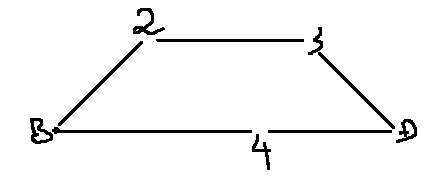
**Chiến lược tham lam (Greedy)**

Thuật toán Greedy (hay còn gọi là thuật toán tham lam) là một trong những kỹ thuật giải thuật quan trọng và phổ biến trong lập trình thi đấu. Khác với các chiến lược như quy hoạch động (Dynamic Programming) hay quay lui (Backtracking), Greedy đưa ra lựa chọn tại mỗi bước dựa trên tiêu chí tốt nhất tại thời điểm đó, mà không xét đến hậu quả lâu dài (tương lai).

Tuy nhiên, thuật toán Greedy không phải lúc nào cũng đảm bảo kết quả tối ưu. Việc xác định đúng tiêu chí lựa chọn và hiểu rõ bản chất bài toán là điều kiện tiên quyết. Vì vậy, cần sử dụng đúng hoàn cảnh, và luôn cần tư duy phản biện để kiểm chứng.

Ví dụ: xét tại thời điểm hiện tại

Giả sử có các phương án di chuyển từ điểm xuất phát (1) đến đích (n)



Ở đây có thể ban đầu ta B🡪Đ chi phí là 4 tức là chọn tại điểm hiện tại có thể là đúng, nhưng trong tương lai có thể là không đúng

Tuy nhiên đây là một chiến lược ta sử dụng được ví dụ một bài toán ta không biết lời giải tối ưu cho nó 🡪 TH này ta có thể dùng tham lam (có thể được một số test hoặc nhiều test.

**1. Chiến lược tham lam như thế nào:**

Thông thường các bài toán thường chia thành 1 số tập: tập kết quả và tập lựa chọn.

Giả sử ta có tập kết quả tại thời điểm đó (x1, x2… xk) chọn xk+1 trong tập lựa chọn (thông thường ta sẽ sắp xếp lại để lựa chọn được dễ dàng hơn.

Ví dụ: Cóp nhạc vào USB:

Có n file dung lượng là a1, a2, a3,…, an (mb) có 1 usb dung lượng m hỏi có thể copy tối đa bao nhiêu file vào usb.

Giải thuật: Sắp xếp tăng dần và ta sẽ lựa chọn các file có dung lượng từ nhỏ đến lớn, thoát khi tổng dung lượng được copy > dung lượng usb.

Thuật toán Greedy là một trong những chiến lược giải thuật phổ biến và hiệu quả, nhưng không phải là phương pháp duy nhất. Việc hiểu rõ sự khác biệt giữa Greedy và các kỹ thuật khác như Quy hoạch động (Dynamic Programming – DP), Vét cạn (Brute-force), và Quay lui (Backtracking) là rất quan trọng để lựa chọn chiến lược giải đúng và tối ưu trong từng bài toán cụ thể. Greedy chỉ xét một lựa chọn tại mỗi bước – lựa chọn cục bộ tốt nhất. Không lưu trữ toàn bộ lời giải của các bài toán con. Điều này giúp Greedy rất nhanh và đơn giản, nhưng tiềm ẩn nguy cơ sai nếu lựa chọn không đảm bảo thuộc về lời giải tối ưu. Dynamic Programming (DP) lưu trữ trạng thái các bài toán con, từ đó xây dựng lời giải bài toán lớn một cách toàn diện. Nếu một bài toán có cấu trúc con.

**2. Một số bài toán**

**Bài 1: Bài toán đổi tiền (Coin Change Problem)** là một trong những bài toán kinh điển của thuật toán tham lam. Mục tiêu là đổi một số tiền T bằng cách sử dụng số lượng ít nhất các đồng tiền từ một hệ mệnh giá cho trước.

**Mô tả bài toán:**

* Cho một tập hợp các mệnh giá đồng tiền d1,d2,…,dk trong đó d1<d2<⋯<dk
* Cho một số tiền cần đổi T, yêu cầu tìm cách đổi T thành số lượng đồng tiền ít nhất.

**Thuật toán tham lam:**

* **Chiến lược tham lam**: Luôn chọn đồng tiền có mệnh giá lớn nhất không vượt quá số tiền còn lại. Lặp lại quá trình này cho đến khi số tiền còn lại bằng 0.

**Phân tích ví dụ:**

Giả sử hệ mệnh giá là {1,2,5,10} và ta cần đổi số tiền T=28.

**Bước 1:**

* Chọn đồng tiền có mệnh giá lớn nhất không vượt quá T=28, đó là đồng 10.
* Ta chọn một đồng 10, số tiền còn lại là 28−10=18

**Bước 2:**

* Tiếp tục chọn đồng tiền có mệnh giá lớn nhất không vượt quá số tiền còn lại 18, đó là đồng 10.
* Ta chọn một đồng 10 nữa, số tiền còn lại là 18−10=8

**Bước 3:**

* Chọn đồng tiền có mệnh giá lớn nhất không vượt quá 8, đó là đồng 5.
* Ta chọn một đồng 5, số tiền còn lại là 8−5=3

**Bước 4:**

* Chọn đồng tiền có mệnh giá lớn nhất không vượt quá 3, đó là đồng 2.
* Ta chọn một đồng 2, số tiền còn lại là 3−2=1

**Bước 5:**

* Cuối cùng, chọn đồng tiền có mệnh giá lớn nhất không vượt qua 1, đó là đồng 1.
* Ta chọn một đồng 1, số tiền còn lại là 1−1=0

**Kết quả:**

* Số lượng đồng tiền cần dùng là: 10,10,5,2,1 (tổng cộng là 5 đồng).

**Phân tích độ chính xác của thuật toán tham lam:**

* Thuật toán tham lam cho ra kết quả đúng khi hệ mệnh giá là **đơn điệu**, tức là không có mệnh giá nhỏ hơn nào có thể thay thế mệnh giá lớn hơn với tổng chi phí thấp hơn. Trong ví dụ này, hệ mệnh giá {1,2,5,10 thoả mãn tính đơn điệu, vì không có sự kết hợp nào tốt hơn việc chọn các đồng tiền lớn nhất trước.

Tuy nhiên, thuật toán tham lam không luôn cho kết quả tối ưu với tất cả các hệ mệnh giá. Ví dụ, nếu hệ mệnh giá là {1,3,4}và ta cần đổi số tiền 6, thuật toán tham lam sẽ chọn:

* 1 đồng 4 (còn lại 2),
* 2 đồng 1 (còn lại 0),

Tổng cộng là 3 đồng, nhưng kết quả tối ưu là chỉ cần 2 đồng: 1 đồng 3 và 1 đồng 3.

Do đó, học sinh cần biết kiểm tra cấu trúc mệnh giá (xem có đồng tiền lớn nào có thể được thay thế bởi cách kết hợp nhiều đồng tiền nhỏ hơn không) trước khi áp dụng Greedy. Tránh vội vàng dùng chiến lược "tham lam chọn đồng lớn nhất", cần xem xét tính chất hệ mệnh giá để đảm bảo rằng Greedy thực sự phù hợp. Nếu không, phải dùng thuật toán khác như quy hoạch động.

* Phân tích sai lầm “**Đưa ra lựa chọn cục bộ tốt nhất**”

**Bài toán 2. Bài toán Knapsack phân chia (Fractional Knapsack)**

**Mô tả bài toán:**

* Cho một túi có sức chứa W (khối lượng tối đa của túi).
* Cho một danh sách các vật phẩm, mỗi vật phẩm có trọng lượng wi và giá trị vi
* Mục tiêu là chọn các vật phẩm sao cho tổng giá trị là tối đa, với điều kiện tổng trọng lượng không vượt quá W. Với bài toán phân chia, ta có thể chọn một phần của vật phẩm (không bắt buộc phải chọn toàn bộ vật phẩm).

**Thuật toán tham lam:**

* Thuật toán tham lam sẽ chọn các vật phẩm có **giá trị cao nhất trên đơn vị trọng lượng** (tức là giá trị vi/wi cao nhất) cho đến khi túi đầy.

**Mô tả bài toán:**

* Vật phẩm 1: Trọng lượng = 2, Giá trị = 10
* Vật phẩm 2: Trọng lượng = 3, Giá trị = 15
* Sức chứa của túi: W=4

**Áp dụng thuật toán tham lam:**

1. **Tính giá trị trên đơn vị trọng lượng của từng vật phẩm**:
   * Vật phẩm 1: 10/2=5 giá trị/kg
   * Vật phẩm 2: 15/4= 3.75 giá trị/kg
2. **Chọn vật phẩm có giá trị cao nhất trên đơn vị trọng lượng**:

Vật phẩm 1 có giá trị cao nhất trên đơn vị trọng lượng, vì 5>3.75

+> **Chọn vật phẩm 1**: Trọng lượng của vật phẩm 1 là 2, còn lại W−2=4−2=2

**+> Chọn vật phẩm tiếp theo**: Không thể chọn vật phẩm 2 vì nó có trọng lượng 4, vượt quá sức chứa còn lại (2).

**Kết quả tham lam**:

* Chọn **vật phẩm 1** với tổng trọng lượng 2 và tổng giá trị 10.
* Tổng trọng lượng: 2, Tổng giá trị: 10.

Thuật toán tham lam chỉ chọn vật phẩm có giá trị cao nhất trên đơn vị trọng lượng mà không tính đến các sự kết hợp có thể tối ưu hơn.

**Kết quả tối ưu**: Nếu ta chọn **vật phẩm 2** (trọng lượng = 4, giá trị = 15), ta sẽ sử dụng hết sức chứa của túi và đạt tổng giá trị là 15, cao hơn giá trị 10 mà thuật toán tham lam đưa ra.

**🡪 Hậu quả:**

- Sai lầm do không thỏa mãn tính chất (**Đưa ra lựa chọn cục bộ tốt nhất)**

### **- Sai lầm do không có cấu trúc con tối ưu**

- Sai lầm do tiêu chí tham lam không phù hợp

- Sai lầm do đánh giá sai bản chất bài toán

Bài toán 3: **Mô tả bài toán:**

* Cho một chuỗi nhị phân gồm các số 0 và 1.
* Mục tiêu là hoán vị lại chuỗi sao cho tất cả số 0 nằm về bên trái và số 1 về bên phải.
* Cần thực hiện ít nhất số lần đổi chỗ.

**Ý tưởng thuật toán tham lam:**

Một cách tự nhiên, **thuật toán tham lam** có thể thực hiện như sau:

1. Quét từ trái qua phải, mỗi khi gặp một số 1 phía trước và số 0 phía sau, ta sẽ đổi chỗ chúng.
2. Lặp lại quá trình này cho đến khi tất cả các số 0 nằm về bên trái và số 1 về bên phải.

**Trực giác** của thuật toán tham lam là: Nếu gặp cặp 1 và 0, đổi chỗ chúng để gần nhau hơn.

**Ví dụ cụ thể:**

Cho chuỗi nhị phân: **1 0 0 1 0 1**

**Thuật toán tham lam**:

Quét từ trái qua phải:

- Gặp cặp (1, 0) ở chỉ số 1 và 2 → đổi chỗ → chuỗi: **0 1 0 1 0 1**.

- Tiếp tục quét, gặp cặp (1, 0) ở chỉ số 3 và 4 → đổi chỗ → chuỗi: **0 0 1 1 0 1**.

- Cuối cùng, gặp cặp (1, 0) ở chỉ số 4 và 5 → đổi chỗ → chuỗi: **0 0 0 1 1 1**.

Tổng cộng: **3 lần đổi chỗ**.

**Giải pháp tối ưu**:

* + Một cách sắp xếp tối ưu hơn sẽ là chỉ cần **2 lần đổi chỗ**.
    - Đổi cặp (1, 0) ở chỉ số 1 và 4 → chuỗi: **0 0 0 1 1 1**.
    - Đổi cặp (1, 0) ở chỉ số 2 và 5 → chuỗi: **0 0 0 1 1 1**.

**Số lần đổi chỗ tối ưu** là **2**, ít hơn so với kết quả của thuật toán tham lam (3 lần).

* Sai lầm do bài toán tham lam "giả"

**Bài tập vận dụng**

**Bài 1:dragon.cpp**

Sau khi không còn gì để đỡ 2 thằng em của mình nữa, Tý quay ra chơi điện tử. Tý rất mê trò chơi điều khiển nhân vật Hiccup của mình chiến đấu cùng với những con rồng, vượt qua tất cả sẽ nhận được chứng chỉ dũng sĩ dragon. Hiccup chỉ có thể vượt qua 1 con rồng nếu như chỉ số chiến đấu của cậu lớn hơn chỉ số chiến đấu của con rồng đó, và sau khi giành chiến thắng, chỉ số chiến đấu của cậu sẽ tăng thêm một mức nhất định tùy vào con rồng mà cậu đánh bại.

Tý muốn biết liệu Hiccup có thể nhận được chứng chỉ không? Bạn hãy giúp Tý với.

**INPUT:**

* Dòng đầu tiên gồm 2 số nguyên **s** và **n**, trong đó **n** là số con rồng Hiccup cần phải vượt qua (1 ≤ n ≤ 105), **s** là chỉ số chiến đấu ban đầu của Hiccup (1 ≤ s ≤ 10⁴).
* n dòng tiếp theo, mỗi dòng gồm 2 số nguyên **x[i]** và **y[i]**. **x[i]** là chỉ số chiến đấu của con rồng thứ i (1 ≤ x[i] ≤ 10⁴), **y[i]** là mức tăng chỉ số cho Hiccup nếu cậu đánh bại nó (0 ≤ y[i] ≤ 10⁴).

**OUTPUT:**

* In ra **"YES"** nếu Hiccup nhận được chứng chỉ dũng sĩ dragon.
* Trường hợp ngược lại:
  + Dòng 1 ghi **"NO"** nếu Hiccup không nhận được chứng chỉ dũng sĩ dragon.
  + Dòng 2 đưa ra số lượng rồng còn lại chưa bị tiêu diệt.

|  |  |
| --- | --- |
| **input** | **output** |
| 2 2  1 99  100 0 | YES |
| 1 1  100 10 | NO  1 |

# Bài 2: **BWPOINTS.cpp**

Trên trục số thực cho n điểm đen và n điểm trắng hoàn toàn phân biệt. Các điểm đen có tọa độ nguyên *a1, a2, …, an* còn các điểm trắng có tọa độ nguyên *b1, b2, …, bn*. Người ta muốn chọn ra k điểm đen và k điểm trắng để nối mỗi một điểm đen với một điểm trắng sao cho *k* đoạn thẳng tạo được đôi một không có điểm chung.

**Yêu cầu:** Cho tọa độ của n điểm đen *a1, a2, …, an* và tọa độ của điểm trắng *b1, b2, …, bn*. Hãy tìm giá trị k lớn nhất thỏa mãn yêu cầu trên.

**Dữ liệu:**

* Dòng thứ nhất chứa số nguyên dương n (n <= 105).
* Dòng thứ hai chứa các số *a1, a2, …, an* (|ai| <= 109, i = 1, 2,…, n)
* Dòng thứ ba chứa các số *b1, b2, …, bn* (|bi| <= 109, i = 1, 2,…, n)

      Các số trên cùng một dòng được ghi cách nhau ít nhất một dấu cách.

**Kết quả:** Ghi ra một số nguyên duy nhất là số *k* lớn nhất tìm được

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  0 3 1  -3 5 -1 | http://vn.spoj.com/content/voj:BWPOINTS.png2 |

**Ràng buộc:** 50% số test ứng với 50% số điểm của bài có 1 <= n <= 100

**Bài 3: THIEF.cpp**

Ở vương quốc Berland, thời đó, người ta quý nhất là diêm (loại đồ vật tạo ta lửa). Một tên trộm lẻn vào kho diêm của nhà vua Berland, trong kho có *m* hòm đựng diêm. Hòm thứ *i* chứa *ai* bao diêm, với mỗi bao diêm trong hòm này có *bi* que diêm (các bao diêm là giống nhau). Ba lô của tên trộm chỉ chứa được không quá *n* bao diêm. Em hãy giúp hắn chọn lấy các bao diêm để số que diêm hắn lấy được là nhiều nhất có thể. Biết rằng tên trộm không có nhiều thời gian để mở các bao diêm.

Dữ liệu:

* Dòng đầu chứa hai số nguyên *n* (1≤ *n* ≤ 2\*108) và *m* (1 ≤ *m* ≤ 20).
* Dòng thứ *i* trong *m* dòng sau chứa hai số nguyên *ai*và *bi* (1 ≤ *ai* ≤ 108,1≤ *bi* ≤10).

Kết quả:

In ra số que diêm tối đa có thể lấy.

Ví dụ:

|  |  |  |  |
| --- | --- | --- | --- |
| **Input1** | **Output1** | **Input2** | **Output2** |
| 7 3 5 10 2 5 3 6 | 62 | 3 3 1 3 2 2 3 1 | 7 |

**Bài 4: tlcontan.cpp**

Một đoàn tàu có *n* toa xe được đánh số từ 1 đến *n* theo thứ tự bất kỳ (không trùng lặp). Trong một thao tác, bạn có thể chọn một toa bất kỳ, dịch chuyển nó khỏi vị trí hiện tại và đưa vào **đầu** hoặc **cuối** của đoàn tàu. Mục tiêu là sắp xếp các toa theo thứ tự tăng dần từ 1 đến *n* bằng số thao tác ít nhất.

# **Input:**

* Dòng đầu chứa số nguyên *n* (1 ≤ *n* ≤ 105).
* Dòng thứ hai chứa *n* số nguyên phân biệt *p*1*, p*2*, . . . , pn* — thứ tự hiện tại của các toa tàu.

**Output:** In ra số nguyên duy nhất — số thao tác tối thiểu để sắp xếp đoàn tàu.

# Ví dụ:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  4 1 2 5 3 | 2 |
| 4  4 1 3 2 | 2 |

**Bài 5: tlkho.cpp**

Có *m* kho chứa hàng, mỗi kho chứa tối đa *w* container. Có *n* lô hàng, mỗi lô hàng *i* gồm *ai* container.

Mỗi lần, chủ hàng sẽ gửi lô hàng *i* vào một kho bất kỳ sao cho:

* Kho đó còn đủ chỗ chứa *ai* container.
* Kho được chọn có chỉ số nhỏ nhất thoả mãn yêu cầu trên

Yêu cầu: Với mỗi lô hàng *i*, xác định kho mà nó được gửi vào, hoặc in ra −1 nếu không có kho nào phù hợp.

**Input:**

* + - * Dòng đầu chứa 3 số nguyên *m*, *w*, *n* (1 ≤ *m, n* ≤ 2 · 105, 1 ≤ *w* ≤ 2 · 109).
      * *n* dòng tiếp theo, mỗi dòng chứa một số nguyên *ai* — số container của lô hàng i 1 ≤ *ai* ≤ w

**Output:** Gồm *n* dòng, mỗi dòng in chỉ số kho nhận lô hàng *i* hoặc -1 nếu không tìm được.

|  |  |
| --- | --- |
| **input** | **Output** |
| 3 5 5  1 2 3 4 5 | 1  1  2  3  -1 |

**Bài 06: tlcir.cpp**

Cho một dãy *n* số nguyên không âm *a*1*, a*2*, . . . , an* được sắp xếp theo hình tròn, trong đó *n* là số lẻ.

Trong mỗi bước, bạn được phép:

* + - * Chọn một phần tử *ai*.
      * Tính tổng hai phần tử liền kề với nó là *ai−*1 và *ai*+1 (lưu ý theo vòng tròn:

*a*0 = *an*, *an*+1 = *a*1).

* Thay *ai* bằng tổng đó, đồng thời xoá *ai−*1 và *ai*+1 khỏi vòng tròn. Lặp lại thao tác trên cho đến khi chỉ còn lại một phần tử.

**Yêu cầu:** Tìm giá trị lớn nhất có thể nhận được ở phần tử cuối cùng.

Input

* + - * Dòng đầu chứa một số nguyên lẻ *n* (1 ≤ *n <* 2 · 105).
      * Dòng tiếp theo chứa *n* số nguyên *a*1*, a*2*, . . . , an (*1 ≤ *ai* ≤ 109

|  |  |
| --- | --- |
| input | output |
| 3  7 10 2 | 17 |