**KỊCH BẢN**

Chương trình được viết trên Dev-C++.Các chương trình sẽ tạo tệp thực thi(.exe), giúp người dùng truyền tham số - là các file dữ liệu. **GenerateItem.exe** nhận tham số là một tệt .txt, cho phép người dùng nhập số ***giá* *trị*** và ***khối* *lượng* *tối* *đa*** có thể chứa của túi**,** sau đó sẽ tạo ngẫu nhiên dữ liệu về ***khối* *lượng*** và ***giá trị*** của các đồ vật rồi lưu vào file **knapsack\_x \_number.txt**.

Chương trình tìm kiếm sẽ đọc dữ liệu từ file và tiến hành đánh giá các thuật toán sau: Bruceforce( vét cạn), Greed1 ( theo giá trị giảm dần), Greed2 ( theo khối lượng tăng dần), Greed3 ( theo trọng số mỗi đồ vật).

**Brute Force ( Vét cạn)**

Thuật toán Brute Force này tạo ra tất cả các tập con của tập hợp các món hàng và kiểm tra từng tập con xem tổng khối lượng của nó có vượt quá sức chứa của cái túi hay không. Nếu không vượt quá, nó tính tổng giá trị của tập con đó và so sánh với giá trị tốt nhất hiện tại. Nếu giá trị của tập con đó lớn hơn giá trị tốt nhất hiện tại, nó lưu giá trị của tập con đó và cập nhật giá trị tốt nhất.

for(i=0;i<iter;i++) {

    isSelected=1;

    maxProfit=0;

    maxWt=0;

    for(j=0;j<size;j++){

        if(temp[j]==1){

            maxWt+=weight[j];

            maxProfit+= profit[j];

        }

    }

    if( maxWt <= capacity && maxProfit>result){

        /\*Copies the selected item in array. \*/

        for(j=0;j<size;j++)

            selected[j]=temp[j];

        result=maxProfit;

    }

    /\*Calculate the next subset \*/

    for(j=0;j<size;j++){

        temp[j]=temp[j]+isSelected;

        isSelected = temp[j]/2;

        temp[j]=temp[j]%2;

    }

}

return result;

Thuật toán liệt kê ***2^n*** tập hợp con, với mỗi tập hợp con sẽ lặp lại ***n*** lần để tính tổng khối lượng và giá trị. Thực hiện tính tổng tất cả các tập hợp con và so sánh với khối lượng giới hạn

Độ phức tạp của thuật toán là O(2^n\*n).

Độ phức tạp không gian là O(n)

**Greed 1 ( theo giá trị giảm dần)**

Phương pháp Greed1, sắp xếp các vật phẩm theo thứ tự giảm dần của giá trị và lưu vào một mảng tạm thời, sau đó duyệt qua các vật phẩm này từ trên xuống dưới, chọn ra các vật phẩm có thể chứa vào túi và tính toán lại trọng lượng giới hạn của túi. Khi một vật phẩm được chọn, ta lưu lại thông tin vật phẩm đó đã được chọn vào mảng selected.

for(i=0;i<size;i++) {

  tempArr[i]=i;

    tempWt[i]=weight[i];

    tempProfit[i]=profit[i];

    selected[i]=0;

}

for(i=0;i<size;i++) {

    for(j=0;j<size - i-1;j++) {

        if(tempProfit[j] < tempProfit[j+1]){

            //swap profit

            temp=tempProfit[j];

            tempProfit[j]=tempProfit[j+1];

            tempProfit[j+1]=temp;

            //swap weights

            temp=tempWt[j];

            tempWt[j]=tempWt[j+1];

            tempWt[j+1]=temp;

            //swap item number as well

            temp=tempArr[j];

            tempArr[j]=tempArr[j+1];

            tempArr[j+1]=temp;

        }

    }

}

//sorted

for(i=0;i<size;i++) {

    if(tempWt[i] <= greedyCap && greedyCap > 0) {

        result+=tempProfit[i];

        greedyCap -=tempWt[i];

        selected[tempArr[i]]=1;

    }

}

return result;

Sắp xếp danh sách các đồ vật dựa trên giá trị của nó có độ phức tạp là O(n^2). Với các đồ vật đã theo thứ tự xắp sếp này, tính tổng giá trị và khối lượng. Có độ phức tạo là O(n).

Độ phức tạp của thuật toán là O(n^2).

Độ phức tạp không gian là O(n)

**Greed 2 ( theo khối lượng giảm dần)**

Thuật toán **s**ắp xếp các món đồ theo thứ tự tăng dần của trọng lượng, sau đó chọn các món đồ có trọng lượng nhỏ nhất trước đến khi nào đạt tới giới hạn trọng lượng của túi. Việc lựa chọn này đảm bảo rằng túi sẽ được đựng đầy nhiều nhất có thể, nhưng có thể không đem lại giá trị lợi nhuận cao nhất.

for(i=0;i<size;i++) {

    for(j=0;j<size - i-1;j++) {

        if(tempWt[j] > tempWt[j+1]){

            //swap profit

           temp=tempProfit[j];

            tempProfit[j]=tempProfit[j+1];

          tempProfit[j+1]=temp;

            //swap weights

            temp=tempWt[j];

            tempWt[j]=tempWt[j+1];

            tempWt[j+1]=temp;

            //swap item number as well

            temp=tempArr[j];

            tempArr[j]=tempArr[j+1];

            tempArr[j+1]=temp;

        }

    }

}

for(i=0;i<size;i++) {

    if(tempWt[i] <= greedyCap && greedyCap > 0) {

        result+=tempProfit[i];

        greedyCap -=tempWt[i];

      selected[tempArr[i]]=1;

    }

}

return result;

Sắp xếp danh sách các đồ vật dựa trên giá trị của nó có độ phức tạp là O(n^2). Với các đồ vật đã theo thứ tự xắp sếp này, tính tổng giá trị và khối lượng. Có độ phức tạo là O(n).

Độ phức tạp của thuật toán là O(n^2).

Độ phức tạp không gian là O(n)

**Greed 3 ( theo trọng số mỗi đồ vật)**

Thuật toán sắp xếp các món hàng theo tỉ lệ giá trị trên trọng lượng của chúng, bắt đầu bằng cách tính toán giá trị trên trọng lượng của từng món hàng. Sau đó, sắp xếp các món hàng theo giá trị giảm dần của tỉ lệ giá trị trên trọng lượng này. Cuối cùng, chọn các món hàng bằng cách bắt đầu với món hàng có tỉ lệ giá trị trên trọng lượng cao nhất, sau đó tiếp tục chọn món hàng có tỉ lệ giá trị trên trọng lượng cao nhất còn lại và lặp lại cho đến khi không thể chọn thêm món hàng nào nữa.

    for(i=0;i<size;i++) {

        for(j=0;j<size - i-1;j++) {

            val1 = ((float)tempProfit[j]/ (float)tempWt[j]);

            val2 = ((float)tempProfit[j+1]/(float) tempWt[j+1]);

            if( val1 < val2 ){

                //swap profit

                temp=tempProfit[j];

                tempProfit[j]=tempProfit[j+1];

                tempProfit[j+1]=temp;

                //swap weights

                temp=tempWt[j];

                tempWt[j]=tempWt[j+1];

                tempWt[j+1]=temp;

                //swap item number as well

                temp=tempArr[j];

                tempArr[j]=tempArr[j+1];

                tempArr[j+1]=temp;

            }

        }

    }

    for(i=0;i<size;i++) {

        if(tempWt[i] <= greedyCap && greedyCap > 0) {

            result+=tempProfit[i];

            greedyCap -=tempWt[i];

            selected[tempArr[i]]=1;

        }

    }

    return result;

Sắp xếp danh sách các đồ vật dựa trên giá trị của nó có độ phức tạp là O(n^2). Với các đồ vật đã theo thứ tự xắp sếp này, tính tổng giá trị và khối lượng. Có độ phức tạo là O(n).

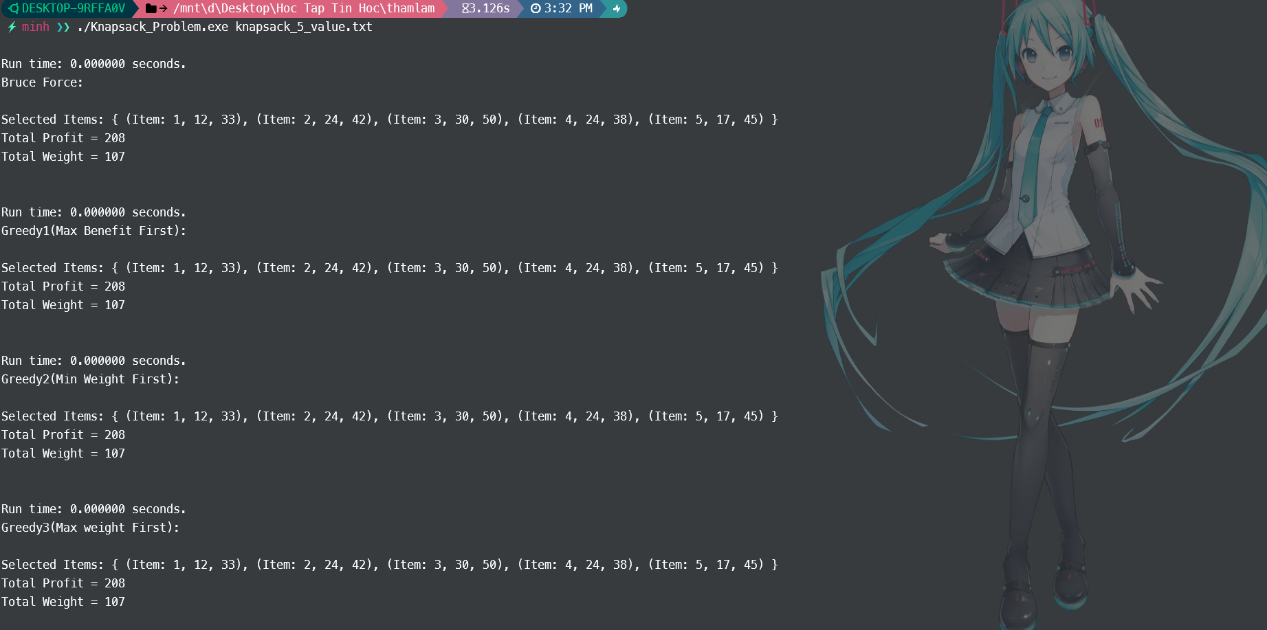
Độ phức tạp của thuật toán là O(n^2).

Độ phức tạp không gian là O(n)

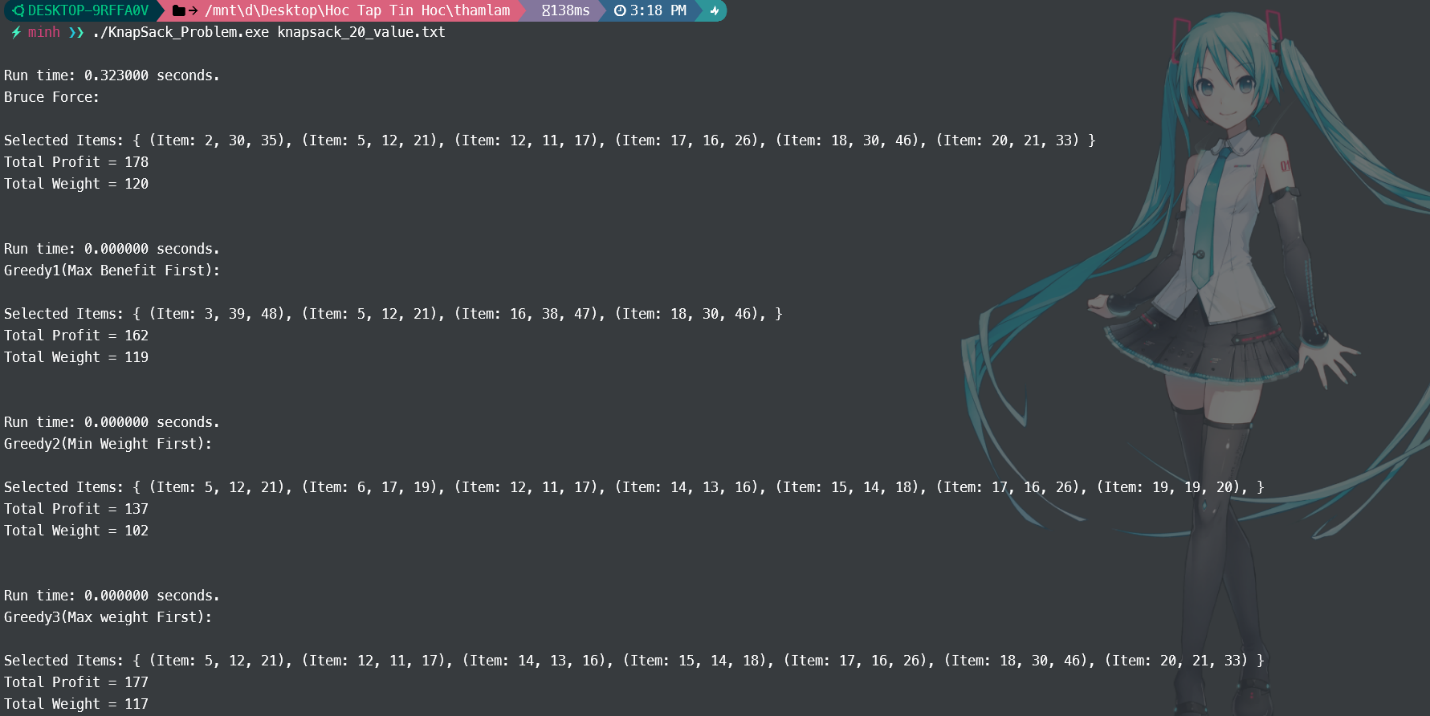
**Kết quả thực nghiệm**

**Lần chạy thứ 1:**

* Với số lượng đồ vật là 5, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, các phương pháp đều tìm được kết quả **tối ưu như nhau**.



* Với số lượng đồ vật là 10, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, tuy nhiên tại **Greed 2**, kết quả giá trị lấy được có **độ chính xác tốt hơn** hai phương pháp còn lại
* Với số lượng đồ vật là 20, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, riêng phương pháp vét cạn tốn nhiều thời gian hơn so vơi các phương pháp khác. Tuy nhiên tại **Greed 3**, kết quả giá trị lấy được có **độ chính xác tốt hơn** hai phương pháp còn lại.

****

* Với số lượng đồ vật là 30, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, nhưng với phương pháp vét cạn – độ phức tạp là **O**(**2^n\*n)**, rất cao dẫn đến chênh lệch nhiều về thời gian
* Tại phương pháp **Greed 3**, kết quả giá trị lấy được có độ chính xác tốt hơn hai phương pháp còn lại.

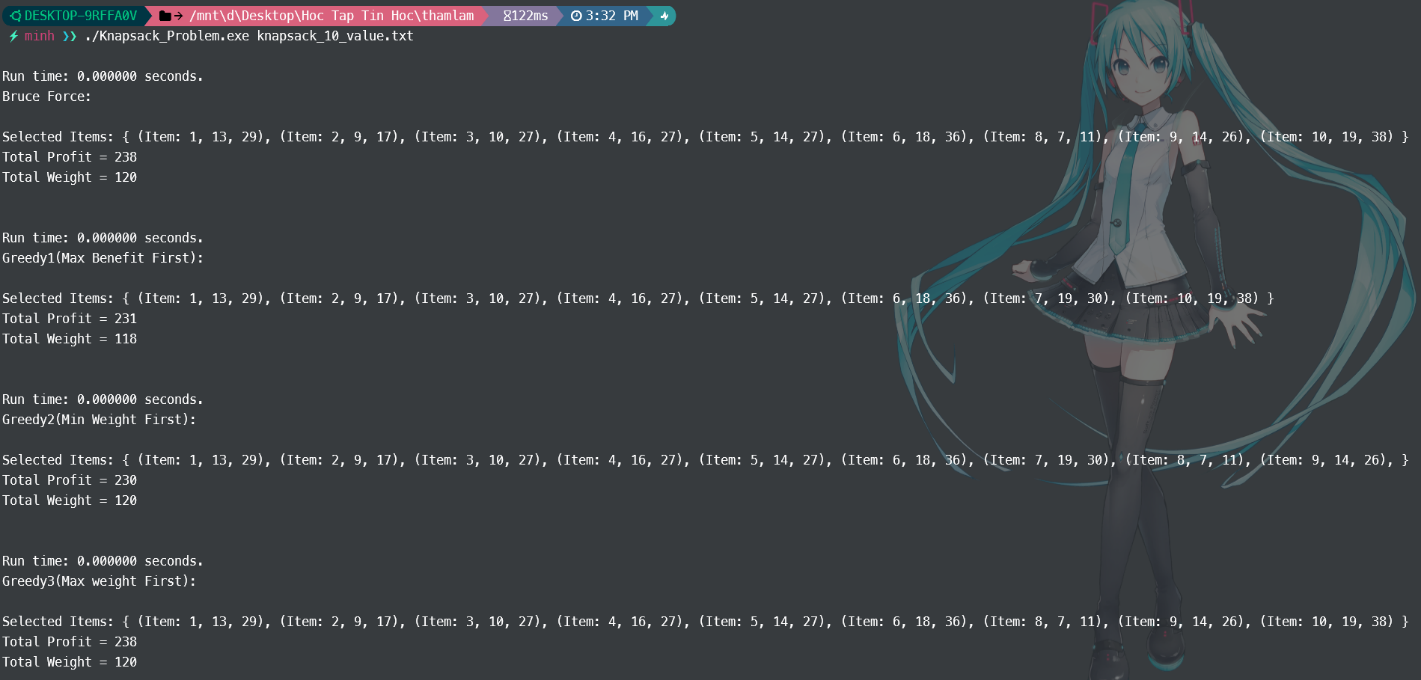


**Lần chạy thứ 2:**

* Với số lượng đồ vật là 5, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, các phương pháp đều tìm được kết quả tối ưu như nhau.

****

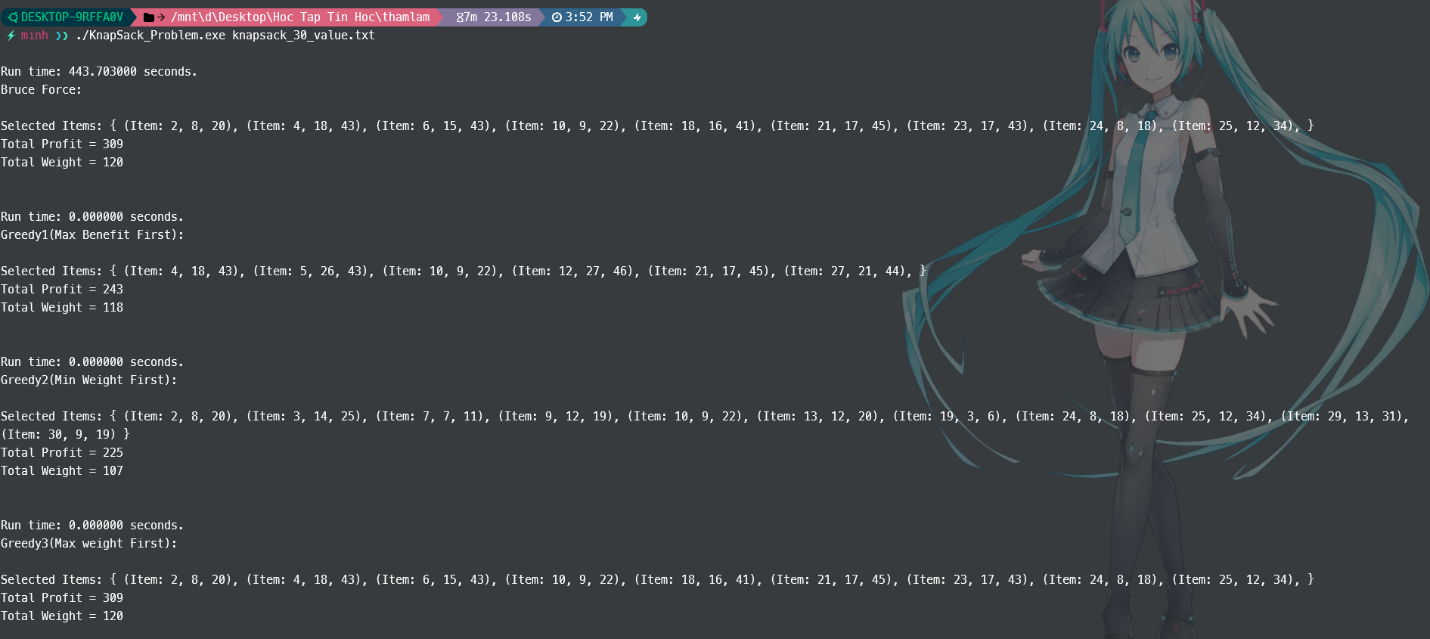
* Với số lượng đồ vật là 10, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, tuy nhiên tại **Greed 1**, kết quả giá trị lấy được chưa đạt kết qua tối đa. Phương pháp **Greed 2 và Greed 3 tìm được kết quả tối ưu**.



* Với số lượng đồ vật là 20, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, tuy nhiên tại **Greed 3**, kết quả giá trị lấy được đạt **kết quả tối ưu**.



* Với số lượng đồ vật là 30, ta nhận thấy ta thấy tốc độ của các phương pháp đồng đều, nhưng với phương pháp vét cạn – độ phức tạp là **O**(**2^n\*n)**, rất cao dẫn đến chênh lệch nhiều về thời gian
* Tại phương pháp **Greed 3**, kết quả giá trị lấy được có độ chính xác tốt hơn hai phương pháp còn lại.



**Kết luận:**

* Các thuật toán vét cạn và Greedy 1, 2, 3 đều được sử dụng để giải quyết bài toán cái túi 0-1. Tuy nhiên, các thuật toán này có những ưu và nhược điểm khác nhau.
* Thuật toán vét cạn cho phép tìm kiếm tất cả các phương án có thể của bài toán và tìm ra phương án tối ưu. Tuy nhiên, độ phức tạp tính toán của nó rất cao và không hiệu quả khi số lượng đồ vật lớn.
* Greedy 1 sắp xếp các đồ vật theo giá trị lợi nhuận lớn nhất đầu tiên, sau đó chọn các đồ vật này cho đến khi không thể chọn thêm nữa. Tuy nhiên, nó có thể dẫn đến việc chọn ra một tập hợp các đồ vật không tối ưu.
* Greedy 2 sắp xếp các đồ vật theo trọng lượng tăng dần đầu tiên, sau đó chọn các đồ vật này cho đến khi không thể chọn thêm nữa. Nó có thể cho kết quả tốt hơn so với Greedy 1 trong một số trường hợp.
* Greedy 3 sắp xếp các đồ vật theo tỷ lệ giá trị lợi nhuận trên trọng lượng giảm dần đầu tiên, sau đó chọn các đồ vật này cho đến khi không thể chọn thêm nữa. Đây là một phương pháp tối ưu hơn Greedy 1 và Greedy 2 trong nhiều trường hợp.
* Tóm lại, nếu số lượng đồ vật là nhỏ, thì thuật toán vét cạn có thể là lựa chọn tốt nhất để đạt được kết quả tối ưu. Tuy nhiên, trong trường hợp các bài toán có kích thước lớn hơn, các thuật toán Greedy 2 và Greedy 3 có thể được sử dụng để tìm ra kết quả tối ưu một cách nhanh chóng và hiệu quả hơn Greedy 1. Tùy thuộc vào từng trường hợp cụ thể, các thuật toán này có thể cho kết quả khác nhau và cần được lựa chọn một cách thận trọng để đảm bảo đạt được kết quả tối ưu nhất.