**Quy hoạch Động Dựa trên Một Yếu tố**

**Bài toán tìm dãy con dài nhất và Ứng dụng**

1- Bài toán đồng xu

2- Bài toán dãy con đơn điệu dài nhất (Longest Sub Array- LSA)

3- LSA và Bài toán lập lịch sử dụng thiết bị.

4- LSA và Bài toán tìm nhóm tam giác lồng nhau nhiều nhất.

5- LSA và Bài toán xếp chồng đá cao nhất.

**Bài toán tối ưu**: Optimization problem, bài toán đòi hỏi chọn ra lời giải tốt nhất trong tập các lời giải.

**Hàm mục tiêu f**: Objective function, là hàm nhận đầu vào là một đề xuất cho lời giải, đầu ra là một số. Hàm mục tiêu là công cụ đánh giá lời giải của bài toán tối ưu.

**Bài toán tối thiểu hoá hàm mục tiêu: Minimize objective function** problem, là bài toán chọn ra lời giải tốt nhất sao cho hàm mục tiêu cho trị bé nhất. Cơ chế thường dùng như sau:

1. Khởi tạo lời giải S với f(S) = val/ hoặc +∞
2. Lặp
   1. Tìm lới giải mới S2.
   2. Nếu f(S2) < f(S) thì S= S2

**Bài toán tối đa hoá hàm mục tiêu: Maximize objective function** problem, là bài toán chọn ra lời giải tốt nhất sao cho hàm mục tiêu cho trị lớn nhất. Chúng ta có thể quy chiếu bài toán tối đa hóa về bài toàn tối thiểu hóa bằng cách thiết lập lại hàm mục tiêu f’(S) = - f(S), trong đó f(S) là hàm mục tiêu trong bài toán tối đa hóa hàm mục tiêu.

**Các bước lập trình quy hoạch động**

**Bước 1**: Chọn cấu trúc dữ liệu lưu trữ dữ liệu đầu vào, thường là mảng

**Bước 2**: Xây dựng hàm mục tiêu, cơ chế lưu trữ của hàm mục tiêu, cơ chế cập nhật (tối đa hay tối thiểu hóa).

**Bước 3**: Chọn cơ chế xem xét các phần tử trong dữ liệu đầu vào, dùng vòng lặp hay dùng phương pháp đệ quy (nguyên lý chia để trị).

**1- Bài toán đồng xu**

**Tình huống**: Các đồng xu được phát hành với các mệnh giá 1, 2, 5$.

**Bài toán**: Có số tiền S, giả sử 11$, cần đổi thành các đồng xu.

**Yêu cầu tối ưu: Số lượng đồng xu sau khi đổi phải tối thiểu.**

**Ý tưởng giải bài toán dựa trên trực giác- Không dùng quy hoạch động**

Ưu tiên chọn đồng xu có mệnh giá cao nhất chúng ta sẽ có số đồng xu ít nhất

|  |  |  |
| --- | --- | --- |
| Số tiền cần đổi S | Xét đồng xu | Kết quả |
| 11 | 5 | {5} |
| 6 | 5 | {5, 5} |
| 1 | 1 | (5,5,1} |
| Xong |  |  |

**Ý tưởng giải bài toán dùng quy hoạch động**

Không phải lúc nào trực giác cũng mách bảo chúng ta cách giải bài toán tối ưu. Do vậy, việc có được lời giải tối ưu thường là quá trình dò tìm. Với các giả định lời giải ban đầu, trong quá trình giải bài toán tối ưu, nếu phát hiện một lời giải tốt hơn thì cập nhật lời giải.

Với bài toán đồi tiền xu này, với số tiền cần đổi là 11$, chúng ta dủng 12 túi (được gọi là bảng phương án) chứa các xu của quá trình đổi: bags[0] … bags[11]. Trong đó túi bags[0] là túi trống, bags[i] là các đồng xu để đổi cho i$. Như vậy bảng phương án là một mảng một chiều.

Từ túi phía trước, thêm vào một đồng xu chúng ta có một cách đổi mới cho túi phía sau.

Từ bags[0], thêm xu 1, ta có thể cập nhật cho bags[1].

Từ bags[0], thêm xu 2, ta có thể cập nhật cho bags[2].

Từ bags[0], thêm xu 5, ta có thể cập nhật cho bags[5].

Như vậy từ bags[4] = {2, 2}, bags[6] = {}

Thêm đồng xu 2 🡪 {2,2,1} có thể cập nhật cho bags[6] = {2,2,2}

Từ bags[5] = {5}, bags[6] = {2,2,2}

Thêm đồng xu 1 🡪 {5,1} có thể cập nhật cho bags[6] = {2,2,2} 🡪 {5,1}

Ta cũng có thể suy luận ngược:

Bags[11] có thể được cập nhật

Từ bags[10] nếu thêm xu 1,

Từ bags[9] nếu thêm xu 2),

Từ bags[6] nếu thêm xu 5.

Cứ như thế,

Tóm lại 2 chiều suy luận:

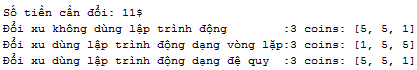
* Cách 1: Dùng vòng lặp. Đi từ túi phía trước (chỉ số nhỏ) để cập nhật các túi phía sau.
* Cách 2: Dùng đệ quy. Đi từ túi phía sau, xác định túi phía trước có thể dùng để cập nhật cho túi phía sau này.

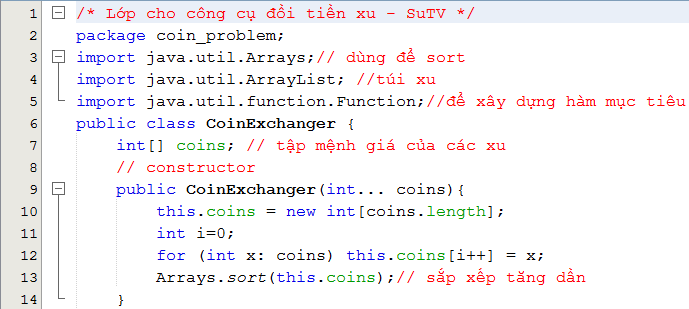
Sau khi cập nhật xong, túi cuối cùng là kết quả việc đổi xu, Bags[11] theo thí dụ.

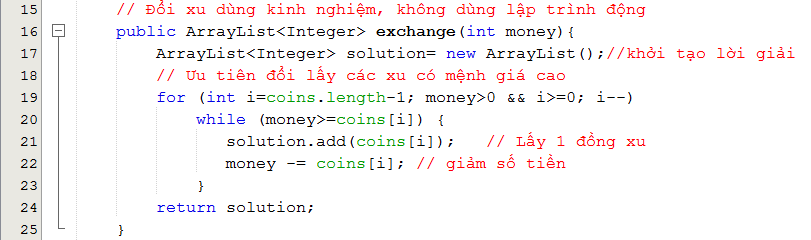
Minh họa dưới đây giới thiệu 3 cách hiện thực đổi xu:

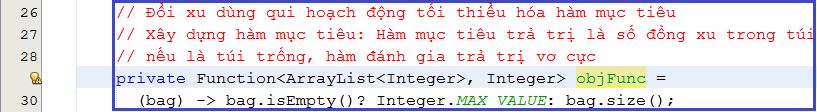
* Đổi xu bằng giải thuật trực giác, không dùng lập trình động.
* Đổi xu dùng lập trình động bằng vòng lặp theo chiều tiến.
* Đổi xu dùng lập trình động bằng đệ quy quay lui.

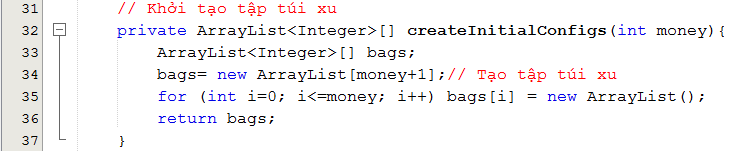
**Kết quả**:

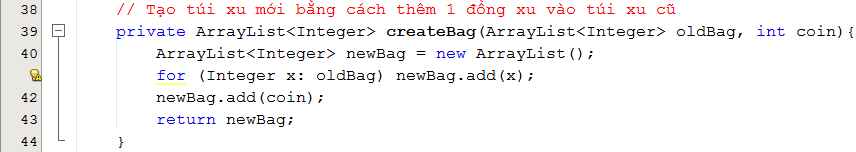


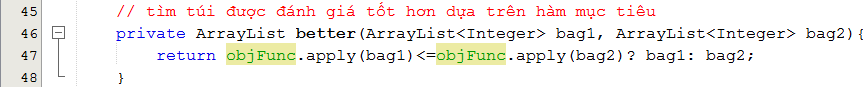
****

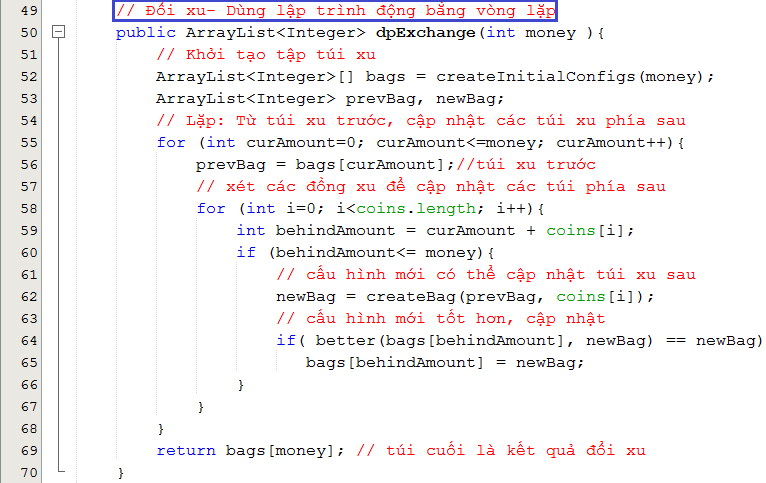
****

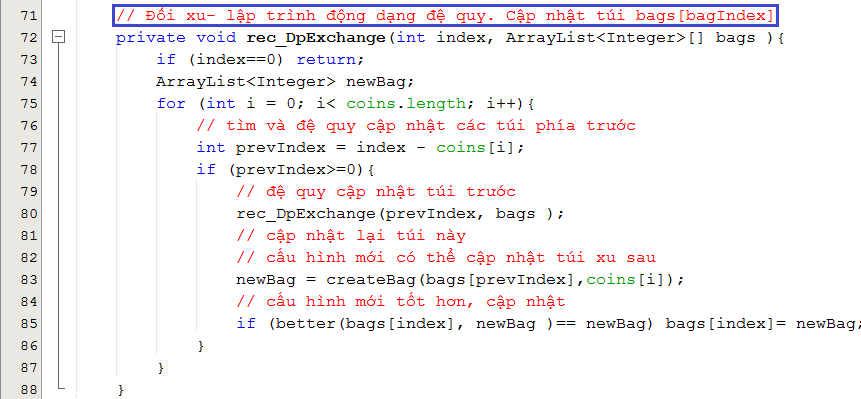
****

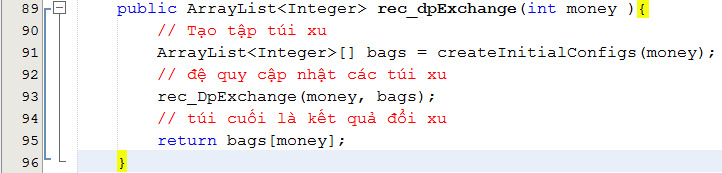
****

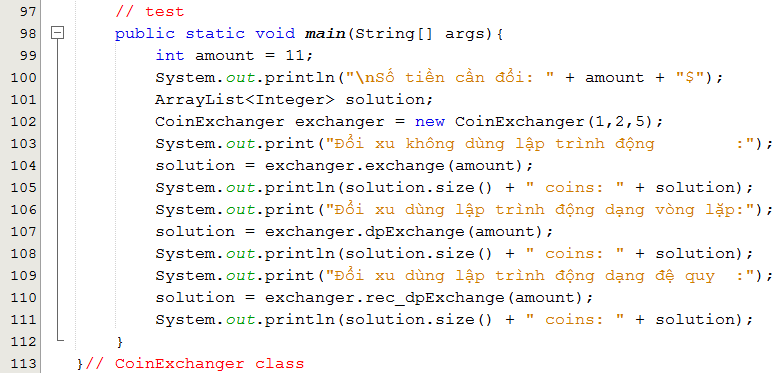
****

****

****

****

****

****

**2- Bài toán dãy con đơn điệu dài nhất- LSA**

**Nhóm**: dãy số hay mảng

**Nhóm con đơn điệu:** Nhóm các phần tử trong nhóm ban đầu có đặc điểm không giảm (ta còn gọi là nhóm con tăng) hoặcnhóm con không tăng (ta còn gọi là nhóm con giảm).

Thí dụ:

Nhóm ban đầu: {1, 5, 7, 2, 5, 6, 7, 8, 9, 0}

Các nhóm con đơn điệu không giảm: {1, 5, 7, 7, 8, 9}, { 1, 5, 5, 6, 7, 8, 9}, {2, 5, 6, 7, 8, 9}, {0}

Nhóm con tăng dài nhất: { 1, 5, 5, 6, 7, 8, 9}, 7 phần tử.

**Phân tích:**

Gọi ***a*** là mảng

Với mỗi phần tử trong nhóm, chúng ta cần thêm các thông tin đi cùng với phần tử tại vị trí ***i*** này:

* Nhóm dài nhất tính đến vị trí ***i*** có độ dài là bao nhiêu 🡪 Cần thêm mảng F[]. Nhờ mảng F[], chúng ta biết được mảng con dài nhất kết thúc tại vị trí nào. F[] chính là bảng hàm mục tiêu.
* Trước phần tử này là phần tử ở vị trí nào 🡪 Mảng prev[]. Nhờ mảng prev[], chúng ta có thể truy ngược để có được mảng con dài nhất.

Sau quá trình tính toán sử dụng phương pháp tối đa hóa hàm mục tiêu nghĩa là khi phát hiện chuỗi con dài hơn khi xét đến vị trí thứ i của mảng a thì chúng ta cập nhật F[] và prev[]. Sau khi đã xét đếnm vị trí cuối (i=9), kết quả là:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |
| **F[]** | **1** | **2** | **3** | **2** | **3** | **4** | **5** | **6** | **7** | **1** |
| prev[] | **-1** | **0** | **1** | **0** | **1** | **4** | **5** | **6** | **7** | **-1** |

Từ F[], chỉ số có trị lớn nhất là 8, L[8[=7. Độ dài nhóm con dài nhất: 7.

Cách tìm ra nhóm con tăng dài nhất:

Index = chỉ số có trị lớn nhất trong F[] = 8

|  |  |
| --- | --- |
| Chỉ số/ index | Chuỗi con kết quả |
| 8 | Lấy a[8] 🡪 {9} |
| Prev[8]= 7 | Lấy a[7] 🡪 {8, 9} |
| Prev[7] = 6 | Lấy a[6] 🡪 {7,8,9} |
| Prev[6] = 5 | Lấy a[5] 🡪 {6,7,8,9} |
| Prev[5]= 4 | Lấy a[4] 🡪 {5,6,7,8,9} |
| Prev[4]= 1 | Lấy a[1] 🡪 {5,5,6,7,8,9} |
| Prev[1]= 0 | Lấy a[0] 🡪 {1,5,5,6,7,8,9} |
| Prev[0]= -1. Ngưng | Kết quả: {1,5,5,6,7,8,9} |

**Cách tính toán F[] và prev[]:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** | **Khởi tạo** |
| **F[]** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Nhóm con 1 phần tử: 1 |
| prev[] | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | không có phần tử đứng trước: -1 |
| Xét i=1 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Xét các vị trí j trước I, cập nhật F[] và prev[] |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | **2** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Đến chỉ số 1, F[1]= 2 |
| prev[] | -1 | **0** | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | Phần tử trước của a[1] là a[0] |
| Xét i=2 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | **3** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Đến i=2, chuỗi con dài nhất đến a[2] là 3 |
| prev[] | -1 | 0 | **1** | -1 | -1 | -1 | -1 | -1 | -1 | -1 | trước a[2] là a[1] |
| Xét i=3 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | **2** | 1 | 1 | 1 | 1 | 1 | 1 | Đến i=3, chuỗi con dài nhất đến a[3] là 2 |
| prev[] | -1 | 0 | 2 | **0** | -1 | -1 | -1 | -1 | -1 | -1 | trước a[3] là a[0] |
| Xét i=4 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | 2 | **3** | 1 | 1 | 1 | 1 | 1 |  |
| prev[] | -1 | 0 | 2 | 0 | **1** | -1 | -1 | -1 | -1 | -1 |  |
| Xét i=5 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | 2 | 3 | **4** | 1 | 1 | 1 | 1 |  |
| prev[] | -1 | 0 | 2 | 0 | 1 | **4** | -1 | -1 | -1 | -1 |  |
| Xét i=6 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | 2 | 3 | 4 | **5** | 1 | 1 | 1 |  |
| prev[] | -1 | 0 | 2 | 0 | 1 | 4 | **5** | -1 | -1 | -1 |  |
| Xét i=7 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | 2 | 3 | 4 | 5 | **6** | 1 | 1 |  |
| prev[] | -1 | 0 | 2 | 0 | 1 | 4 | 5 | **6** | -1 | -1 |  |
| Xét i=8 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | 1 | 2 | 3 | 2 | 3 | 4 | 5 | 6 | **7** | 1 |  |
| prev[] | -1 | 0 | 2 | 0 | 1 | 4 | 5 | 6 | **7** | -1 |  |
| Xét i=9 |  |  |  |  |  |  |  |  |  |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| a[] | **1** | **5** | **7** | **2** | **5** | **6** | **7** | **8** | **9** | **0** |  |
| **F[]** | **1** | **2** | **3** | **2** | **3** | **4** | **5** | **6** | **7** | **1** |  |
| prev[] | **-1** | **0** | **1** | **0** | **1** | **4** | **5** | **6** | **7** | **-1** |  |

**Thuật toán:**

For (i=0;i<n; i++) { // khởi tạo 2 mảng F[] và prev[]

F[i]= 1;

prev[i]=-1;

}

for (int i=0; i<n; i++) { // cập nhật tăng F[i] nếu được

for (int j=0; j<i; j++)// xét các phần tử j đứng trước i

if (a[i] >= a[j])

if ((F[i]<F[j]+1) {

F[i]=F[j]+1; // chuỗi con dài ra 1 phần tử đến i

prev[i] = j; // trước i là j

}

}

**Đánh giá:**

* **Độ phức tạp thời gian: Hai vòng lặp🡪 O(n2)**
* **Độ phức tạp không gian: 3 mảng n phần tử: O(n)**

**Áp dụng bài toán dãy con dài nhất:**

Ngoài dữ liệu đầu vào là mảng số hay chuỗi ký tự. Cơ chế giải bài toán tìm dãy con còn được áp dụng cho việc tối ưu hóa việc sử dụng 1 thiết bị với n đăng ký.

**Bài toán lập lịch họp/ lịch thuê thiết bị**: Có 1 phòng họp/ thiết bị với n đăng ký. Lập lịch họp/ lịch sử dụng máy tối ưu sao cho tổng thời gian sử dụng phòng họp nhiều nhất/ doanh thu cao nhất.

**Hiện thực:**

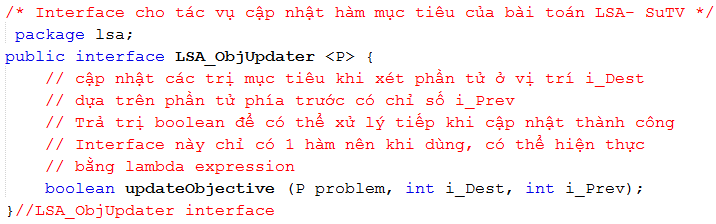
Khi hiện thực giải bài toán thuộc nhóm này, mỗi bài toán khác nhau có cách cập nhật hàm mục tiêu khác nhau. Trong bài toán tìm dãy con: Khi xét phần tử thứ i, chúng ta cập nhật F[i] bằng cách cộng thêm 1 như đã được trình bày trong thuật toán ở trên hoặc cộng thêm một trị khác 1 tùy thuộc bài toán chẳng hạn như trong bài toán lập lịch sử dụng thiết bị, mảng F[] chứa giá trị thời gian sử dụng nên F[i] được cộng thêm một giá trị là thời gian sử dụng của đăng ký thứ i. Như vậy, việc cập nhật hàm mục tiêu F[], phụ thuộc vào mảng dữ liệu đầu vào a[].

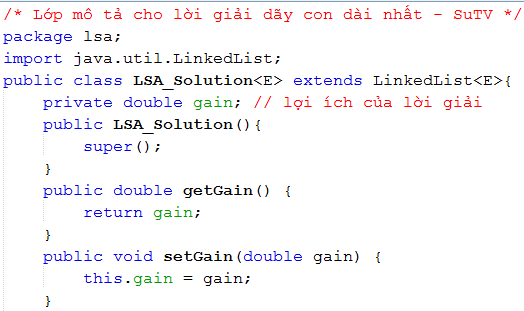
Để có thể tổng quát hóa lời giải cho nhóm bài toán này, hàm cập nhật mảng mục tiêu F[] được tách ra bằng interface ObjectiveUpdater. Khi giải bài toán cụ thể nào, chúng ta sẽ tự định nghĩa phù hợp.

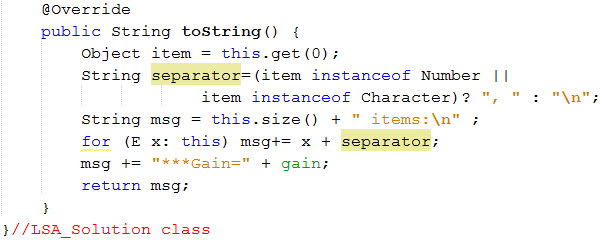
Phần hiện thực sau đây mô hình hóa cơ chế giải bài toán tối ưu này.

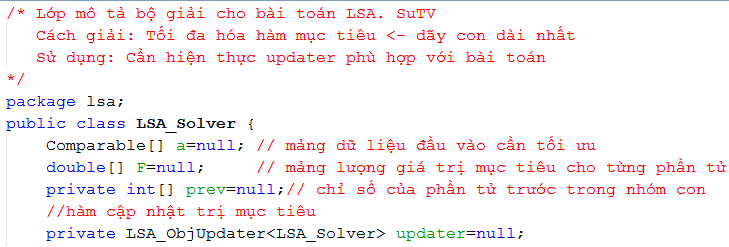
Cấu trúc các lớp:

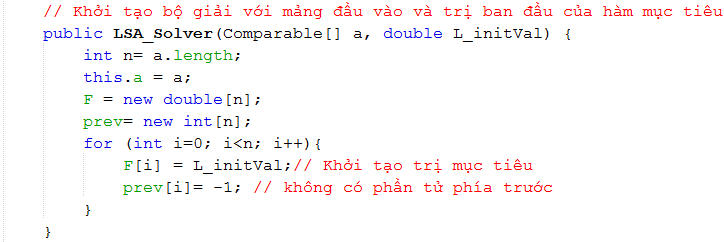
|  |  |
| --- | --- |
|  | Nhóm công cụ hình thành mô hình giải bài toán dãy con dài nhất:   * LSAObjUpdater: Interface cho việc cập nhật hàm mục tiêu. * LSA\_Solution: Lớp mô tả lời giải của bài toán * LSA\_Solver: Lớp mô tả công cụ giải bài toán   Nhóm các thí dụ sử dụng mô hình:   * LSA\_Use1\_IntCharArrayProblem: Lớp mô tả một bài mẫu tìm dãy con dài nhất cho mảng số và chuỗi. * TimeRegistration: Lớp mô tả một thông tin đăng ký sử dụng thiết bị * LSA\_Use2\_RegistrationScheduler: Lớp mô tả một bài mẫu lập lịch tối ưu sử dụng thiết bị |

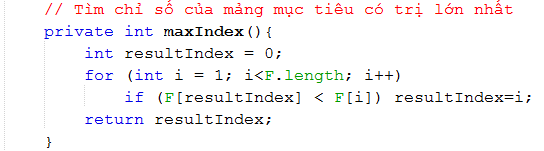
****

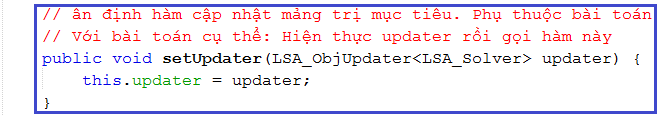
****

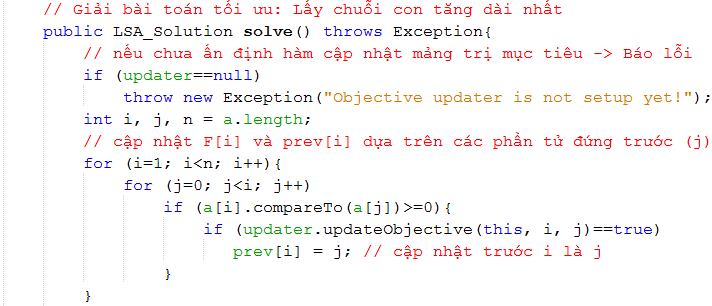
****

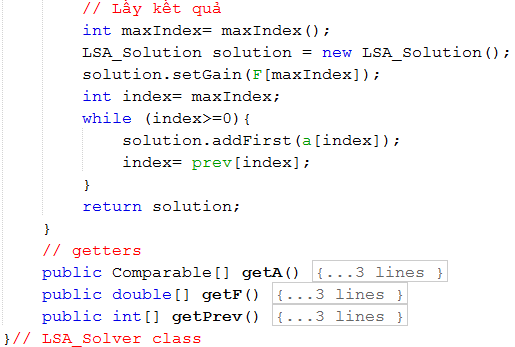
****

****

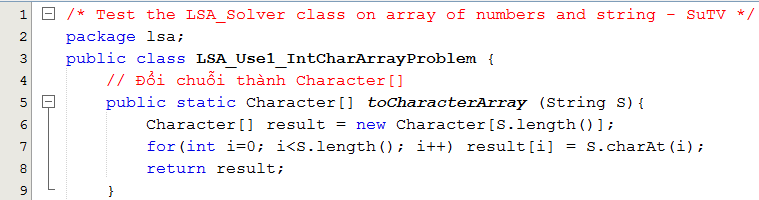
****

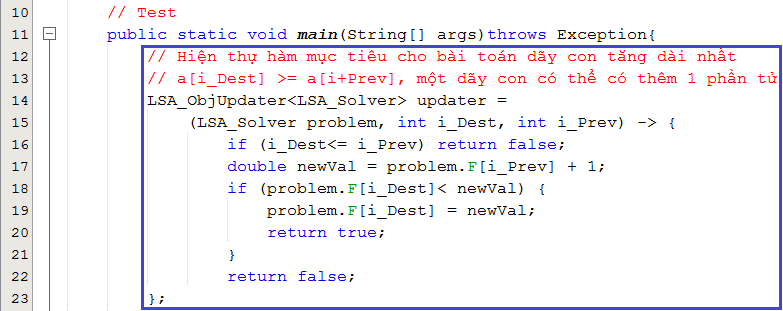
****

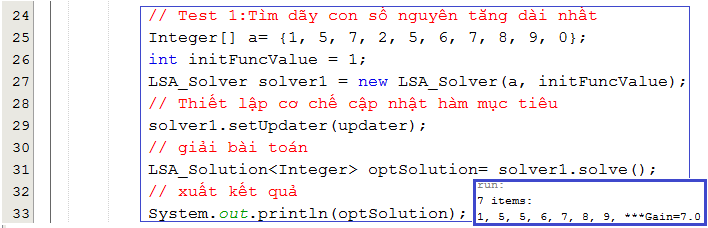
****

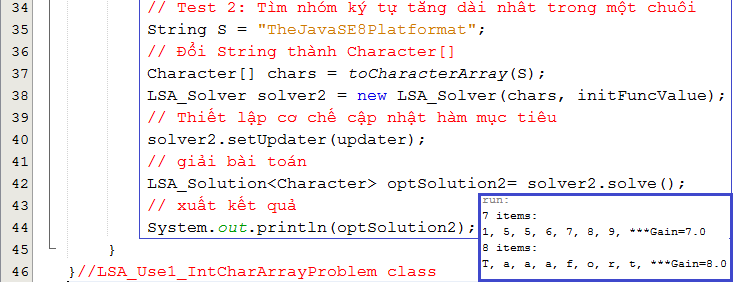
****

**Minh họa tìm dãy con dài nhất trên mảng số và chuỗi ký tự.**

****

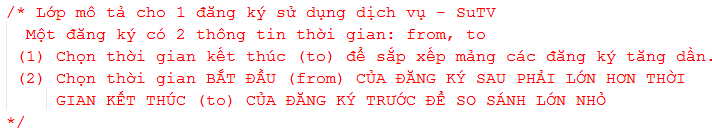
****

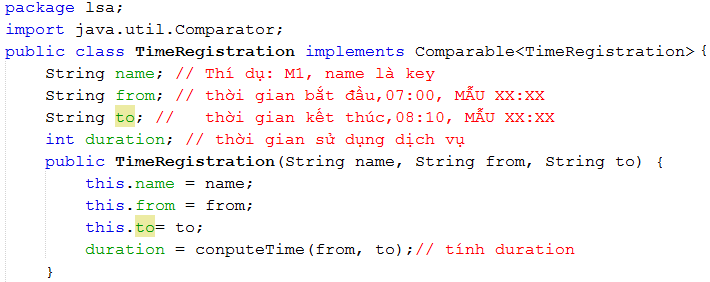
****

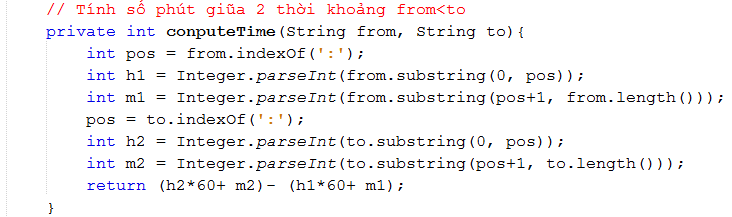
****

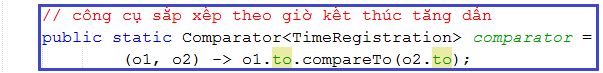
**3- LSA và Bài toán Lập lịch sử dụng thiết bị**

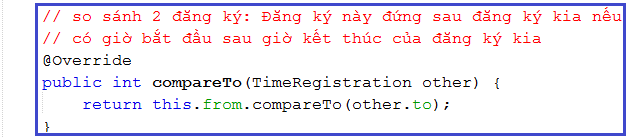
**Minh họa lập lịch tối ưu sử dụng thiết bị nhiều nhất.**

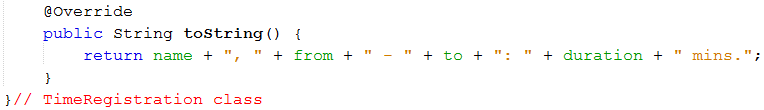
****

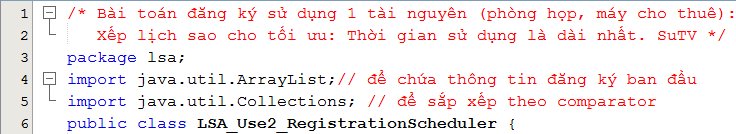
****

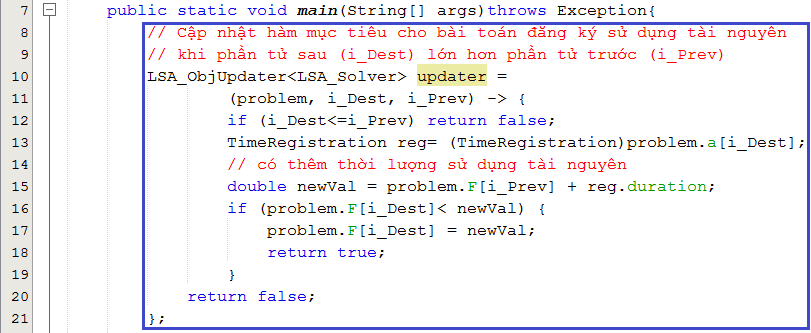
****

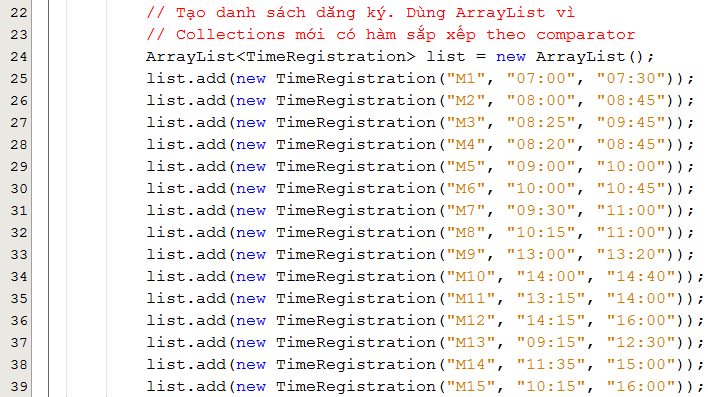
****

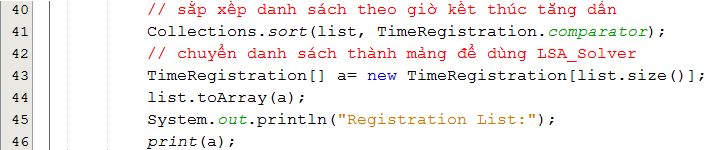
****

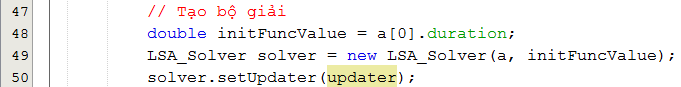
****

****

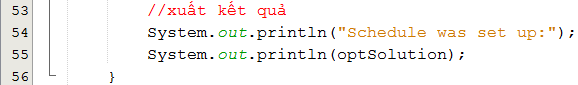
****

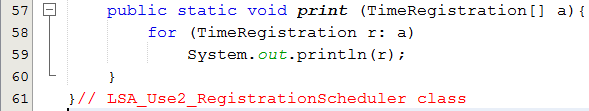
****

****

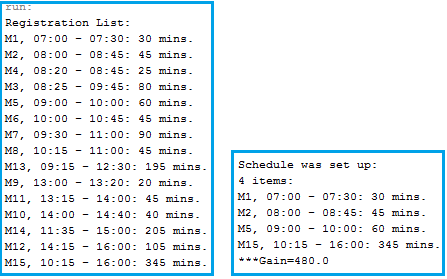
****

****

****

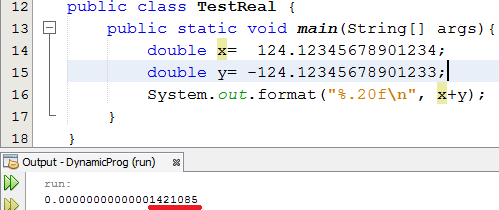
****

**Kết quả giải bài toán**

****

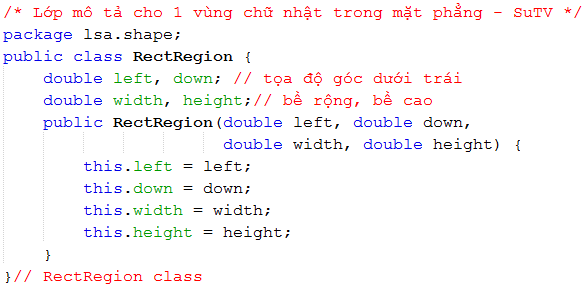
**4- LSA và Bài toán tìm nhóm tam giác lồng nhau nhiều nhất.**

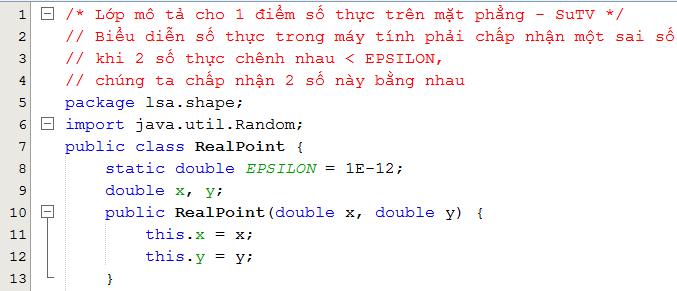
Một giới hạn khi biểu diễn số thực trong máy tính đó là chúng ta phải chấp nhận một sai số vì số thực được lưu trữ thành hai phần: phần số mũ (exponent) và phần định trị (mantissa). Như vậy, với phép so sánh 2 số thực, chúng ta nên tính chênh lệch tuyệt đối của hai số bằng phép trừ. Nếu sai số này nhỏ hơn EPSILON tự định nghĩa thì xem như hai số thực này bằng nhau.

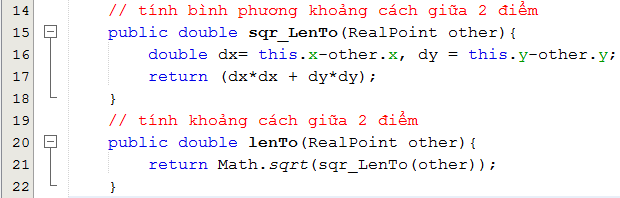


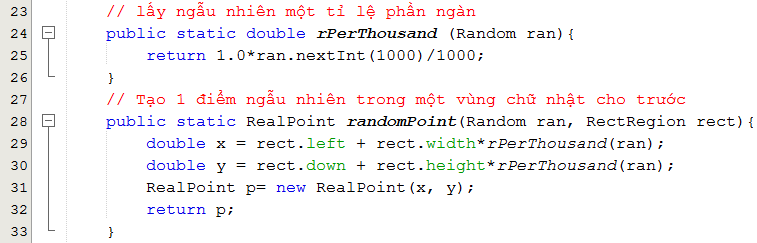
**Cấu trúc các lớp:**

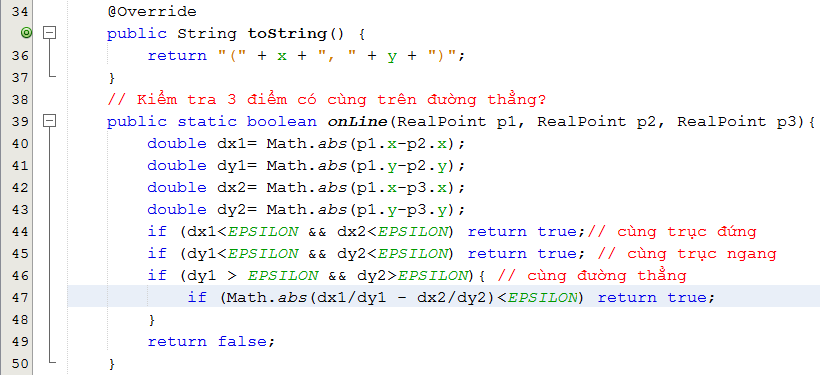
|  |  |
| --- | --- |
|  | **RectRegion**: Lớp mô tả 1 vùng chữ nhật trong hệ tọa độ số thực  **RealPoint**: Lớp mô tả một điểm  **Triangle**: Lớp mô tả một tam giác  **TriangleList**: Lớp mô tả một danh sách các tam giác  **Triangle\_LongestNestedProblem**: Lớp minh họa giải bài toán nhóm tam giác lồng nhau nhiều nhất |

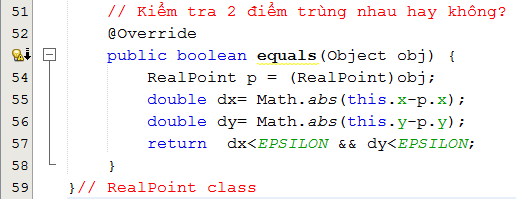






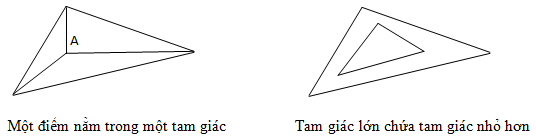




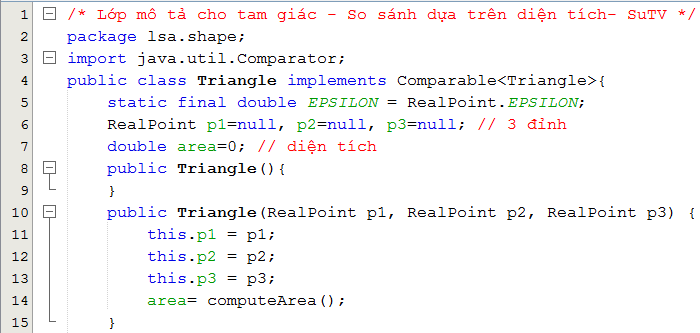


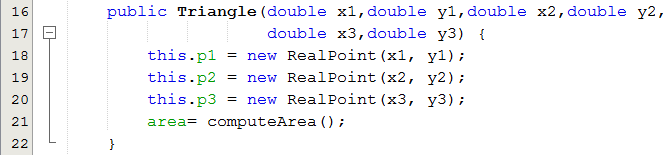
Một số đặc điểm của bài toán tìm nhóm các tam giác lồng nhau nhiều nhất:

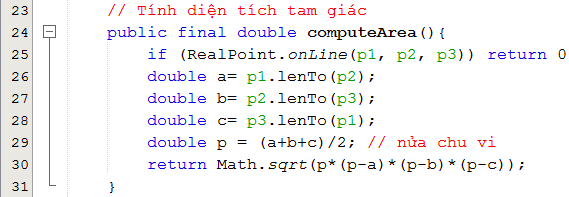
* Một tam giác có diện tích lớn thì có khả năng có những tam giác phía trong. Như vậy, các tam giác nên được sắp xếp theo diện tích tăng dần.

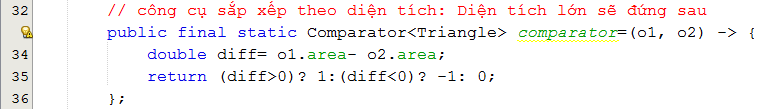


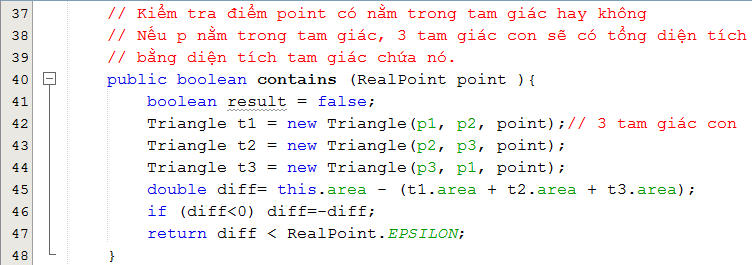
* Tam giác T1 chứa tam giác T2 nghĩa là cả 3 đỉnh của tam giác T2 đều nằm trong tam giác T1.
* Để xét 1 điểm A có nằm trong tam giác T hay không chúng ta dựa vào nguyên tắc: Nếu A nằm trong T thì A chia T thành 3 tam giác con. Tổng 3 tam giác con này bằng diện tích của T.
* Như vậy, để giải bài toán này, chúng ta cần
  + Một cơ chế giúp sắp xếp các tam giác đã có theo diện tích tăng dần.
  + Một cơ chế xác định tam giác phía sau T2 được gọi là lớn hơn tam giác phía trước T1 chỉ khi T2 chứa T1.
  + Đến đây, chúng ta đã có thể dùng mô hình LSA để giải bài toán này.

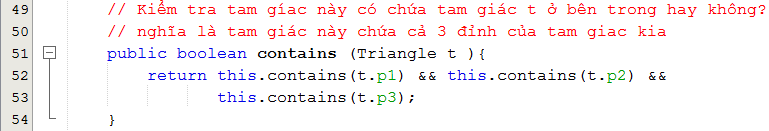


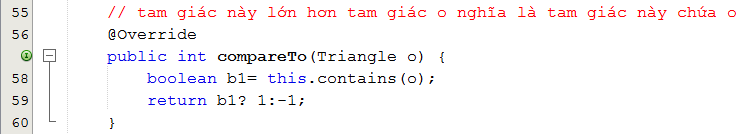


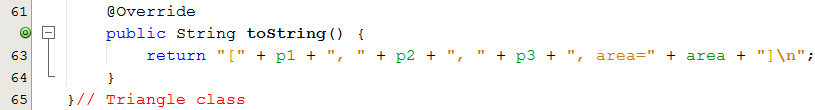


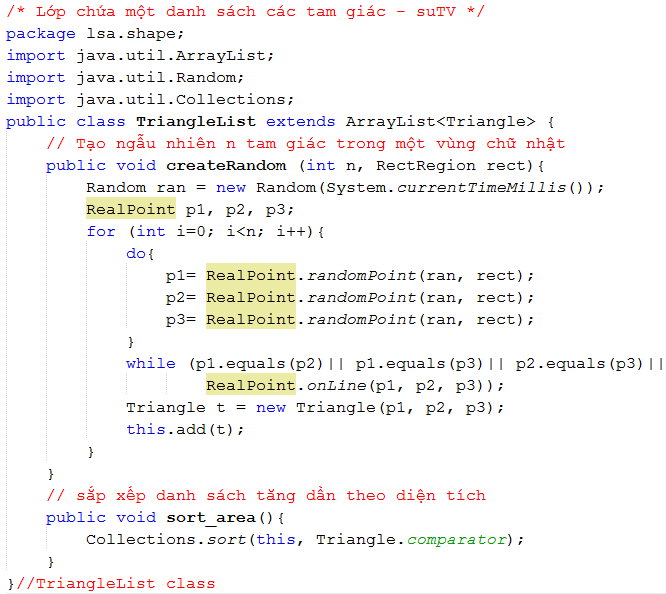


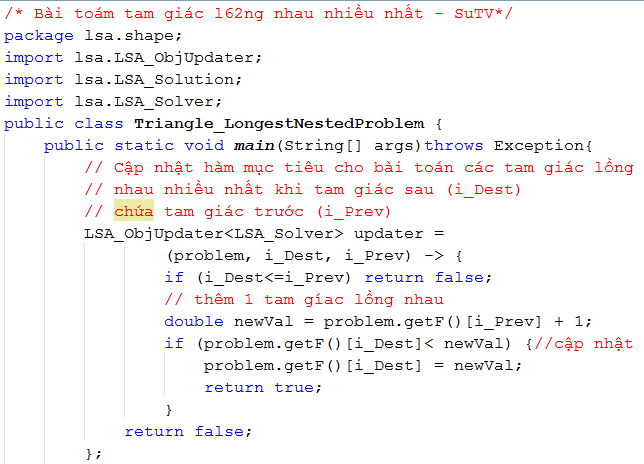


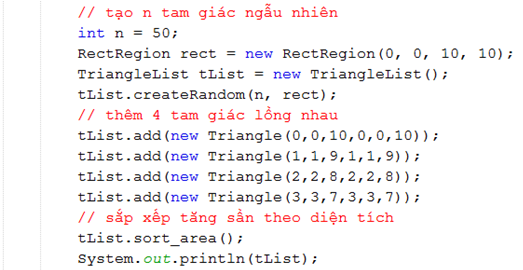


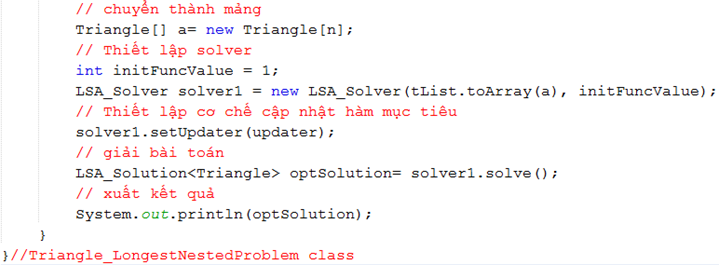




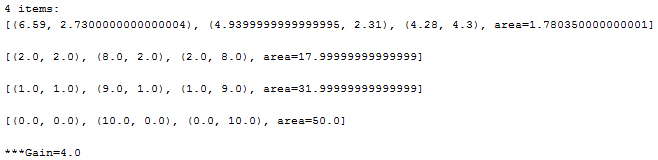








**Kết quả**



**5- LSA và Bài toán Xếp chồng đá cao nhất**

|  |  |
| --- | --- |
| Cho trước 1 số khối đá chữ nhật. Hỏi khi xếp khối này lên khối khác sao cho khối trên không vượt ra khỏi mép của khối dưới. Hỏi số khối đá nhiều nhất có thể chồng lên nhau là bao nhiêu? |  |

Gợi ý để giải:

* Mô tả khối đá: chiều dài (L), chiều rộng (W), chiều cao (H). H được chọn mang trị nhỏ nhất so với hai chiều kia để khi xếp thành chồng, số khối đá chồng lên nhau là nhiều nhất. Nên sắp thứ tự: L>= W >=H
* Khối đá dưới, B1,có diện tích đáy (S=WxL) lớn hơn khối trên, B2, vì khối trên không được vướt ra khỏi mép của khối dưới nghĩa là: L1>=L2 && W1>=W2 (1).
* Sau khi đã có danh sách các khối đá, cách giải:
  + Sắp xếp các khối đá tăng dần theo diện tích đáy S, nếu cùng diện tích đáy thì theo chiều cao tăng dần.
  + Khối sau lớn hơn khối trước (B1.compareTo(B2)) được định nghĩa theo (1).
  + Dùng mô hình LSA để giải tương tự bài toán tam giác lồng nhau dài nhất.