



ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
CƠ SỞ TRÍ TUỆ NHÂN TẠO
PROJECT 01

Sinh viên thực hiện : 21127043 - Lư Trung Hậu

21127083 - Hoàng Đức Kiên

21127285 - Phan Thanh Hoàng

21127553 - Lê Gia Quốc Tỉ

Lớp : 21CLC03

Giáo viên hướng dẫn: Nguyễn Trần Duy Minh
Lê Ngọc Thành
Nguyễn Ngọc Thảo
Nguyễn Hải Đăng

I . Thông tin nhóm:

- + Lư Trung Hậu – 21127043 – 21CLC03
- + Hoàng Đức Kiên – 21127083 – 21CLC03
- + Phan Thanh Hoàng – 21127285 – 21CLC03
- + Lê Gia Quốc Tỉ - 21127553 – 21CLC03

II. Phân công nhiệm vụ:

STT	Họ và Tên	MSSV	Nhiệm vụ	Mức độ hoàn thành
1	Lư Trung Hậu	21127043	Kiểm tra, chạy test case, quay video, hoàn thành báo cáo thuật toán brute force, kiểm tra, sửa lỗi báo cáo Cài đặt thuật toán Genetic Algorithm Phân công nhiệm vụ cho các thành viên nhóm	100%
2	Hoàng Đức Kiên	21127083	Kiểm tra, chạy test case, quay video thuật toán branch and bound, local search, Tạo bộ test case và viết báo cáo về test case và Branch bound Tổng hợp báo cáo	100%
3	Phan Thanh Hoàng	21127285	Cài đặt, kiểm tra, chạy và so sánh kết quả test case, viết báo cáo thuật toán brute force, branch bound và local beam. Hoàn thành file main chạy các thuật toán	100%

			Hoàn thành báo cáo Branch bound và local beam Kiểm tra thuật toán Genetic Algorithm.	
4	Lê Gia Quốc Tỉ	21127553	cài đặt thuật toán genetic algorithms, chạy test case và quay video, viết và hoàn thành báo cáo về thuật toán Genetic algorithm	100%

III. Kết quả tổng quan:

- Các thành viên hoàn thành tốt công việc được giao, hoàn thành các hạn mục được yêu cầu, có sự nhất quán với nhau trong công việc.
- Còn một số hạn chế về máy móc nên công việc chưa được trơn tru.

IV. Cách thức chương trình chạy

B1: Chạy Command Prompt trên folder chứa source code

B2: gõ lệnh `python main.py [số thứ tự thuật toán] [tên file test case]`

với số thứ tự thuật toán như sau:

1: Brute force

2: Branch and bound

3: Local beam search

4: Genetic Algorithm

compare: để chạy cả 4 thuật toán

vd: `python main.py 1 input_x.txt` sẽ chạy thuật toán brute force trên file `input_x.txt`

Sau khi thực hiện hai bước trên, chương trình sẽ chạy và xuất ra màn hình kết quả của thuật toán và thời gian cùng dung lượng sử dụng của thuật toán đó

Lưu ý:

- Cần phải install module numpy và module pandas trước khi chạy source code
- Nếu [tên file test case] không được nhập vào thì chương trình sẽ tự động sử dụng file input_x.txt làm file test.
- Nếu [số thứ tự của thuật toán] không được nhập vào thì chương trình sẽ yêu cầu nhập số thứ tự thuật toán từ bàn phím và chạy file test case input_x.txt

V. Báo cáo công việc

- Các file test được sử dụng:

Test case nhỏ :

10_soluInFirst.txt: solution được tìm thấy ở đầu.

15_largeBound.txt: khối lượng túi lớn để lấy mọi đồ vật

20_soluInLast.txt:solution được tìm thấy ở cuối.

30_noSolu.txt: không tìm thấy solution.

35_soluInMid.txt: solution được tìm thấy ở giữa.

Test case lớn :

50_random.txt: random các giá trị

100_random.txt: random các giá trị

150_random.txt: random các giá trị

200_random.txt: random các giá trị

250_random.txt: random các giá trị

300_random.txt: random các giá trị

1. Algorithm 1: brute force searching

A. Mô tả thuật toán:

- Hàm Brute_Force này là một thuật toán sử dụng BFS (Breadth-First Search) để giải quyết bài toán tìm tập hợp các vật phẩm có giá trị lớn nhất và không vượt quá trọng lượng giới hạn trong một túi có sức chứa cố định. Các tham số truyền vào bao gồm danh sách các vật phẩm với mỗi vật phẩm có trọng lượng và giá trị, sức chứa giới hạn của túi và số lượng các lớp khác nhau của vật phẩm. Thuật toán bắt đầu bằng cách tạo một node với sức chứa và giá trị bằng 0, chỉ số node và số lượng node của từng lớp bằng 0. Node này được đưa vào hàng đợi BFS để xét. Trong quá trình lặp, thuật toán lấy một node từ hàng đợi BFS, kiểm tra xem node này có vượt quá sức chứa và có giá trị lớn hơn node lớn nhất hiện tại không. Nếu có, node này trở thành node lớn nhất hiện tại. Sau đó, thuật toán tạo các node tiếp theo bằng cách thêm từng vật phẩm vào túi một cách lần lượt và đưa các node mới này vào hàng đợi BFS. Thuật toán tiếp tục lặp lại cho đến khi hàng đợi BFS rỗng hoặc tất cả các vật phẩm đã được chọn. Cuối cùng, thuật toán trả về node có giá trị lớn nhất và không vượt quá sức chứa giới hạn của túi, cùng với danh sách các vật phẩm đã chọn.

B. Mã giả của thuật toán:

```
func brute_force(list_of_item)

    empty_bag -> queue

    max_state = empty_bag

    while queue is not empty

        current_state <- queue.pop()

        if value of current_state > value of max_state

            and weight of current_state <= bag_capacity

            and current_state has contain all item class

        then max_state = current_state

        for every item hasn't been in current_state
```

next_state = current_state + item

next_state -> queue

return max_state

C. Bảng giá trị đã thực hiện:

Test case(n)	7	8	9	10	20	50	100	150	200	250	300
Thời gian(s)	0,13 86	0,92 6	8.05	66,2 36	#	#	#	#	#	#	#

: thời gian chờ quá lâu, máy tính không tính được

D. Hình ảnh minh chứng:

```
Brute_Force
Total memory: 106 KB
- n = 7 : Running time: 0.13862943649291992 s
```

```
Brute_Force
Total memory: 1291 KB
- n = 8: Running time: 0.9259438514709473 s
```

```
Brute_Force
Total memory: 12399 KB
- n = 9: Running time: 8.050220727920532 s
```

```
dapter/../../debugpy\launcher' '53690'
1
Brute_Force
Total memory: 124789 KB
- n= 10: Running time: 66.2362425327301 s
```

Những test case với n lớn thì thời gian chờ quá lâu, máy không đủ để xử lý được

E. Đánh giá chung về thuật toán Brute Force:

- Qua các trường hợp thử nghiệm trên, chúng ta có thể dễ dàng nhận thấy rằng Brute force hoạt động tốt trong các tập dữ liệu quy mô nhỏ và trở nên không

thực tế khi đối mặt với các quy mô lớn. Đây là nhược điểm khó tránh khỏi của thuật toán brute force.

- Thuật toán brute force là một cách tiếp cận đơn giản để giải quyết bài toán Knapsack . Nó xem xét một cách có hệ thống tất cả các kết hợp có thể có của các mặt hàng và tính toán tổng giá trị, trọng lượng của chúng. Sau đó, nó chọn tổ hợp có giá trị cao nhất không vượt quá khả năng chứa của ba lô.
- Độ phức tạp : $O(n^n)$
- Đường link video: <https://www.youtube.com/watch?v=34fZhyLFApE>

2. Algorithm 2: branch and bound

A. Mô tả thuật toán :

- Thuật toán Branch and Bound là một phương pháp tìm kiếm trên cây (tree search) được sử dụng để giải quyết các bài toán tối ưu trong đó các bước giải quyết liên quan tới việc phân chia các phần tử, tập con hoặc các điều kiện.
- Khi xét một chỉnh hợp các vật phẩm có tổng khối lượng lớn hơn khối lượng được giới hạn, dù cho chúng ta có thêm bất kì vật phẩm nào đi chăng nữa thì tổng khối lượng của chúng đều lớn hơn khối lượng giới hạn được cho phép ($w > W_m \Leftrightarrow w + w_i > W_m$, với w là khối lượng đang xét, w_i là khối lượng đồ vật được cho thêm, $w_i > 0$ và W_m là khối lượng sức chứa tối đa). Vì vậy chúng ta sẽ không cần xét các nút con của nút hiện đang có sức chứa lớn hơn khối lượng cho phép
- Thêm nữa, việc xét bất kì vật phẩm cho thêm vào nút hiện tại sẽ tạo ra các hoán vị của một nút bất kì trên cây. Để có thể tránh các trường hợp này, chúng ta sẽ xét việc lấy các vật phẩm sẽ theo thứ tự tăng dần. Vì vậy, tại mỗi nút sẽ là một dãy tăng của các vật phẩm được lấy, và sẽ không xuất hiện các hoán vị nào khác. Điều này sẽ giảm số nút cần xét từ n^n còn $n!$

B. Mã giả của thuật toán:

```
func branch_bound(list_of_item)
```

```
    empty_bag -> queue
```

```
    max_state = empty_bag
```

```

while queue is not empty

    current_state <- queue.pop()

    if value of current_state > value of max_state

        and current_state has contain all item class

    then max_state = current_state

    for every item after last item in current_state

        next_state = current_state + item

        if weight of next_state <= bag_capacity

            next_state -> queue

return max_state

```

C. Bảng giá trị đã thực hiện

Test case(n)	10	15	20	30	35	50	100	150	200	250	300
Thời gian(s)	0,004	0,661	0,003	0,003	0,003	83,606	#	#	#	#	#

: thời gian chờ quá lâu, máy tính không tính được

D. Hình ảnh minh chứng

n = 10

```

Branch and bound
1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0
Total memory: 12 KB
Running time: 0.003991365432739258 s

```

n=15

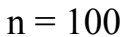
```

Branch and bound
961
1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1
Total memory: 867 KB
Running time: 0.6611340045928955 s

```



```
Branch and bound
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1
Total memory: 13 KB
Running time: 0.0029571056365966797 s
```

[illegible][illegible][illegible]

E. Đánh giá thuật toán

- 9

dataset lớn hơn. Tuy nhiên thuật toán này vẫn không khả thi với các dataset đủ lớn.

- Đường link video: <https://www.youtube.com/watch?v=2giKdJv1sLg>

3. Algorithm 3: local beam search

A. Mô tả thuật toán:

- Thuật toán Local beam search là thuật toán tìm kiếm trên đồ thị, mở rộng k nút tốt nhất theo heuristic và dần dần dần tụ lại vào một giá trị mà thuật toán cho là đúng nhất
- Đầu tiên ta sẽ tạo ra các giá trị khởi tạo và mở rộng các nút tại thời điểm đó, các giá trị này được xác định là các nút có lấy duy nhất một loại vật phẩm. Đẩy các giá trị trên vào một hàng đợi ưu tiên, chúng ta lấy tối đa $2 * n$ nút từ hàng đợi ưu tiên và xóa các giá trị còn lại. Mở rộng các nút được lấy ở trên và đẩy chúng vào hàng đợi ưu tiên, các nút vượt quá sức chứa sẽ không được cho vào. Các nút được mở rộng sẽ có 50% tỉ lệ đem vào hàng đợi Random Restart. Lặp lại cho tới khi các hàng đợi ưu tiên lần đầu được trống, hàng đợi ưu tiên sẽ được thay thế bởi hàng đợi Random Restart và chạy lại. Giá trị tốt nhất từ các lần chạy này sẽ cho ra các kết quả được cho là tốt nhất.
- Hàm Heuristic được tính theo giá trị của các vật phẩm được lấy hiện tại và được ưu tiên bởi số lượng loại vật phẩm đã có.

B. Mã giả thuật toán

```
func local_beam(item_list)

    n = size of item_list

    push initialized states -> priority_queue

    random_restart_queue

    maximum_node = max_value of initialized states

    while priority_queue is not empty

        take 2 * n first element of priority_queue

        for each element

            if element has all class
```

```

        and element.value > maximum_node.value
    then
        maximum_node = element
    expand all subnode from element
    if subnode.capacity < max_capacity
        if random_pass
            subnode -> random_restart_queue
            subnode -> priority_queue
    if priority_queue is empty for the first time
        random_restart_queue -> priority_queue

    return maximum_node

```

C. Bảng giá trị đã thực hiện

Test case(n)	10	15	20	30	35	50	100	150	200	250	300
Thời gian(s)	0,003	0,739	0,004	0,004	0,15	0,107	0,214	#	#	#	#

: thời gian chờ quá lâu, máy tính không tính được

D. Hình ảnh minh chứng

n=10

```
Local beam search
215
1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0
Total memory: 9 KB
Running time: 0.002965688705444336 s
```

n=15

```
Branch and bound
961
1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1
Total memory: 867 KB
Running time: 0.7390191555023193 s
```

n=20

```
Local beam search
210
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1
Total memory: 8 KB
Running time: 0.003941059112548828 s
```

n=30

```
Local beam search  
0  
0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0  
Total memory: 8 KB  
Running time: 0.003989696502685547 s
```

n=35

```
Local beam search
1096
0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 ,0 ,1 ,1 ,0 ,1 ,0 ,0 ,0 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1
Total memory: 94 KB
Running time: 0.15049386024475098 s
```

n=50

```
Local beam search
972
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0
Total memory: 140 KB
Running time: 0.10741543769836426 s
```

n=100

[illegible]

E. Đánh giá thuật toán

- Với bảng giá trị đã xét ở trên, có thể nhận ra thuật toán local beam search chạy rất nhanh và tiết kiệm bộ nhớ hơn so với 2 thuật toán đã được nêu ra ở trên. Điều này khiến cho thuật toán này có thể chạy ổn định trên mọi dataset, kể cả các dataset rất lớn.
- Tuy nhiên, kết quả của thuật toán này lại không phải lúc nào cũng cho ra kết quả tối ưu nhất
- Đường link video : <https://www.youtube.com/watch?v=Qc44gkDC9dk>

4. Algorithm 4: genetic algorithms

A. Mô tả thuật toán:

- khởi tạo: population size, tỉ lệ mutation, số lượng thế hệ tạo ra
- thuật toán chỉ giới hạn số lượng tạo ra thế hệ sau ở một mức độ đã được gán ban đầu, được bắt đầu bằng cấp lập qua số thế hệ đã chỉ định
- đối với mỗi thế hệ, hai nhiễm sắc thể được chọn từ quần thể, sau đó trao đổi chéo được thực hiện ở một index được random (em sẽ gọi đó là crossover point), thì child1 sẽ được ghép từ index thứ 0 đến crossover point được random ra, child thứ hai sẽ được chọn từ crossover-point cho đến chiều dài của nhiễm sắc thể con thứ hai
- Sau đó chọn ngẫu nhiên 1 index để thay đổi gene ngay tại index đó, nếu là 1 thì đổi về 0, nếu là 0 thì đổi về 1
- các nhiễm sắc thể mới có thể bị "đột biến gen" ngẫu nhiên riêng lẻ nếu xác suất được tạo ngẫu nhiên vượt quá ngưỡng xác định trước
- Cuối cùng, quần thể cũ được thay thế bằng quần thể mới, bao gồm hai nhiễm sắc thể mới và các nhiễm sắc thể còn lại từ quần thể cũ
- với số lượng population size và generations được generate với số lượng càng nhiều thì các tăng suất xuất hiện của các chuỗi nhiễm sắc thể sẽ được tính toán chính xác hơn.
- một tập hợp các giải pháp khả thi (hoặc thế hệ ban đầu) được tạo ngẫu nhiên và sau đó được đánh giá dựa trên một tập hợp các tiêu chí. Sau đó, những giải pháp phù hợp nhất với tiêu chí sẽ được chọn và các đột biến gen được áp dụng để tạo ra các biến thể giải pháp mới (hoặc các thế hệ tiếp theo). Thế hệ biến

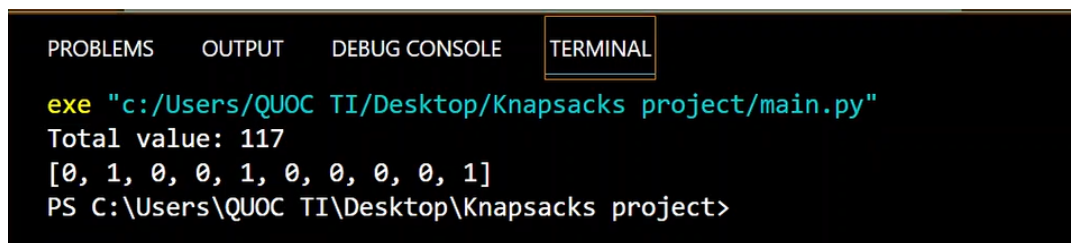
thể mới này sau đó được đánh giá và quá trình này được lặp lại cho đến khi tìm ra giải pháp thỏa đáng. Quá trình này được lặp lại cho đến khi tìm được giải pháp tối ưu hoặc gần nhất với giải pháp tối ưu.

B. Mã giả thuật toán :

- #Initialize
- weight_limit, number_of_classes, number_of_items, list_item = Input.read()
- population_size = 500
- mutation_probability = 0.2
- generations = 500
- ListOfChromosome = GeneratePopulation(population_size, len(list_item), number_of_items)
- for j in range(generations):
 - ❖ select two chromosomes for crossover
 - ❖ crossover to generate two new chromosomes
 - ❖ mutation on the two new chromosomes
 - ❖ replace the old population with the new population
- get the best chromosome from the population
- get the weight and value of the best solution

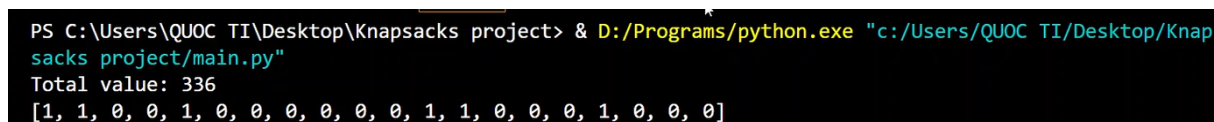
C. Hình ảnh minh chứng :

- n = 10



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
exe "c:/Users/QUOC TI/Desktop/Knapsacks project/main.py"
Total value: 117
[0, 1, 0, 0, 1, 0, 0, 0, 0, 1]
PS C:\Users\QUOC TI\Desktop\Knapsacks project>
```

- n = 20



```
PS C:\Users\QUOC TI\Desktop\Knapsacks project> & D:/Programs/python.exe "c:/Users/QUOC TI/Desktop/Knapsacks project/main.py"
Total value: 336
[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0]
```

- n = 30

```
PS C:\Users\QUOC TI\Desktop\Knapsacks project> & D:/Programs/python.exe "c:/Users/QUOC TI/Desktop/Kna
sacks project/main.py"
Total value: 779
[1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1]
```

- $n = 35$

```
PS C:\Users\QUOC TI\Desktop\Knapsacks project> & D:/Programs/python.exe "c:/Users/QUOC TI/Desktop/Knapsacks project/main.py"
Total value: 6370
[0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0]
```

- $n = 50$

```
PS C:\Users\QUOC TI\Desktop\Knapsacks project> & D:/Programs/python.exe "c:/Users/QUOC TI/Desktop/Knapsacks project/main.py"
```

Total value: 3405

[0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

- $n = 100$

```
PS C:\Users\QUOC TI\Desktop\Knapsacks project> & D:/Programs/python.exe "c:/Users/QUOC TI/Desktop/Knap
sacks project/main.py"
Total value: 12102
[0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1]
```

- $n = 300$

[illegible]

D. Đánh giá thuật toán :

- Khi nghiên cứu cũng như tìm hiểu, theo cách em hiểu đối với genetic algorithm, đúng với tên gọi thuật toán, nó giống như tập hợp các mã di truyền, đối với thuật toán thì các mã di truyền này được sinh ngẫu nhiên, vì vậy theo em nhận xét, kết quả của thuật toán cũng sẽ đưa ra output mang tính tương đối. Nó có thể ra kết quả chính xác nhất, cũng có thể đưa ra kết quả gần nhất với kết quả chính xác nhất.
- Độ phức tạp của thuật toán Genetic Algorithm là $O(g(2n*m+n))$ với g là số lượng thế hệ sau, n là population size và m là độ lớn của từng phần tử
- Ưu điểm của Genetic Algorithms là có thể tìm thấy một giải pháp đủ tốt một cách nhanh chóng mà không cần phải tìm kiếm một cách thấu đáo trong tất cả các giải pháp có thể. Điều này làm cho nó trở thành một cách tiếp cận hiệu quả hơn nhiều so với các thuật toán truyền thống và cho phép tìm ra giải pháp nhanh hơn nhiều.
- Nâng cấp thuật toán bằng cách kết hợp giữa Roulette Wheel và Tournament trong thuật toán Genetic Algorithm. Thay vì chọn ra ngẫu nhiên nhiễm sắc thể cha và mẹ, em sẽ bổ sung thêm bước loại trừ đi nhiễm sắc thể không thỏa yêu cầu đề bài, rồi tiếp tục chọn cặp nhiễm sắc thể cha và mẹ rồi làm tương tự. Khi đó mỗi lần chọn nhiễm sắc thể cha mẹ chúng ta có thể loại trừ những nhiễm sắc thể không đạt yêu cầu. Do đó, giảm thời gian cũng như không gian bộ nhớ cần để thực hiện thuật toán.
- Video : <https://www.youtube.com/watch?v=CYG2sbU5UpM>

VI : References: (Nguồn tham khảo)

- kdnuggets: knapsack-problem-genetic-programming-python
- [The Knapsack Problem & Genetic Algorithms - Computerphile](#)