**Quy hoạch Động:**

**Giải Bài toán bằng Hồi quy (Backtracking)**

1- Giới thiệu

2- Một số định nghĩa liên quan đến bài toán tổ hợp.

3- Cách giải vét cạn dùng hồi quy

4- Hiện thực giải thuật hồi quy và áp dụng

5- Cách giải vét cạn với cơ chế sinh dữ liệu

6- Hiện thực bộ sinh dữ liệu và áp dụng.

**1- Giới thiệu**

Rất nhiều bài toán có dạng: Cho trước một tập hợp S, lấy ra một tập con sao cho thỏa mãn một điều kiện nào đó. Một số thí dụ:

Thí dụ 1: Cho một mảng số nguyên a[]. Hãy lấy ra các mảng con có tổng là sum.

Thí dụ 2: Bài toán cái túi cơ bản: Cho một tập món đồ, mỗi món đồ :

* Chỉ có duy nhất một bản (thí dụ bút chì chỉ có 1 cái).
* Có khối lượng riêng w[i]
* Có giá trị riêng v[i],

Hãy chọn các món đồ để cho vào một túi sao cho tổng khối lượng <=W và tổng giá trị lớn nhất.

Thí dụ 3, Bài toán cái túi cho phép lấy nhiều lần: Cho một tập món đồ, mỗi món đồ:

* Có thể có nhiều bản (thí dụ bút chì có 5 cái).
* Có khối lượng riêng w[i]
* Có giá trị riêng v[i],

Khi chọn món đồ có thể không lấy , cũng có thể lấy 1, 2,… vật này.

Hãy chọn các món đồ để cho vào một túi sao cho tổng khối lượng <=W và tổng giá trị lớn nhất.

Thí dụ 3, Một cửa hàng điện máy đang bán TV, tử lạnh, … Một khách hàng muốn được tư vấn để mua 3 TV, 2 tủ lạnh, 2 quạt máy sao cho tổng chi phí đầu tư tối đa là 30 triệu đồng.

Dạng bài toán này thuộc nhóm bài toán tổ hợp (combination) và chúng ta có thể giải nhóm bài toán này bằng kỹ thuật hồi quy giúp xem xét hết mọi khả năng (vét cạn, exhausted searching) có thể có lời giải.

**2- Một số định nghĩa**

* **Tập biến, vars[]:** Tập các biến (variable) cần được gán trị. Việc giải một bài toán là đi gán trị cho các biến.
* **Tập miền trị D[], domains:** Miền trị là tập hợp các giá trị có thể gán cho một biến. Mỗi miền trị D[i] mô tả khả năng xuất hiện giá trị của biến vars[i] trong lời giải.
* **Ràng buộc (constraint) của bài toán**: Là điểu kiện phải thỏa mãn của lời giải.
* **Đề xuất(Proposal):** Một bộ trị (tuple) có thể được gán vào các biến với hy vọng đó là một lời giải. Nếu một đề xuất thỏa mãn các ràng buộc của bài toán thì được gọi là một lời giải (solution).
* **Không gian của bài toán:** Tập hợp các đề xuất có thể có. Nếu chúng ta có cách duyệt xét (vét cạn) từng phần tử trong không gian của bài toán (proposal) thì nếu bài toán có lời giải, chắc chắn lời giải sẽ bị phát hiện.
* **Tập lời giải (solutions):** Dựa trên ràng buộc của bài toán, bài toán có thể không có lời giải mà cũng có thể có một tập lời giải. Việc tìm ra tất cả các lời giải của bài toán có thể tiêu tốn một thời gian khá dài hoặc rất dài (xem phần đánh giá độ phức tạp của thuật toán ở dướI). Trong lĩnh vực trí tuệ nhân tạo (AI), người ta thường chỉ muốn tìm ra một lời giải ban đầu rồi thông qua các giải thuật, người ta có thể phát hiện các lời giải tốt hơn. Việc này không được bàn đến ở đây.

**Độ phức tạp tính toán (thời gian) của bài toán tổ hợp:**

Gọi N là số biến của bài toán,

Gọi M là lượng số trung bình của mỗi miền trị.

Chúng ta vét cạn mọi khả năng có thể có của lời giải nên theo nguyên lý nhân, độ phức tạp thời gian của giải thuật sẽ là MN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vars[0] | Vars[1] | … | … | Vars[N-1] |
| M lựa chọn | M lựa chọn | M lựa chọn | … M lựa chọn | M lựa chọn |

*Hình giải thích độ phức tạp tính toán của giải thuật vét cạn:* O(MN)

**Một số cách tinh chỉnh để sớm tìm ra lời giải đầu tiên và cải thiện độ phức tạp tính toán:**

Bài toán phải xem xét N biến, mỗi biến có thể mang một trong M trị. Những yếu tố ảnh hưởng đến thời gian phát hiện lời giải đầu tiên đó là: thứ tự các biến và thứ tự các trị của mỗi biến được xem xét. Như vậy, một số cải tiến được tiến hành trước khi giải bài toán đó là:

1. Sắp xếp lại thứ tự của các biến,
2. Chọn chiều tăng/giảm của miền trị phù hợp,

Nếu trị của biến sau lại có những ràng buộc với trị của biến trước, việc thu giảm miền trị của biến phía sau sau khi đã gán trị cho các biến phía trước cũng làm giảm thời gian giải bài toán.

**3- Cách giải vét cạn dùng hồi quy**

Hồi quy, backtracking, là một kỹ thuật sau khi gán trị cho biến vars[i] thì đi xem xét gán trị cho biến phía sau vars[i+1] nếu có thể được. Trong tình huống bị bí thì quay lui lại phần tử trước đó.

* Hồi quy một bước (one-step): Khi không thể gán trị cho biến xi thì tìm trị khác để gán cho biến xi-1
* Hồi quy một k bước (k-step): Khi không thể gán trị cho biến xi thì tìm trị khác để gán cho biến xi-k

***Hồi quy một bước là dạng hồi quy thông dụng nhất vì dễ hiểu, dễ áp dụng.***

**Khác biệt giữa đệ quy (recursion) và hồi quy (backtracking):**

|  |  |
| --- | --- |
| ***Đệ quy*** | ***Hồi quy*** |
| Bắt đầu  Xử lý bước thứ n // kết quả  Xử lý bước n-1  …  Xử lý bước 2  Xử lý bước 1// Bước chặn đệ quy  Xử lý bước 2 tiếp tục  …  Xử lý bước n-1 tiếp tục  Xử lý bước n tiếp tục  Kết thúc   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **1** | **2** | **…** | **…** | **n-1** | **n** | | Bắt đầu  Xử lý bước 1 // gán trị cho biến thứ 1, biến đầu tiên Xử lý bước 2 // gán trị cho biến thứ 2  …….  Xử lý bước i // gán trị cho biến thứ i  Xử lý bước i+1 // gán trị cho biến thứ i+1  Nếu không thể xử lý thành công ở bước i+1 thì quay lại “Xử lý bước i”  ….  Xử lý bước n;  Kết thúc   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **1** | **…** | **i** | **i+1** |  | **n** | |

*Khác biệt giữa đệ quy và hồi quy*

Giải thuật hồi quy một bước như sau:

***Giải thuật xem xét biến thứ i của bài toán***

**Procedure backtrack (i, vars[], D[], solutions)**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | Bắt đầu  Với mỗi trị x trong D[i] {  vars[i] = x;  Nếu (i==n-1) { // gán xong các biến tức là có một đề xuất mới  Nếu (vars[] thỏa mãn điều kiện của bài toán) thì {  Tạo một solution từ vars[];  Thêm solution vào solutions ; // có thêm 1 lời giải  }  }  Ngược lại backtrack (i+1, vars[], D[], solutions); // xem xét biến kế tiếp  }  Kết thúc |

*Giải thuật hồi quy một bước*

**Chú ý**

* **Hồi quy 1 bước** thể hiện ở chỗ nếu như không thể gán được trị cho biến thứ i+1 (dòng 9) thì quay về gán trị kế tiếp cho biến thứ i (dòng 1 và dòng 2)
* Tập trị đã gán trị đủ các biến hình thành một đề xuất (nhóm trị trong vars[] là một đề xuất)

Giải thuật giải bài toán dùng hồi quy

**Procedure bkSolve (vars[], D[])**

Khởi tạo tập Solutions là tập trống;

Backtrack(0, vars[], D[], solutions); // xem xét từ biến đầu tiên

**Solutions là kết quả;**

End

*Giải bài toán bằng hồi quy*

**Độ phức tạp bộ nhớ của giải thuật hồi quy:**

Tương tự như việc quản lý bộ nhớ của hàm đệ quy, mô hình quản lý bộ nhớ khi chạy giải thuật hồi quy trên bài toán N biến, mỗi miền trị có trung bình M trị là một cây M nhánh. Tại một thời điểm, có một đường đi của các hàm gọi nhau hiện hành trong bộ nhớ với chiều cao là số biến N. Như vậy, độ phức tạp về bộ nhớ của giải thuật là **O(N)**. Với bài toán với số biến N nhỏ (vài chục biến), O(N) không là vấn đề nhưng với N rất rất lớn, bộ nhớ hữu hạn của máy tính có thể không đủ chứa thông tin về các hàm/hành vi hồi quy gọi nhau (tương tự như các hàm đệ quy gọi nhau).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vars[0] | Vars[1] | … | … | Vars[N-1] |
| M lựa chọn | M lựa chọn | M lựa chọn | … M lựa chọn | M lựa chọn |

*Độ phức tạp không gian của giải thuật hồi quy:* O(N)

***Tóm lại:***

***Độ phức tạp về thời gian của giải thuật hồi quy:* O(MN)**

***Độ phức tạp về bộ nhớ của giải thuật hồi quy:* O(N)**

**4- Hiện thực giải thuật Hồi quy và Áp dụng**

**4.1- Thư viện cho cơ chế giải bài toán bằng hồi quy**

|  |  |
| --- | --- |
|  | **Domain**: Lớp mô tả cho miền trị của biến  **Variable**: Lớp mô tả cho một biến  **Proposal**: Lớp mô tả cho một đề xuất  **BK\_Problem**: Lớp trừu tượng mô tả cho một bài toán. Lớp này là lớp trừu tưo75ng vì chỉ có thể hiện thực các ràng buộc tại một bài toán cụ thể |

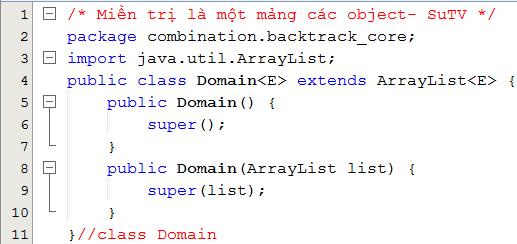
Domain là tập trị sẽ gán cho một biến. Tùy bài toán mà Domain sẽ là tập trị nào.

Bài toán chuỗi bit: Domain là {0,1}

Bài toán tìm mảng con có tổng là sum, domain sẽ là {null, a[i]}. Trong đó null nghĩa là mảng con không có phần tử này.

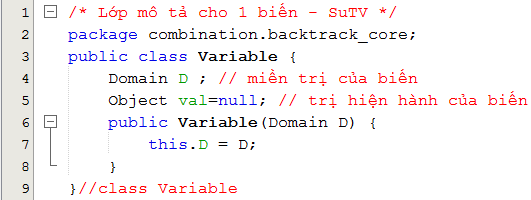
Bài toán cái túi (xem bên dưới để biết về bài toán cái túi): domain mang trị {0,1, 2,…, số vật cùng loại}

Để tổng quát, domain là một tập các object.

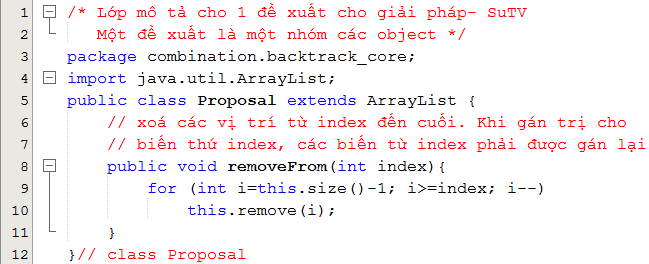


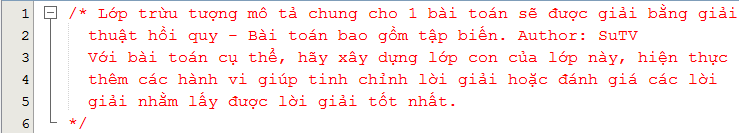
Tùy bài toán mà biến mang trị nào, đa phần là 1 con số, có thể là một object.

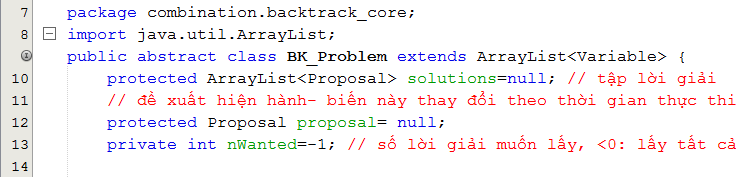
Tổng quát hóa: Trị của biến được gán hiện hành là một object.

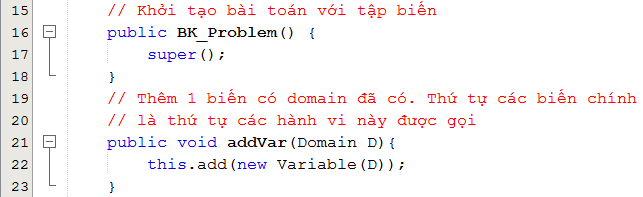


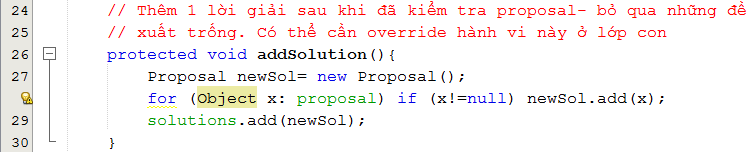
Một tập trị gán cho biến đa phần là một con số (bit 0/1 hoặc hoặc số lượng/ hoặc chỉ số của đồi tượng thực). Như vậy, ta cần một lớp mô tả cho nhóm đối tượng thực giúp mô tả một đề xuất thực.

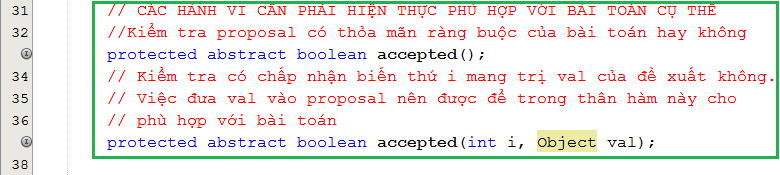


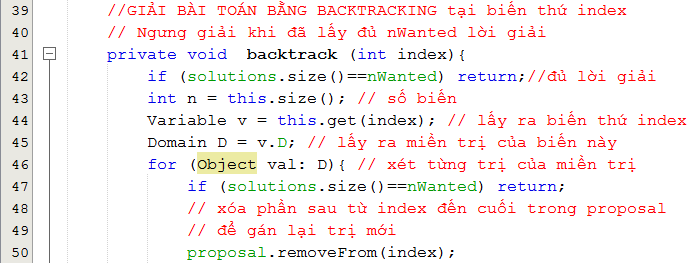


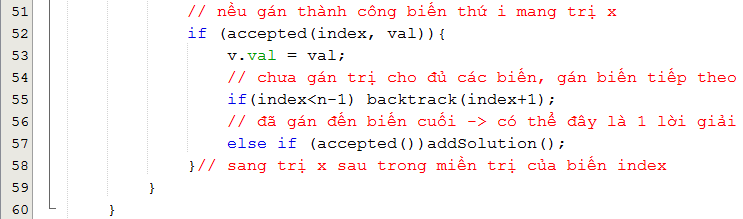


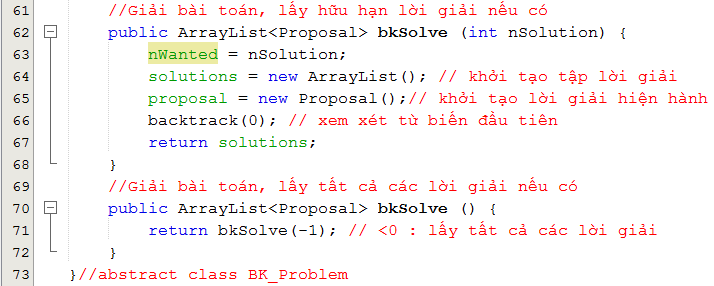










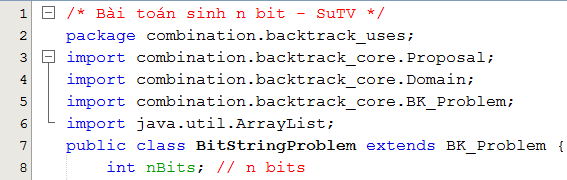


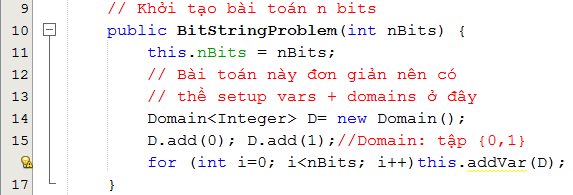
Sau đây là một số bài toán được giải bằng công cụ hồi quy đã xây dựng nêu trên:

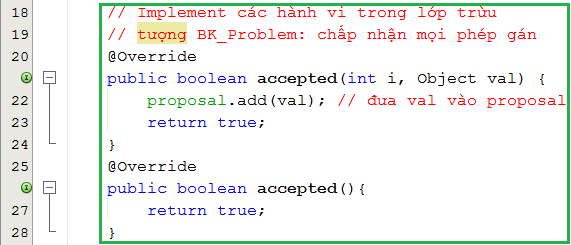
|  |  |
| --- | --- |
|  | Bài toán chuỗi bit  Bài toán mảng con có tổng cho trước  Bài toán cái túi  Bài toán quân hậu  Bài toán tư vấn mua hàng |

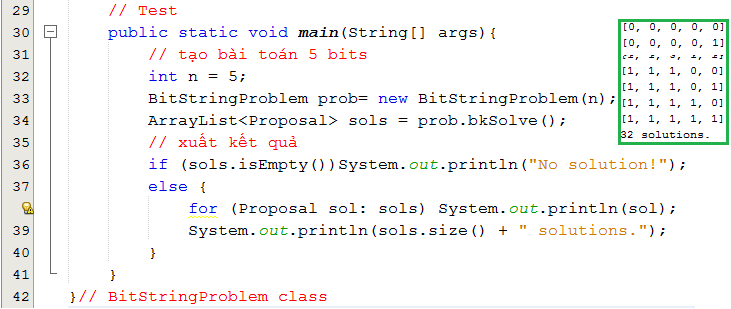
**4.2- Bài toán chuỗi bits**

**Bài toán**:Xuất toàn bộ các chuỗi n bit ra màn hình

****

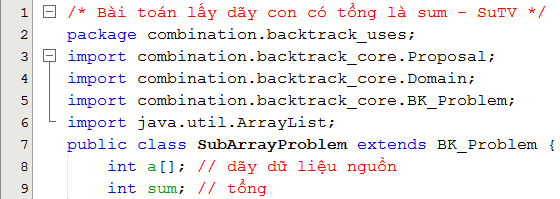
****

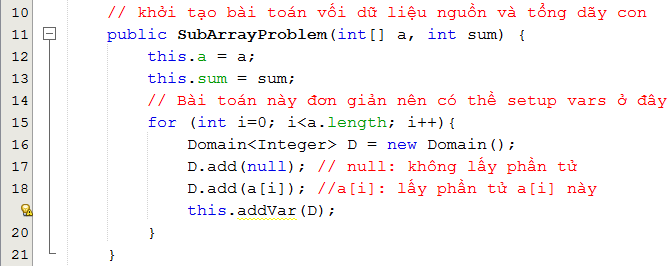
****

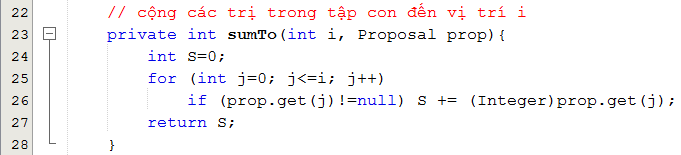
****

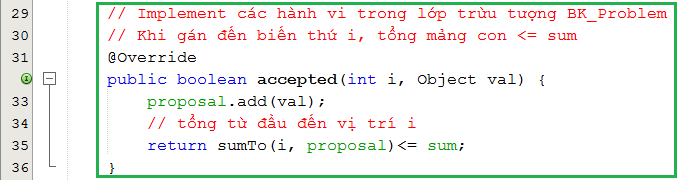
**4.3- Bài toán mảng con có tổng cho trước**

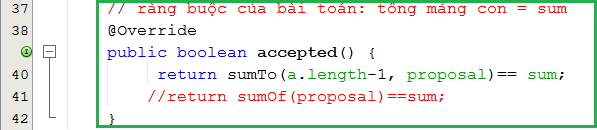
**Bài toán**:Cho trước một mảng số nguyên và số nguyên sum. Hãy xuất ra các mảng con có tổng là sum.

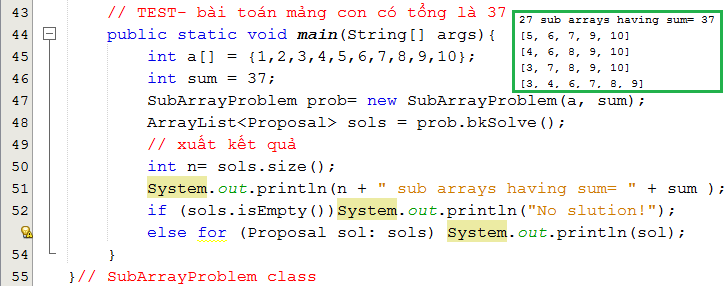












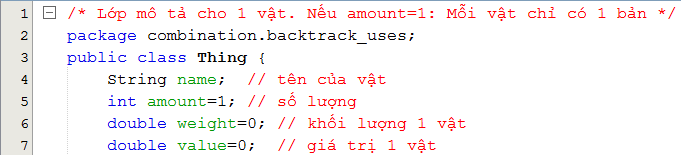
**4.4- Áp dụng giải bài toán cái túi (bài toán vali)**

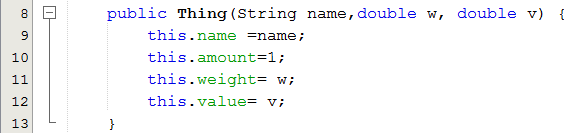
**Bài toán cái túi cơ bản**: Mỗi vật chỉ có 1 bản (thí dụ: bút chì chỉ có 1 bút) có khối lượng và giá trị riêng. Có một cái túi có khả năng chứa một khối lượng maxWeight cho trước. Hãy chọn các vật cho vào túi sao cho tổng giá trị túi lớn nhất.

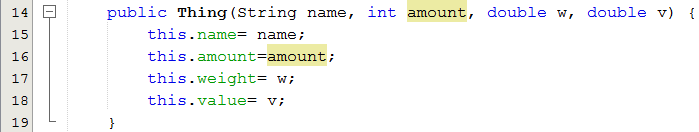
**Bài toán cái túi nâng cao**: Mỗi vật có thể có nhiều bản (thí dụ: bút chì có 5 cái) có khối lượng và giá trị riêng. Có một cái túi có khả năng chứa một khối lượng maxWeight cho trước. Hãy chọn các vật cho vào túi sao cho tổng giá trị túi lớn nhất. Dĩ nhiên có thể chọn cả 5 cái bút chì.

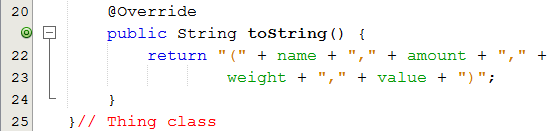
Như vậy, để tổng quát hóa, một đối tượng thực có số lượng thực. Thí dụ, bút chì có 5 cái. Nếu mỗi vật thật chỉ có 1 thực thể thì bài toán trở thành bài toán cái túi cơ bản.

Tóm lại, domain trong bài toán cái túi là {1,2,…, số lượng vật cùng loại}.

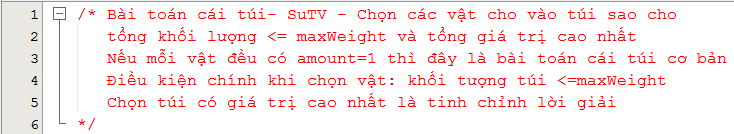


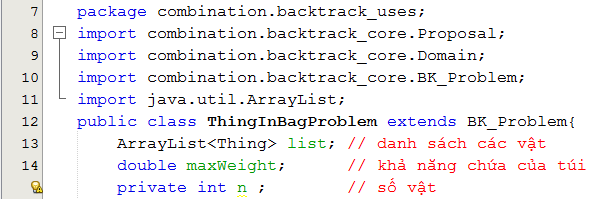


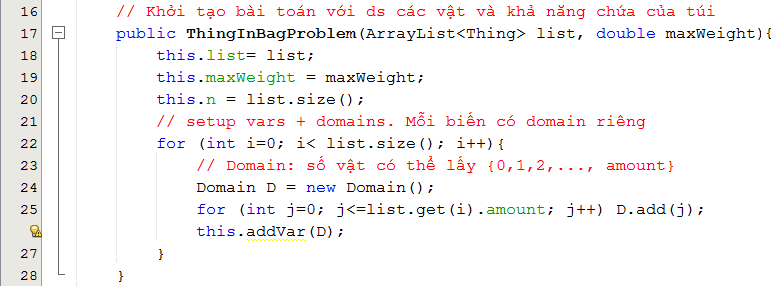


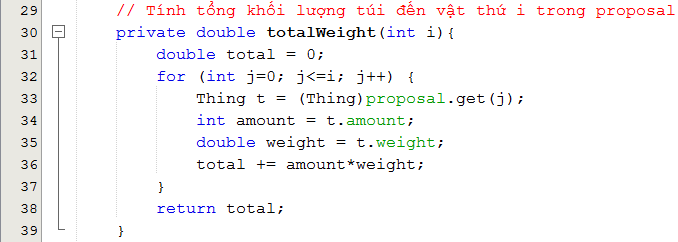


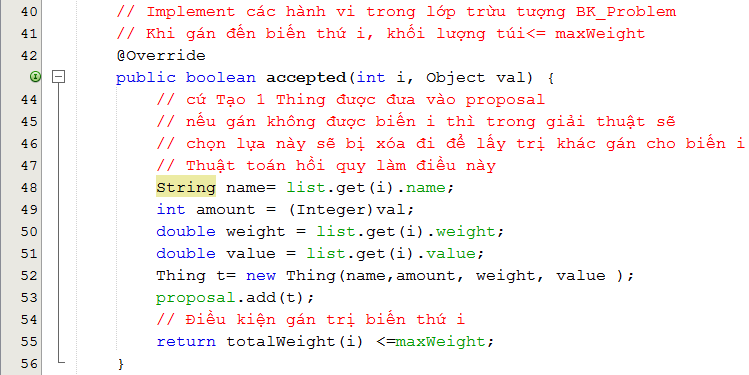
Trong bài toán cái túi sau đây, bài toán này được giải nhằm minh họa cách dùng mô hình giải bài toán bằng hồi quy theo cơ chế thu gom các lời giải thỏa điều kiện cơ bản rồi tìm ra lời giải tối ưu sau. Với một chút thay đồi khi hiện thực hành vi thêm lời giải, addSolution(), chúng ta có thể chỉ lấy lời giải tốt nhất. Phần này để dành cho độc giả.

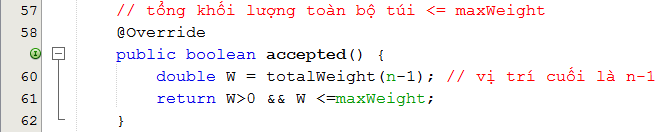


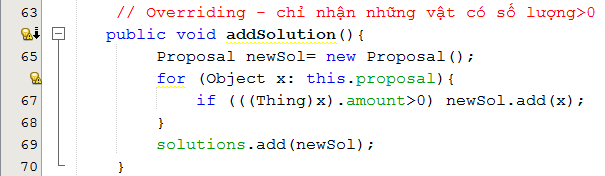


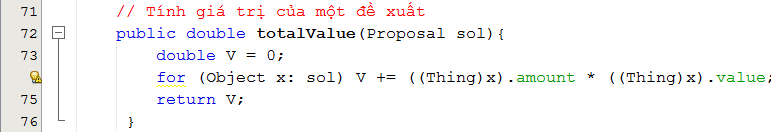


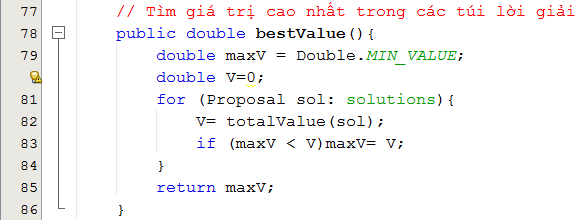


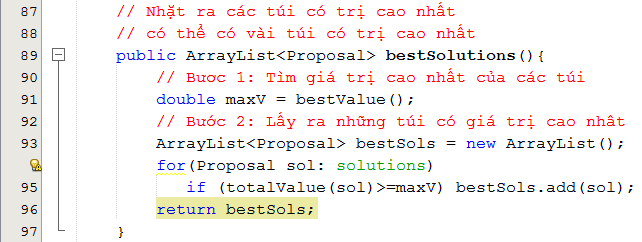


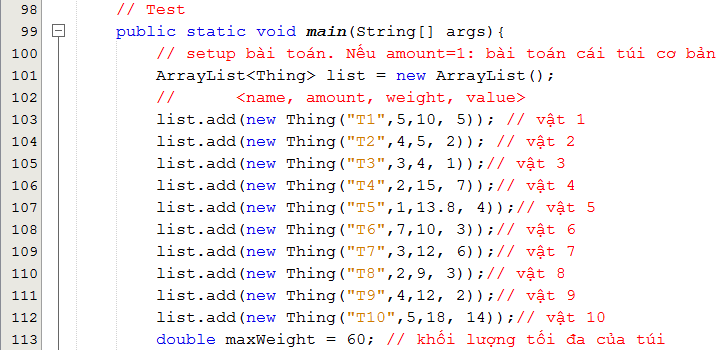


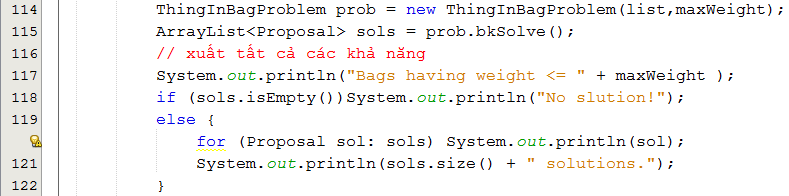


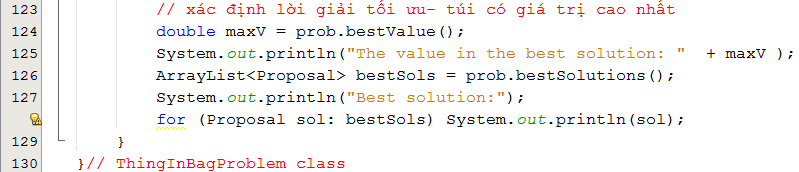




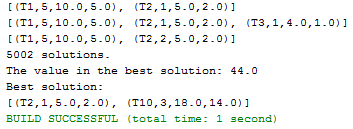






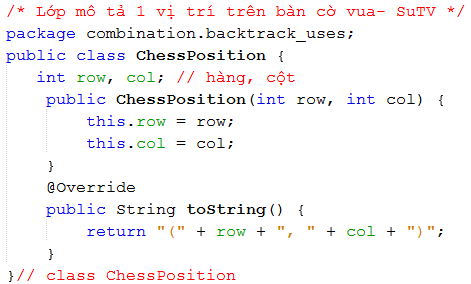


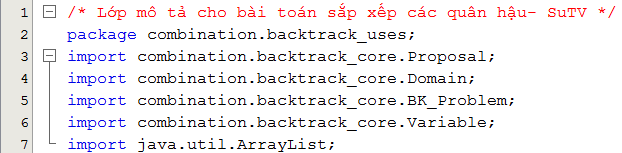
**Kết quả**

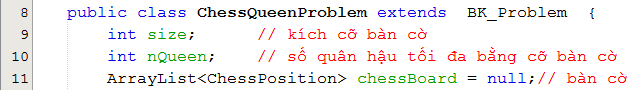


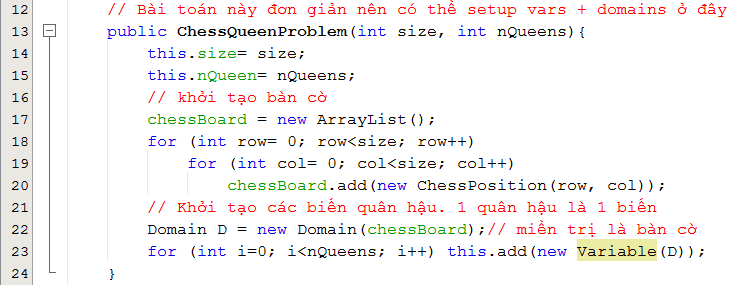
**4.5- Bài toán quân hậu**

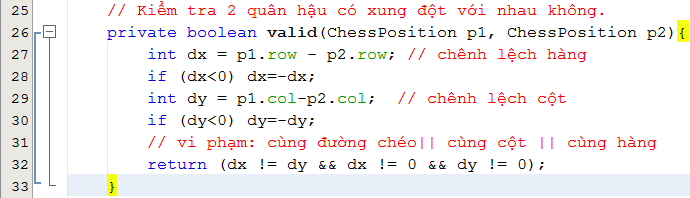
Cho bàn cờ vua có kích thước size\*size, cho số quân hậu nQueens. Hãy cho biết các cách sắp nQueens lên bàn cờ sao cho chúng không vi phạm luật chơi cờ (quân hậu có quyền ăn ngang, dọc, chéo).

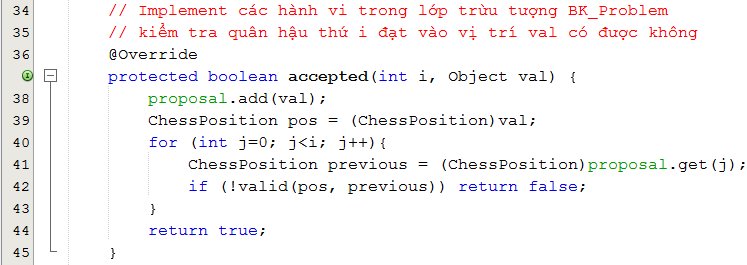


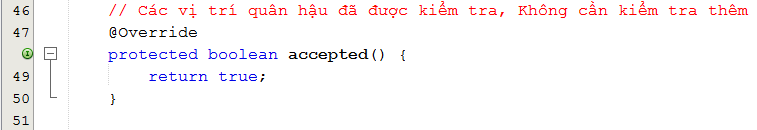


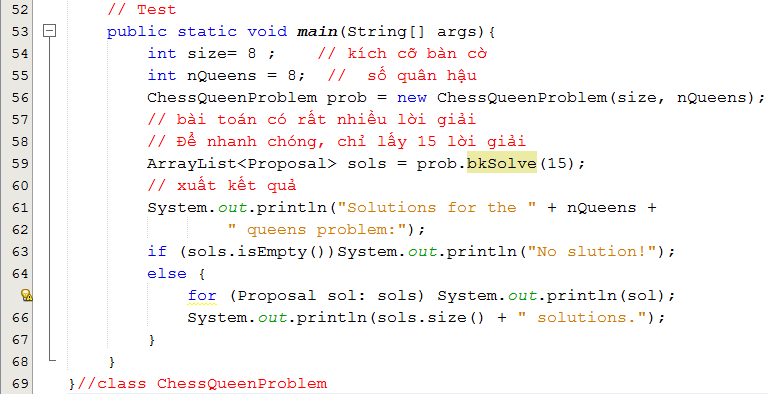




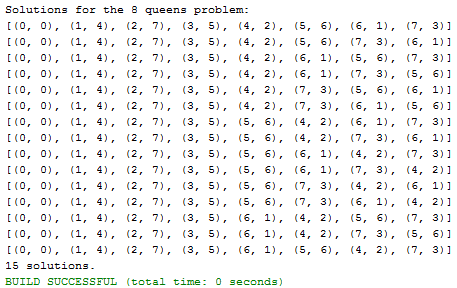








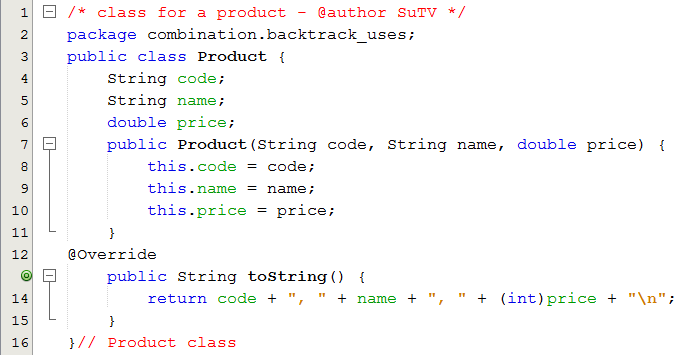
**Kết quả:**

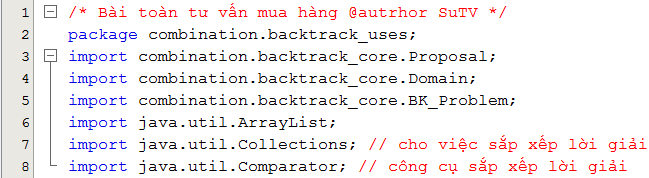


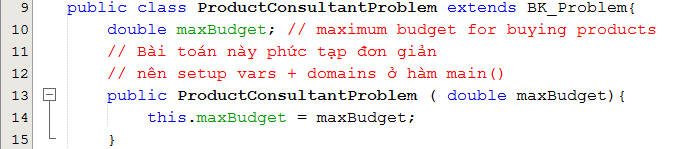
**4.5- Áp dụng giải bài toán tư vấn mua hàng**

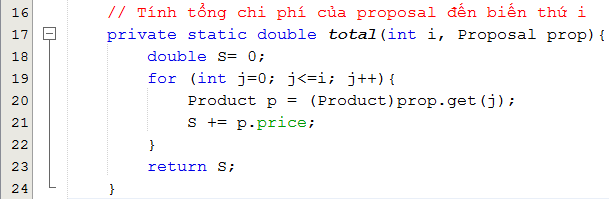
Tại một của hàng điện máy, một khách hàng muốn được tư vấn mua 3 TV, 2 tủ lạnh, 2 quạt với khoản đầu tư tối đa 30.000.000$. (Có thể áp dụng trong các website thương mại điện tử)

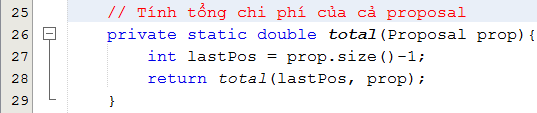
* Bài toán 7 biến, 3 biến choi 3 TV với domain là tập TV đang có trong cửa hàng, 2 biến cho 2 tủ lạnh và 2 biến cho 2 quạt.

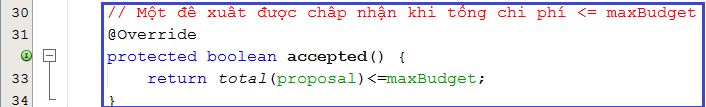


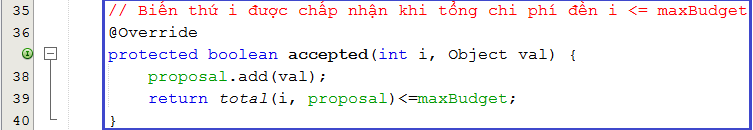


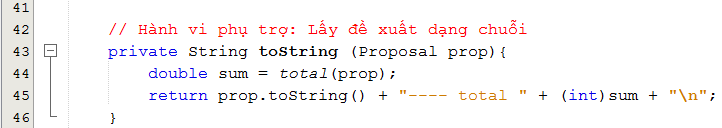


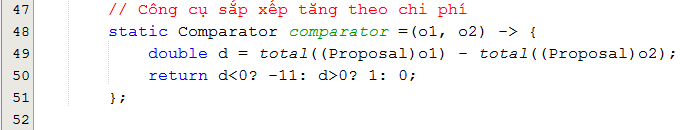


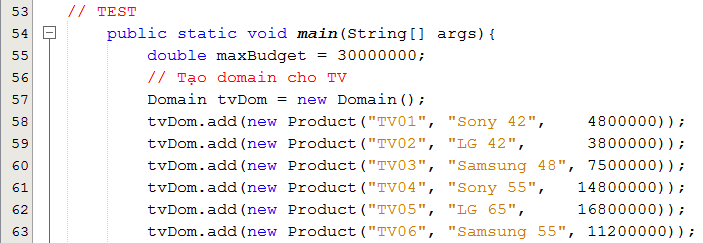


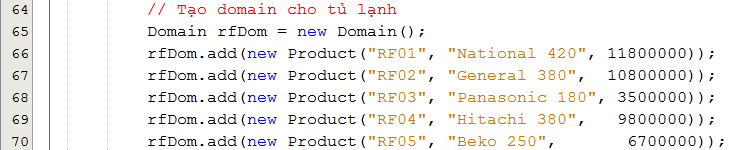


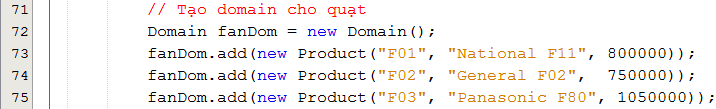


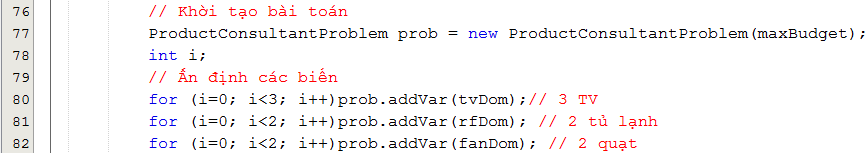


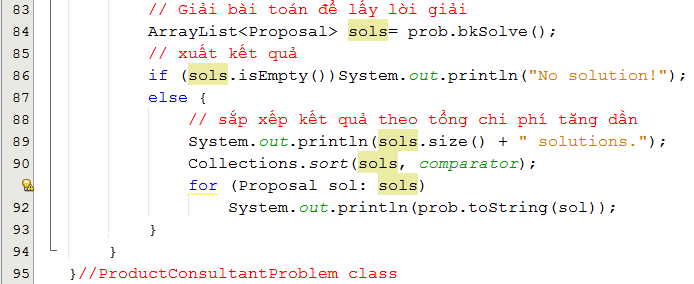




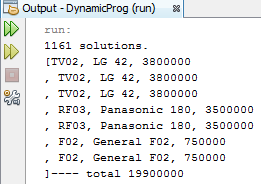








**Kết quả**



**5- Cách giải vét cạn với cơ chế sinh dữ liệu**

Khi áp dụng giải thuật hồi quy, độ phức tạp về thời gian của giải thuật là O(MN) và độ phức tạp về bộ nhớ của giải thuật là O(N). Câu hỏi đặt ra là co cách nào cải tiến hay không?

Vì dùng cách vét cạn nên không thể cải tiến độ phức tạp về thời gian nhưng nếu dùng vòng lặp thay cho hồi quy thì chúng ta tiết kiệm được chi phí bộ nhớ. Từ bài toán đã được thiết lập, chúng ta cần một cơ chế vòng lặp sinh ra các đề xuất (gọi là Generator) mà không quan tâm đến điều kiện của bài toán. Đề xuất này sẽ được kiểm tra xem có là một lời giải của bài toán hay không.

Generator sẽ nhận tập D[] từ bài toán để phát sinh các đề xuất. ***Đề xuất đầu*** chính là bộ trị mà mỗi trị là phần tử đầu tiên của mỗi D[i]. ***Đề xuất cuối*** chính là bộ trị mà mỗi trị là phần tử cuối của mỗi D[i]. Các hình sau minh họa cơ chế và giải thuật giải bài toán tổ hợp bằng cơ chế sinh dữ liệu:

Khởi tạo (D[])

**Generator**

D[]

***proposal***

**Problem**

Vars[]

D[]

solutions

Lấy proposal ()

proposal

*Mô hình giải bài toán với bộ sinh dữ liệu:*

Procedure solve (vars[], D[]) // giải bài toán

Khởi tạo tập Solutions là tập trống;

Khởi tạo Generator bằng tập miền trị của bài toán;

Lấy Proposal đầu tiên từ Generator;

Khi còn Proposal {

Nếu Proposal thỏa mãn ràng buộc thì thêm Proposal vào Solutions;

Lấy Proposal kế tiếp từ Generator;

}

Solutions là kết quả;

End

*Thuật toán giải bài toán bằng bộ sinh*

**Cơ chế sinh ra đề xuất kế tiếp từ đề xuất hiện hành**

Cơ chế sau minh họa phương pháp đi từ đề xuất này đến đề xuất kế tiếp với giả định là có 6 biến, các D[i] = {0,1,2,3,4,5,6,7,8,9}.

Đề xuất đầu: 0,0,0,0,0,0

Đề xuất cuối: 9,9,9,9,9,9

Đề xuất hiện hành: 0,0,4,9,9,9 🡪 Đề xuất kế tiếp: 0,0,5,0,0,0

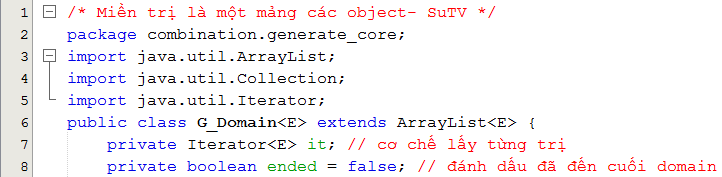
* Đi từ phải sang trái ( last = 5)
  + Cho các trị cuối domain về trị đầu cũa domain **0,0,4,9,9,9 🡪 0,0,4,0,0,0**

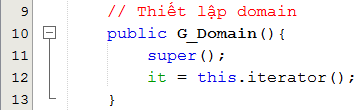
(Sau bước này last=2))

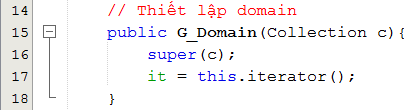
* + Gán trị kế tiếp cho vị trí last // **0,0,4,0,0,0**  🡪 0,0,**5,0,0,0**

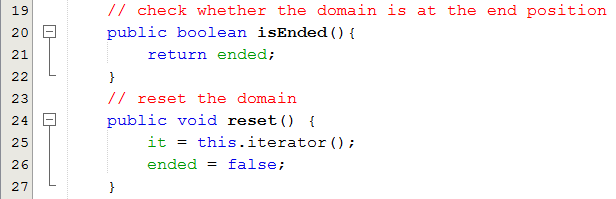
**6- Hiện thực bộ sinh dữ liệu**

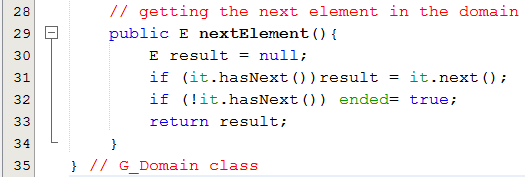
|  |  |
| --- | --- |
|  | **G\_Domain**: Lớp cho miền trị, lớp này hiện thực khác với lớp Domain ở trên vì phải thêm vào cơ chế lấy từng trị. **G\_Proposal:** Lớp cho 1 đề xuầt  **Generator**: Lớp mô tả cho bộ sinh dữ liệu với đầu vào là tập Domains  **G\_Problem**: Lớp trừu tượng cho bài toán giải bằng bộ sinh.  Các bài toán cụ thể áp dụng cơ chế giải này:   * Bài toán tư vấn mua hàng * Bài toán mảng con có tổng cho trước. |



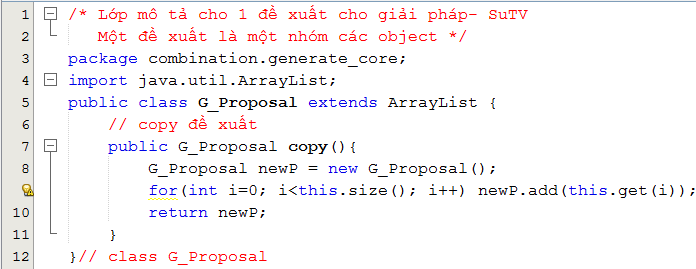


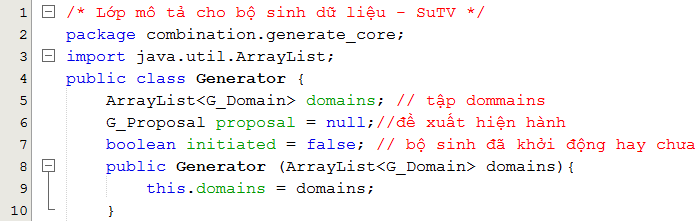


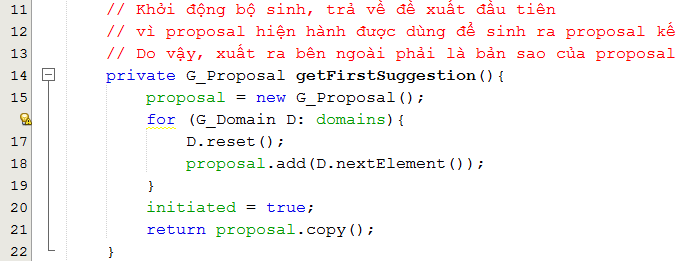


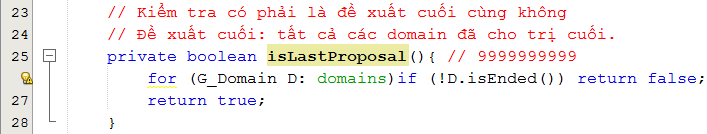


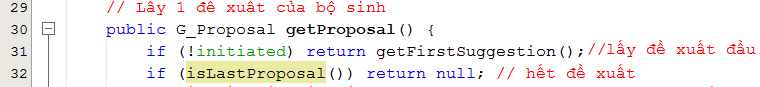
Lớp Generator duy trì một đề xuất (proposal) hiện hành. Do vậy, bên ngoài khi lấy đề xuất từ Generator thì sẽ lấy được một bản sao của đề xuất.

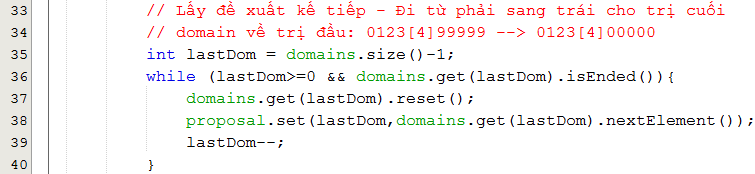


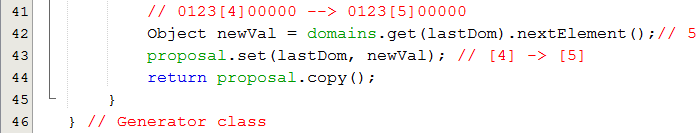


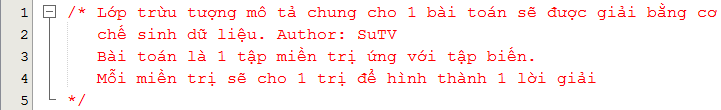


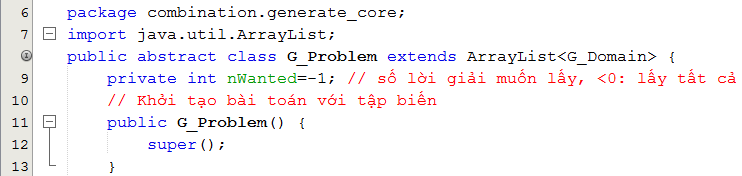


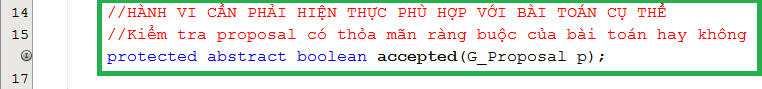


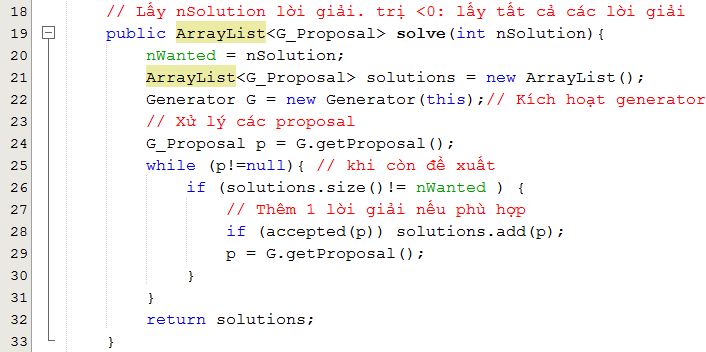


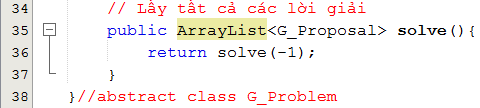




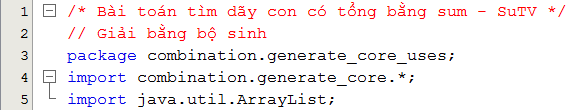


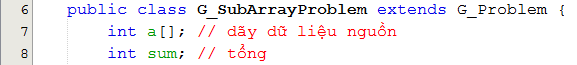


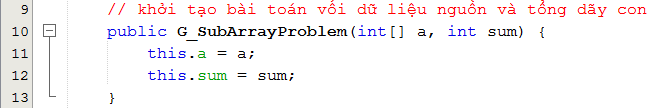


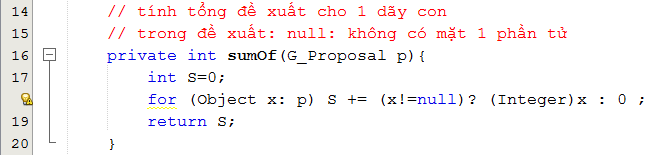


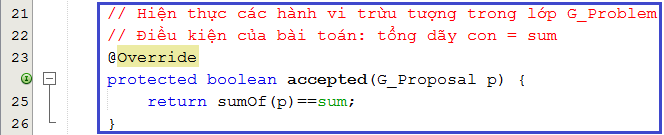
**6,1- Giải bài toán mảng con có tổng cho trước**

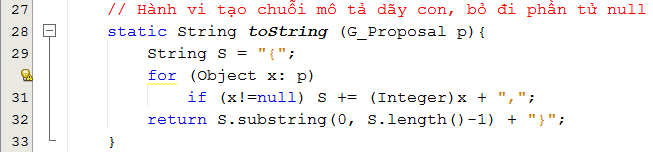


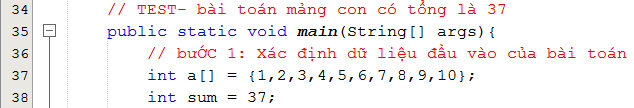


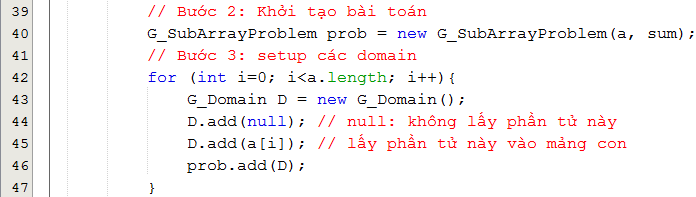


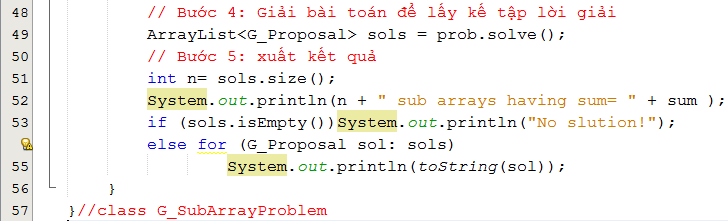












**6.2- Giải bài toán tư vấn mua hàng**

