***Biên bản buổi thảo luận***

***Nhóm 9***

1. ***Quá trình trên lớp:***

- Nhóm thuyết trình về Greedy Approach.

- Nhóm lần lượt đặt ra các vấn đề để các bạn trong lớp cùng giải quyết, sau đó giải đáp.

- Các nhóm tham gia trao đổi, đóng góp ý kiến : Nhóm 7, Nhóm 4, Nhóm 5, Nhóm 3.

- Nhóm 4 đưa ra câu hỏi: Các bài toán con trong Greedy Approach có gối nhau hay không ?

\* Câu trả lởi của nhóm: Các bài toán con trong Greedy Approach không gối nhau.

- Nhóm 3 đưa ra câu hỏi: Độ phức tạp được tính như thế nào ?

\* Câu trả lời của nhóm: (Nhóm chưa trả lời được câu hỏi này trên lớp)

1. ***Đánh giá bài tập về nhà:***

- Giải bài toán cái túi (Knapsack Problems).

- Đặt ra câu hỏi liên quan đến Greedy Approach.

**\* Nhóm 1:**

- Code: Đã nộp

- Câu hỏi: Tại sao thuật toán tham lam nhiều lúc đưa ra kết quả không chính xác nhưng người ta vẫn sử dụng ?

- Trả lời : người ta vẫn sử dụng vì các ưu điểm của thuật toán là chạy nhanh hơn các thuật toán khác, dễ tiếp cận và dễ thiết kế hơn.

**\* Nhóm 3:**

- Code: Đã nộp

- Câu hỏi: Đánh giá độ phức tạp thuật toán như thế nào ?

- Trả lời : Độ phức tạp thuật toán bằng tổng độ phức tạp của các hàm thành phần (cộng thêm độ phức tạp của hàm sắp xếp nếu có).

\* **Nhóm 4:**

- Code: (Nộp trễ)

- Câu hỏi: Không có

**\* Nhóm 5:**

- Code: Đã nộp

- Câu hỏi: Không có

**\* Nhóm 6:**

- Code: Đã nộp

- Câu hỏi:

1. Hàm khả thi là gì ?

2. Tại sao phải dùng hàm khả thi trong Greedy Approach ?

3. Không có hàm khả thi có áp dụng được phương pháp Greedy được hay không ?

- Trả lời:

+ Hàm khả thi là hàm dùng để xét xem một ứng cử viên có thỏa mãn một điều kiện nào đó hay không, nếu thỏa thì sẽ được dùng để xây dựng lời giải.

+ Phải dùng hàm khả thi vì nó là hàm để xét các ứng cử viên tham gia vào việc xây dựng lời giải, nó ảnh hưởng lớn đến kết quả cuối cùng.

+ Như trình bày ở trên thì hàm khả thi rất quan trọng và cần phải có khi sử dụng Greedy.

**\* Nhóm 7:**

- Code: Đã nộp

- Câu hỏi: Trong slide có nói nhược điểm của Greedy Approach là có thể đưa ra kết quả bài toán không chính xác. Vậy làm thế nào để xác định 1 bài toán có thể dùng phương pháp Greedy Approach có cho ra kết quả chính xác hay không ?

- Trả lời: Ta sẽ xét các trường hợp nhỏ cụ thể rồi đánh giá xem là Greedy Approach chạy có chính xác hay không.

**\* Nhóm 8:**

- Code: Đã nộp

- Câu hỏi: Các thành phần của Greedy trong một bài toán cụ thể.

- Trả lời: Lấy ví dụ bài Knapsack Problem trong slide:

+ Tập ứng cử viên sẽ là mảng với giá trị value/weight.

+ Hàm lựa chọn: ta sẽ sắp xếp mảng theo giá trị value/weight giảm dần, sau đó lần lượt chọn các giá trị đó để đưa vào hàm khả thi.

+ Hàm khả thi : if (weight[i] <= remaining\_weight):

remaining\_weight -= weight[i]

+ Hàm mục tiêu: bag\_value += value[i]

+ Hàm đánh giá: hàm đánh giá ở đây là while(i < n) để duyệt qua hết các phần tử trong mảng.

**\* Nhóm 10:**

- Code: Đã nộp

- Câu hỏi: Khi dùng hướng tiếp cận tham lam, làm sao để xác định được cấu trúc con tối ưu cho bài toán?

- Trả lời : Sau khi đươc chọn bởi hàm lựa chọn, nếu ứng cử viên thỏa điều kiện của hàm khả thi và được sử dụng để xây dựng lời giải thì đây là lời giải tối ưu của một bài toán con, tổng hợp các lời giải này ta được lời giải tối ưu của bài toán lớn.

**\* Nhóm 11:**

- Code: Đã nộp

- Câu hỏi: Ưu điểm lớn nhất của Greedy Approach là gì ?

- Trả lời: Nó chạy nhanh hơn các thuật toán khác.

****

* Để thuận tiện cho việc kiểm tra, đánh giá code của các nhóm cũng như để so sánh và hiểu rõ hơn về cách thuật toán **Tham lam** hoạt động. Chúng em đã viết một vài đoạn code như sau:

+ Các hàm giải quyết bài toán **Knapsack Problem** sử dụng kỹ thuật **Greedy Appoach**.

+ Tạo bộ testcase đơn giản bằng thư viện random.

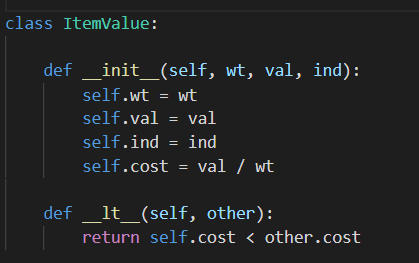
+ Hàm kiểm tra và chấm điểm code các nhóm submit dựa trên testcase.

* + - 1. Testcase nhỏ để kiểm tra hoạt động chương trình.
      2. Testcase lớn để kiểm tra hiệu năng chương trình.

1. **Giải quyết Knapsack Problem bằng Greedy Appoach**

Theo như ý tưởng cũng như giải thuật đã trình bày, giới thiệu trên buổi báo cáo trực tiếp trên lớp học. Bài toán giải quyết bằng kỹ thuật tham lam và tiếp cận theo tỉ số Trị giá / Cân nặng từ lớn tới nhỏ.

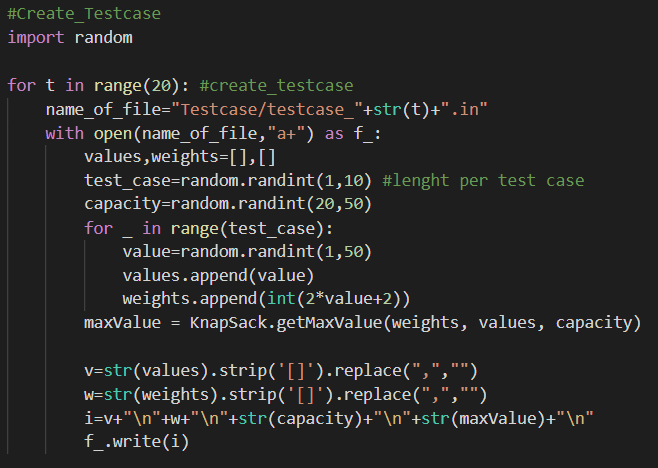
* + Một class nhỏ để quản lý các thuộc tính về trị giá, cân nặng, tỉ số.

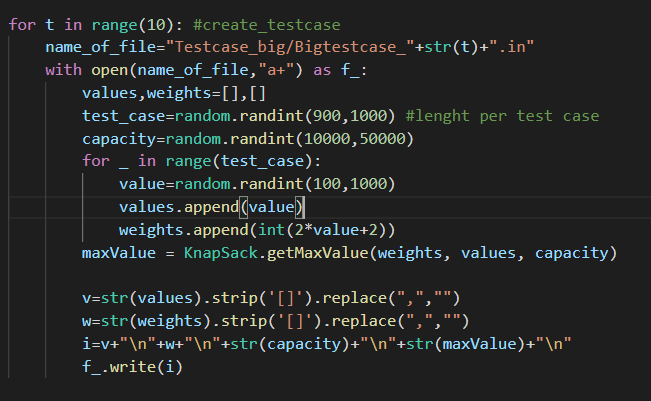


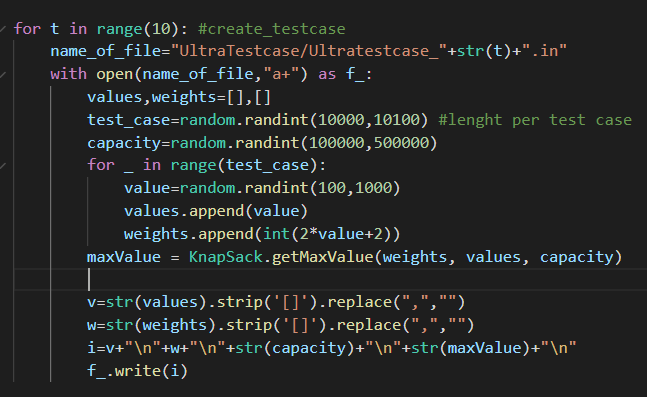
* + Một class Knapsack và hàm giải quyết kèm theo.



1. **Tạo Testcase**
   * Ở đây em tạo bộ testcase đơn giản bằng hàm random và không cần sử dụng thư viện unitest.
   * Sử dụng hàm random để tạo các giá trị như trị giá, số lượng mặt hàng trong mỗi testcase,…
   * Bộ test case nhỏ gồm có 20 test case có tối đa 10 cặp value/weight được lưu thành 20 file .txt

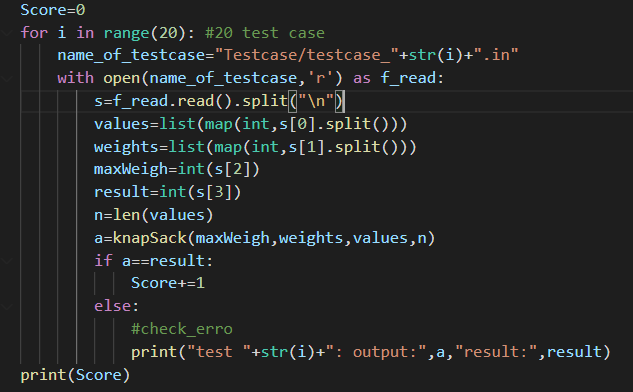


* + Bộ test case lớn gồm 10 test case có tối đa 1000 cặp value.weight 
  + Đối với các chương trình đã vượt qua hơn 50% test case lớn trên sẽ được tiếp tục kiểm tra thông qua bộ test cuối cùng, đây là bộ test có độ lớn hơn hơn nhiều lần nhằm kiểm tra được khả năng tối ưu tính toán. Bao gồm 10 test case có tối đa 10000 cặp value/weight.

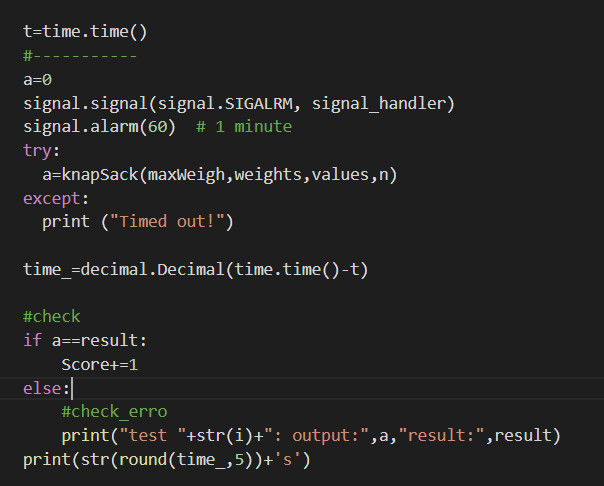


* + Để đơn giản hóa và trực quan dữ liệu giữa trị giá và cân nặng cho phù hợp, em đã chọn hàm số bậc nhất y=2.x + 2 để mô tả cho quan hệ này. Điều này có thể gây ra sự tương đồng giữa các cặp trị giá – cân nặng, nhưng nó lại dể mô tả, trong lúc kiểm tra code.

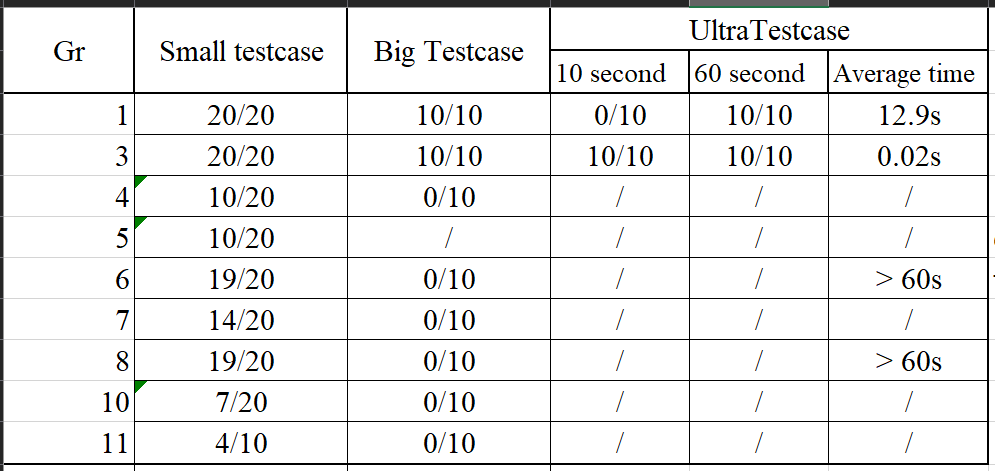
1. **Kiểm tra code các nhóm submit**
   * Hàm kiểm tra kết quả giữa code submit và testcase dựa vào số lượng test case đúng thông qua hàm check result.



* + Để hạn chế thời gian chạy cũng như tránh trường hợp chương trình chạy quá lâu thì phần BigTestcase và UltraTestcase sẽ được thêm các lệnh đếm thời gian thông qua 1 số thư viện và các thư viện này chỉ hoạt động trên linux nên cần sử dụng colab để test.



* Kết quả:



* Nhận xét:
  + Dựa vào kết quả cho thấy nhóm 3 là nhóm có kết quả tốt nhất.
  + Đa số các nhóm đều vượt qua small testcase (>=50%).
  + Chỉ có duy nhất 2 nhóm vượt quá Big testcase với max score nhưng chỉ có nhóm 3 là xuất sắc vượt qua Ultratestcase với thời gián ấn tượng.
  + Các nhóm không thể vượt qua Bigtestcase thì 1 số là do chưa đúng ở minTestcase (nhóm 4,5,7,11) còn 1 số nhóm thì do code chưa hiệu có hiệu năng cao nên bị time limit do sử dụng kỹ thuật đệ quy (nhóm 6,8).

**Tóm lại:**

* **Giải quyết bài toán bằng tham lam rất nhanh, dể cài đặt.**
* **Nếu không yêu cầu về độ chính xác tuyệt đối, ở mức tham khảo xấp xỉ kết quả thì tham lam thực sự hiệu quả.**