu cầu tối ưu. **Theo lý thuyết đồ thị, bài toán này thực chất là tìm kiếm một đồ thị đều bậc k (regular graph)**

Với những kiến thức đã học, ta có thể giải quyết bài toán này theo hai hướng : Tìm kiếm không có thông tin và tìm kiếm theo kinh nghiệm.

Đối với giải thuật tìm kiếm không có thông tin: Ta sử dụng giải thuật tìm kiếm theo chiều ngang- trước (breadth –first search).

Đối với giải thuật tìm kiếm theo kinh nghiệm: Ta sử dụng giải thuật leo đồi(hill climbing) để giải quyết.

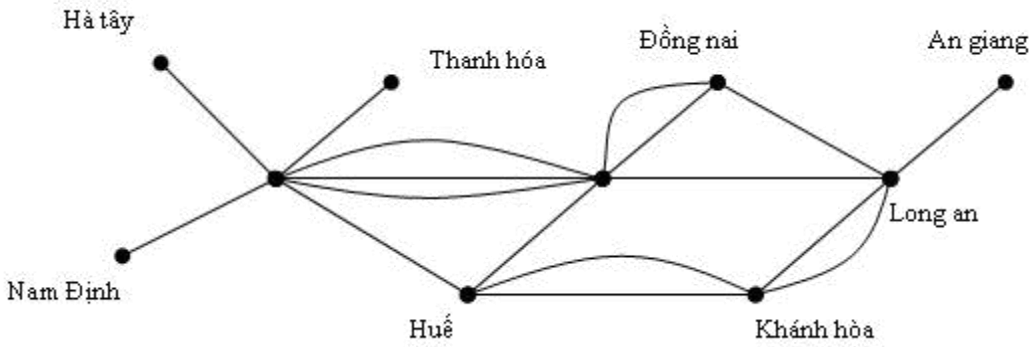
3.1 Nhắc lại lý thuyết về đồ thị và đồ thị đều (regular graph)

3.1.1 Định nghĩa về đồ thị

Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Được mô tả hình thức:

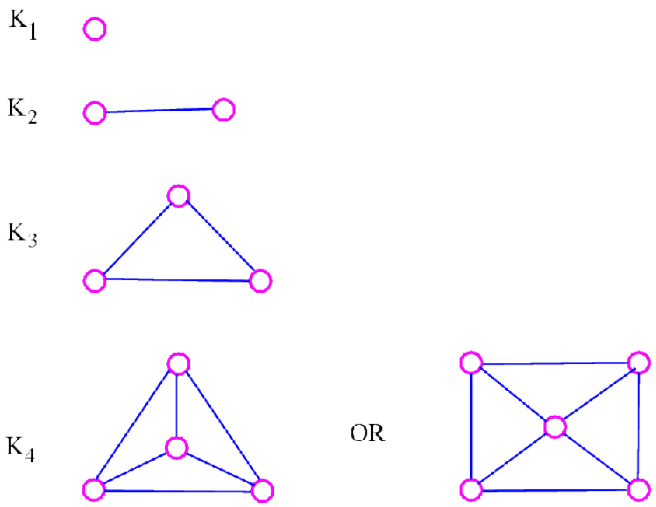
$$G = (V, E)$$

Với V gọi là tập các đỉnh (Vertices) và E gọi là tập các cạnh (Edges). Có thể coi E là tập các cặp (u, v) với u và v là hai đỉnh của V.



3.1.2 Đồ thị đều bậc K

- Mọi đỉnh đều có cùng bậc K
- Số đỉnh: | V | = n
- Bậc: dev(v) = k , $\forall v \in V$
- Số cạnh: E = $\frac{n*k}{2}$



3.2 Nhắc lại về giải thuật DFS (Depth –first search)

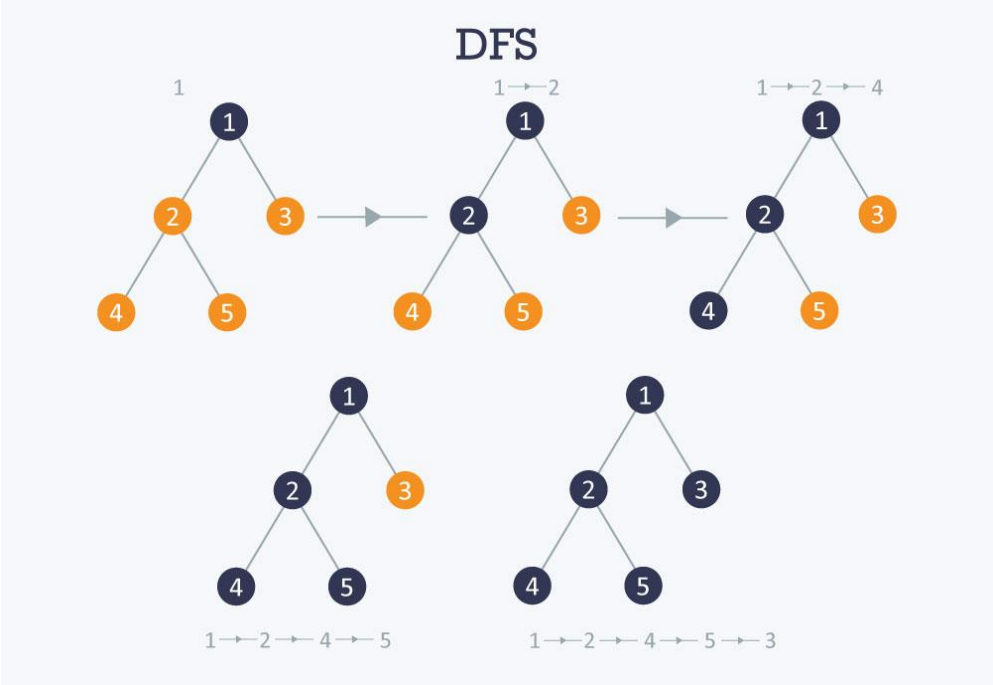
3.2.1 Khái niệm

Thuật toán Depth First Search (DFS – Tìm kiếm theo chiều sâu) là một dạng thuật toán duyệt hoặc tìm kiếm trên cây hoặc đồ thị. Trong lý thuyết khoa học máy tính, thuật toán DFS nằm trong chiến lược tìm kiếm mù (tìm kiếm không có định hướng,

không chú ý đến thông tin, giá trị được duyệt) được ứng dụng để duyệt hoặc tìm kiếm trên đồ thị.

3.2.2 Ý tưởng thuật toán

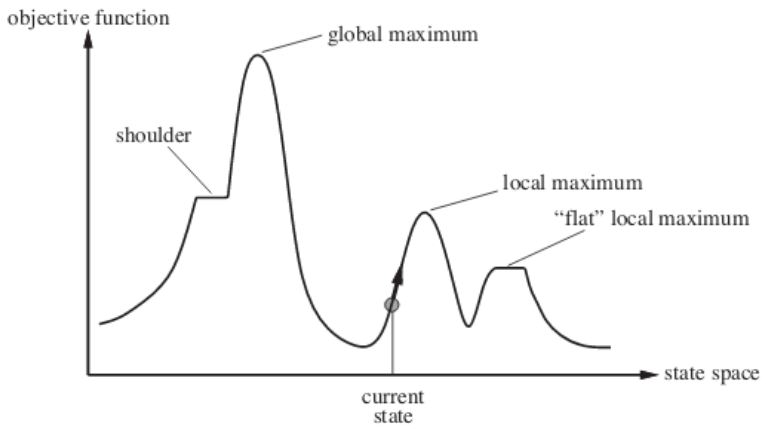
Từ đỉnh (nút) gốc ban đầu, thuật toán duyệt đi xa nhất theo từng nhánh, khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo. Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.



3.3 Nhắc lại về giải thuật leo đồi

3.3.1 Khái niệm

Trong khoa học máy tính, giải thuật Leo dồi (Hill Climbing) là một kỹ thuật tối ưu toán học thuộc họ tìm kiếm cục bộ. Nó thực hiện tìm một trạng thái tốt hơn trạng thái hiện tại để mở rộng. Để biết trạng thái tiếp theo nào là lớn hơn, nó dùng một hàm H để xác định trạng thái nào là tốt nhất.



Giải thuật như sau:

1. Lượng giá trạng thái khởi đầu
2. Lặp cho đến khi đạt đến trạng thái mục tiêu hoặc không còn tác vụ để thử
- Chọn một tác vụ để thử
- Lượng giá trạng thái mới do tác vụ sinh ra
- Nếu đó là trạng thái mục tiêu thì kết thúc
- Nếu trạng thái mới tốt hơn trạng thái hiện tại thì chuyển sang trạng thái mới.

3.3.2 Giải thuật leo đồi dốc nhất (steepest-ascent hill climbing)

Ý tưởng của giải thuật này là thay vì chuyển đến bất kỳ trạng thái nào tốt hơn trạng thái hiện tại, giải thuật chuyển đến trạng thái tốt nhất trong số các trạng thái tốt hơn đó.

Giải thuật như sau:

1. Lượng giá trạng thái khởi đầu
2. Lặp cho đến khi đạt đến trạng thái mục tiêu hoặc không chuyển đến được trạng thái mới
- Với mỗi tác vụ, lượng giá trạng thái do tác vụ sinh ra; nếu đó là trạng thái mục tiêu thì kết thúc
- Chọn trạng thái tốt nhất trong số các trạng thái do các tác vụ sinh ra. Nếu nó tốt hơn trạng thái hiện tại thì chuyển sang trạng thái đó.

3.4Xác định hướng giải quyết bài toán

3.4.1 Giải quyết bài toán bằng thuật toán tìm kiếm không có thông tin : DFS

Với những kiến thức đã học, ta sẽ giải quyết bài toán này như sau:

Bước 1: Xét đỉnh 1, sinh ra các tổ hợp chập K của (N-1) đỉnh.

Bước 2: Cứ mỗi tổ hợp, ta lưu trạng thái tổ hợp đó. Sau đó lần lượt xét từng đỉnh trong tổ hợp, cứ mỗi đỉnh như vậy lại sinh ra các tổ hợp con khác. Một nhánh cây dừng khi đỉnh đang xét không thể sinh ra các tổ hợp con hoặc đã tìm thấy một phương án.

Bước 3. Khi tìm thấy phương án, đánh giá phương án đó là tốt hơn phương án cũ không. Nếu tốt hơn thì lưu phương án mới.

Giải thuật kết thúc khi đã tìm thấy hết các phương án và đưa ra phương án tối ưu nhất.

3.4.2 Giải quyết bài toán bằng thuật toán tìm kiếm theo kinh nghiệm: Giải thuật leo đồi

Bước 1: Sinh ra một phương án đầu tiên và lượng giá phương án đầu tiên đó

Bước 2: Sử dụng giải thuật leo đồi tốt nhất (steepest-ascent hill climbing) để tìm các phương án tốt nhất có thể có.

4. GIẢI QUYẾT BÀI TOÁN

4.1Hàm lượng giá

4.1.1 Mục tiêu tối ưu

Điểm số trung bình các đối thủ của 2 vđv bất kì (tbinh-i và tbinh-j) không quá chênh lệch.

4.1.2 Phân tích điều kiện tối ưu

Với **tbinh** là mảng chứa điểm trung bình các đối thủ của mỗi vận động viên ở một trạng thái xếp lịch nhất định, ta gọi **maxp** là vận động viên có điểm số trung bình các đối thủ cao nhất, **minp** là vận động viên có điểm số trung bình các đối thủ thấp nhất. Vậy **tbinh-maxp** là điểm số trung bình đối thủ cao nhất, **tbinh-minp** là điểm số trung bình đối thủ thấp nhất ở phương án đang xét.

Ta thấy rằng, khi giảm chênh lệch của tbinh-maxp và tbinh-minp thì điểm trung bình các đối thủ chênh lệch giữa hai vận động viên bất kỳ sẽ có xu hướng giảm đi.

Thật như vậy, ta nhận thấy:

$$\text{tbinh(maxp)} - \text{tbinh(minp)} \geq \text{tbinh(i)} - \text{tbinh(j)}$$

Với **tbinh-i** và **tbinh-j** là điểm trung bình đối thủ của các vận động viên bất kỳ trong trạng thái xếp lịch đang xét.

Mặt khác ta cũng có:

$$\text{tbinh(maxp)} - \text{tbinh(i)} \geq \text{tbinh(i)} - \text{tbinh(j)}$$

$$\text{tbinh(i)} - \text{tbinh(minp)} \geq \text{tbinh(i)} - \text{tbinh(j)}$$

Vậy để giảm sự chênh lệch điểm trung bình đối thủ của các vận động viên, thì việc giảm sự chênh lệch điểm trung bình cao nhất và điểm trung bình thấp nhất ở từng trạng thái đang xét là thích hợp và khả thi.

4.1.3 Hàm lượng giá

Từ những lập luận trên, ta đưa ra hàm lượng giá cho bài toán này như sau:

H = tbinh(maxp) – tbinh(minp)

4.2Xác định không gian mẫu

Ta thấy không phải bất kỳ một số k, n với một tập hợp cách sắp xếp nào đều hợp lý để tạo thành một phương án đúng. Vậy nên việc xác định không gian mẫu là cần thiết để biết chúng ta phải làm sao cho có được kết quả chính xác, tránh trường hợp không tìm thấy phương án. Bên cạnh những lý thuyết đã nêu trên , ta còn phải dựa vào thực tế để đưa ra những ràng buộc đúng đắn như sau:

- Số vận động viên phải nhiều hơn số trận đấu họ phải đấu
- Khi số vận động viên là số chẵn, thì số trận đấu có thể là chẵn hoặc lẻ
- Khi số vận động viên là số lẻ, thì số trận đấu phải là số chẵn.
- Số vận động viên không thể bé hơn 1.
- Số trận đấu của mỗi vận động viên không thể bé hơn 0.
- Số trận đấu mà các vận động viên đấu phải như nhau.

Ta biểu diễn những điều kiện đó thành dạng toán học như sau:

$$N > K$$

$$N * K \ \vdots \ 2$$

$$N \geq 1$$

$$K \geq 0$$

4.3 Giải quyết bài toán

4.3.1 Thuật toán tìm kiếm không có thông tin : DFS

Đối với giải thuật nãy, ta sử dụng một số cấu trúc dữ liệu sau:

MatrixState[0..N-1][0..N-1] : Là mảng 2 chiều lưu trạng thái các trận đấu đang xét. Với MatrixState[i][j] = 1 nghĩa là i đấu với j, MatrixState[i][j] = 0 nghĩa là i không đấu với j.

Score[0..N]: Mảng chứa điểm số của vận động viên.

Count[0..N]: Mảng lưu số lượng các trận đấu của mỗi vận động viên trong trạng thái đang xét. Nếu Count[i] = K, $\forall i \in N$ => Đây là một phương án.

Mã giả của bài toán khi sử dụng giải thuật DFS như sau:

```
Def DFS(vertex):
    Tìm tập hợp các đỉnh vẫn có thể đấu và chưa đấu với vertex (Gọi là tập set)
    Tính số trận mà vertex còn có thể đấu
    Nếu tập hợp các đỉnh vẫn có thể đấu là rỗng thì ta kiểm tra:
        Nếu đó là một phương án thì so sánh với phương án cũ
        Còn không phải là phương án thì thoát
  Từ số trận mà vertex có thể đấu và với tập set, ta sinh ra tổ hợp các trường hợp có thể xảy ra. List các tổ hợp gọi là comb.
  For i in comb: # Duyệt lần lượt từng tổ hợp
      Count[vertex] = K
      Cập nhật Count và MatrixState của các đỉnh có trong tập set
      For j in i:  # Duyệt lần lượt từng đỉnh trong tổ hợp
          DFS(j)
      Khôi phục lại State ban đầu
```

4.3.2 Thuật toán tìm kiếm theo kinh nghiệm: Thuật toán leo đồi dốc nhất(steepest- ascent hill climbing)

Trạng thái khởi đầu (Init State): (binh-maxp, tbinhminp, MatrixState) tại trạng thái tìm được ở bước 1.

MatrixState được biểu diễn bằng một ma trận hai chiều. Với MatrixState[i][j] = 1 tức có trận đấu giữa i và j. MatrixState[i][j] = 0 thì i và j không đấu nhau.

Ví dụ:

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

Để sinh ngẫu nhiên trạng thái khởi đầu, trong python, ta sử dụng thư viện **networkx.**

Trạng thái (state): (tbinh-maxp,tbinh-minp, MatrixState).

Bước chuyển trạng thái (move): Để giảm tối đa độ chênh lệch thì ta sẽ lần lượt đổi vị trí lần lượt từng đối thủ của vận động viên có điểm trung bình đối thủ cao nhất với từng đối thủ có điểm thấp nhất của vận động viên có điểm thấp nhất, nhằm giảm tbinh-maxp và tăng tbinh-minp nhiều nhất. Ta chọn trạng thái tốt nhất trong số các trạng thái được sinh ra. Nếu trạng thái đó tốt hơn (là trạng thái có tbinh-maxp - tbinh-minp mới bé hơn trạng thái cũ), ta sẽ lưu trạng thái đó rồi từ trạng thái mới này tìm bước nhảy khác tốt hơn khi còn có thể. Nếu không tìm được trạng thái tốt hơn thì ta nhận thấy không còn trạng thái tốt hơn có thể tìm ra với thuật toán này, nên dừng lại và báo kết quả.

Ràng buộc để giảm chi phí tính toán và tránh trùng lặp xảy ra khi hoán đổi đối thủ cho nhau: Nếu đối thủ của maxp là minp và ngược lại thì ta không đổi vị trí chúng cho nhau, không đổi vị trí đối thủ chung của maxp và minp.

Mã giả của thuật toán này như sau:

```
Tạo phương án đúng bất kỳ
Cập nhất MatrixState theo phương án đó.
Tính tbmax, tbmin, max, min của phương án đó.
While (True):
    Liệt kê số đối thủ của max mà min chưa đấu. (set1)
    Liệt kê số đối thủ của min mà max chưa đấu. (set2)
    # lần lượt swap 1 phần tử của set1 với 1 phần tử của set2
    For i in set1
        For j in set 2
            Swap trận đấu giữa max với i thành max với j
            Swap trận đấu giữa min với j thành min với i
            Tính toán tbmax, tbmin, max, min ở phương án này.
            So sánh với kết quả của phương án con khác
    Nếu phương án con tốt nhất là phương án tốt hơn so với phương án ban đầu
( phương án cha) thì nhảy sang phương án con. Tiếp tục thực hiện vòng lặp.
    Nếu không có phương án nào tốt hơn phương án cha thì dừng vòng lặp
```

5. KẾT QUẢ VÀ ĐÁNH GIÁ

5.1 Sinh input

Đoạn code sinh input:

```
import random
def main(output_file):
    file = open(output_file,'w')
    N = -1
    K = -1
    while (True):
        K = random.randint(2,100)
        N = random.randint(K+1,101)
        if K*N % 2 != 0 :
            continue
        else:
            break
    String = str(N) +" " + str(K)+'\n'
    for i in range(0,N):
        String += str(random.randint(1,10000)) + '\n'
    file.write(String)
```

```
    file.close()
```

Với đoạn sinh input trên ta giới hạn các thông số như sau:

K < N < 101

1 <= Score[i] <= 10000

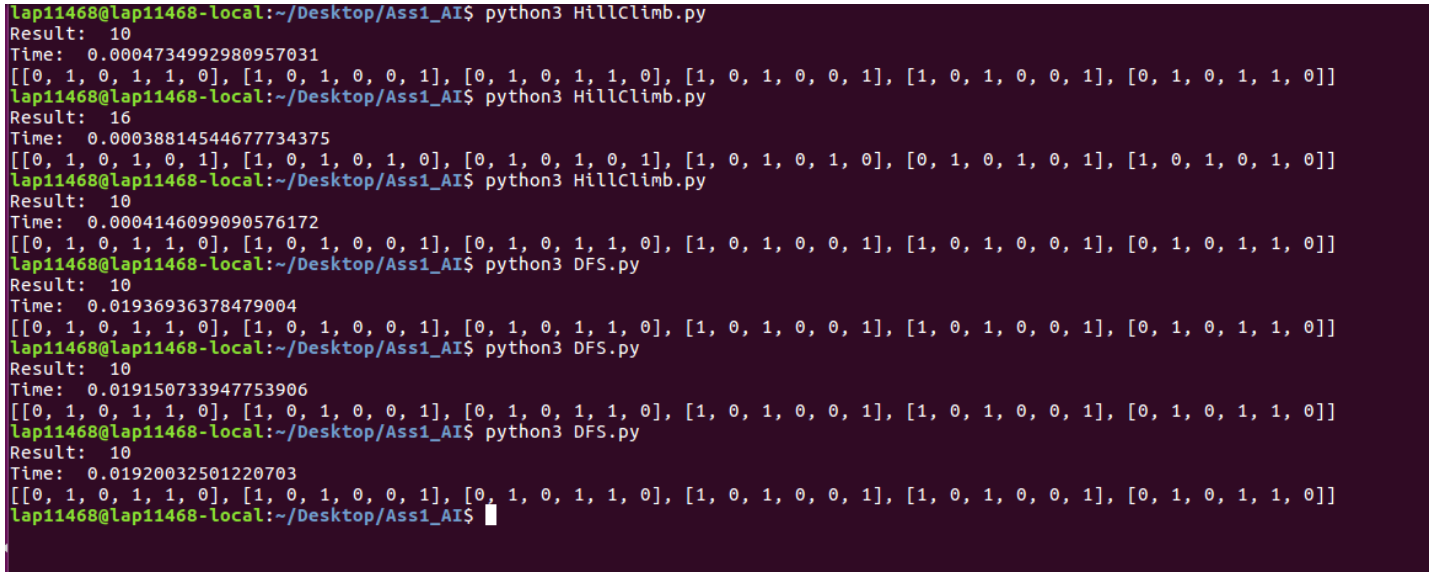
K * N là một số chẵn

5.2 Kết quả

Cho cùng 1 file input như sau: (N = 6, K = 3)

```
6 3
30
21
34
56
20
23
```

Mỗi giải thuật, ta chạy 3 lần, kết quả thu được như sau:



Từ kết quả trên ta thấy được:

- Thời gian chạy của giải thuật Hill Climbing rất nhanh (gấp 50 lần) so với giải thuật DFS trong testcase này.
- Kết quả thu được của hai giải thuật không quá chênh lệch. Có trường hợp Hill Climbing đưa ra được kết quả tối ưu như DFS

Ta xét thêm 1 ví dụ nữa: (N = 10, K = 4)

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

Mỗi giải thuật, ta chạy 3 lần, kết quả thu được như sau:

```
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 HillClimb.py
Result:  0
Time:  0.0006384849548339844
[[0, 0, 1, 0, 1, 1, 0, 1], [0, 0, 1, 0, 1, 1, 0, 1], [1, 1, 0, 1, 0, 0, 1, 0], [0, 0, 1, 0, 1, 1, 0, 1], [1, 1, 0, 1, 0, 0, 1, 0], [1, 1, 0, 1,
 0, 0, 1, 0], [0, 0, 1, 0, 1, 1, 0, 1], [1, 1, 0, 1, 0, 0, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 HillClimb.py
Result:  6
Time:  0.0005526542663574219
[[0, 1, 1, 0, 1, 1, 0, 0], [1, 0, 0, 1, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0], [0, 1, 0, 0, 1, 1, 1, 0], [1, 0, 1, 1, 0, 0, 0, 1], [1, 0, 1, 1,
 0, 0, 0, 1], [0, 1, 1, 1, 0, 0, 0, 1], [0, 1, 0, 0, 1, 1, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 HillClimb.py
Result:  2
Time:  0.0006361007690429688
[[0, 0, 1, 0, 1, 0, 1, 1], [0, 0, 0, 1, 1, 1, 0, 1], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0], [1, 1, 0, 1, 0, 1, 0, 0], [0, 1, 1, 0,
 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [1, 1, 1, 0, 0, 0, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 DFS.py
Result:  0
Time:  80.46582365036011
[[0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0,
 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 DFS.py
Result:  0
Time:  80.77608370780945
[[0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0,
 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$ python3 DFS.py
Result:  0
Time:  80.90188956260681
[[0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0,
 1, 0, 1, 0], [1, 0, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 0, 1, 0]]
lap11468@lap11468-local:~/Desktop/Ass1_AI$
```

Từ kết quả trên ta thấy được:

- Thời gian chạy giải thuật Hill Climbing rất nhỏ khi N tăng. Nhưng ngược lại, với DFS, khi N tăng, thời gian chạy tăng lên rất nhiều.
- Kết quả giữa hai giải thuật không chênh lệch nhau quá nhiều.

5.3Đánh giá

Từ hai ví dụ trên ta thấy rằng: Nếu chúng ta quan tâm đến việc thời gian chạy thuật toán chạy nhanh và tạm thời bỏ qua đi việc tìm phương án tối ưu nhất thì giải thuật leo đồi (Hill Climbing) là giải thuật tốt nhất trong việc này.

Từ đây, ta tập trung đánh giá và cải thiện thuật toán leo đồi (Hill Climbing).

Thuật toán leo đồi (Hill Climbing) xử lý nhanh, tuy nhiên có một số nhược điểm sau:

- *Tối ưu cục bộ*: Trạng thái đạt đến khi giải thuật kết thúc có thể chỉ là trạng thái tốt nhất so với các trạng thái gần xung quanh, chứ không phải tốt nhất toàn cục so với tất cả các trạng thái của bài toán
- *Rơi vào đồng bằng*: Giải thuật có thể dẫn đến trạng thái có cùng giá trị như tất cả như tất cả các trạng thái tiếp theo, và sẽ kết thúc tại đó mà chưa đến được trạng thái mục tiêu
- *Rơi vào rìa đồi*: Đây là trường hợp mà tất cả các trạng thái có thể tiếp theo đều xấu hơn trạng thái hiện tại, và do đó giải thuật phải kết thúc.

6. CẢI THIỆN THUẬT TOÁN LEO ĐỒI (HILL CLIMBING)

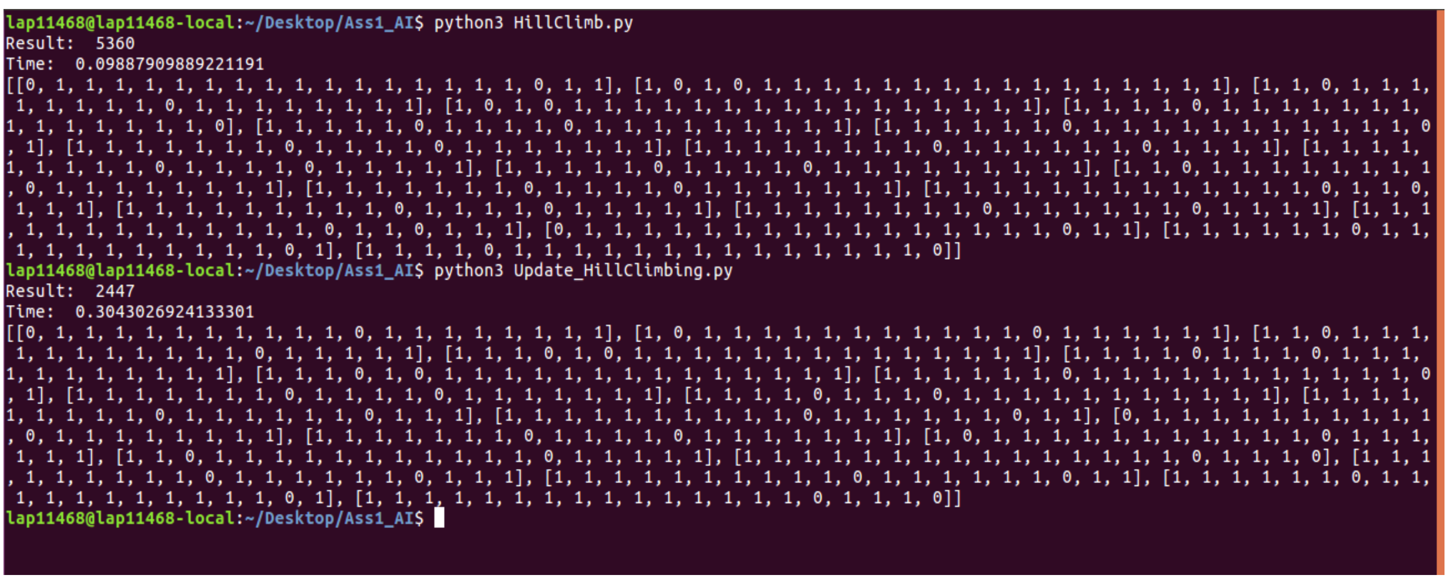
Để có thể khắc phục và hạn chế nhược điểm của giải thuật trên.Với bài toán này, ta đề ra một phương pháp như sau:

- Ta sẽ sinh ra 10 trạng thái ban đầu và tiến hành leo đồi ở từng trường hợp.
- Sau đó, cứ mỗi trường hợp, ta thu được một kết quả tốt nhất khi leo đồi ở trường hợp đó.
- Sau đó, lấy kết quả tốt nhất trong tất cả các trường hợp.

Phương pháp này làm tăng khả năng tìm ra được kết quả tối ưu nhất có thể có, thậm chí nếu may mắn ta có thể tìm đến kết quả tối ưu nhất của bài toán.

Xét một ví dụ với N = 20, K =18

20 18 4919 3851 5013 5493 855 2125 8431 6406 8095 4398 8590 2441 1018	5107 3589 2441 1018 5107 3589 942 4621 436 1376 7354



Sau khi cải thiện , ta thấy kết quả trả về tốt hơn nhiều so với ban đầu.

Link source code: https://github.com/nguyenlechibao/Ass1_AI