

Thuật toán Tham lam

Thuật toán Tham lam

Nguồn bài: [Topcoder](#) [🔗](#)

Ví dụ minh họa

John Smith đang gặp rắc rối! Anh ấy là một thành viên của **Topcoder** và sau khi học cách để trở thành bậc thầy trong việc đối phó với các bài toán quy hoạch động, anh ấy bắt đầu giải quyết hàng loạt các bài tập. Nhưng chiếc máy tính "dễ bảo" của anh bắt đầu trở chứng vào hôm nay. Vào mỗi buổi sáng như thường lệ, John thức dậy vào lúc 10 giờ sáng, uống một cốc cà phê và bắt đầu giải các bài tập trước khi thưởng thức bữa sáng. Mặc dù có thứ gì đó "sai sai" so với mọi hôm, nhưng dựa vào kho tàng kiến thức mà anh ấy vừa mới gặt hái được, John đã viết chương trình với một tốc độ thần thánh. Một mối với việc cấp phát ma trận vào mỗi buổi sáng, chiếc máy tính thông báo rằng: **"Segmentation fault!"**. Dù cho dạ dày còn đang rỗng, song với ý tưởng thông minh của mình, John đã vượt qua rắc rối bằng cách chèn thêm một vòng lặp. Nhưng chiếc máy tính lại gào lên: **"Time limit exceeded!"**.

Thay vì tiếp tục vướng vào mớ rắc rối, John đã có một quyết định thông minh hơn. *"Quá đủ cho việc lập trình!"* - John nói. Anh ấy quyết định sẽ có một kỳ nghỉ như là phần thưởng cho những nỗ lực của mình.

Là một con người tràn trề sinh lực, John muốn dành thời gian nhiều cho cuộc đời của mình. Có quá nhiều thứ mà anh ta muốn làm, nhưng không may là anh ta không thể nào làm hết tất cả chúng được. Thế nên trong lúc ăn sáng, John đã vạch ra một *"Fun plan"* được thể hiện bằng một thời gian biểu cho từng hoạt động như sau:

Id	Hoạt động	Thời gian
1	Sửa phòng	Thứ 2, 22:00 đến thứ 3, 1:00
2	Du lịch Hawaii	Thứ 3, 6:00 đến thứ 7, 22:00
3	Vô địch cuộc thi cờ vua	Thứ 3, 11:00 đến thứ 3, 21:00
4	Thăm dự nhạc hội Rock	Thứ 3, 19:00 đến thứ 3, 23:00
5	Chiến thắng cuộc thi Starcraft	Thứ 4, 15:00 đến thứ 5, 15:00
6	Chơi trò bắn súng nước sơn	Thứ 5, 10:00 đến thứ 5, 16:00
7	Tham gia kỳ thi SRM trên Topcoder	Thứ 7, 12:00 đến thứ 7, 14:00

8	Tắm rửa	Thứ 7, 20:30 đến thứ 7, 20:45
9	Tổ chức tiệc ngủ	Thứ 7, 21:00 đến Chủ nhật, 6:00
10	Tham gia thử thách "All you can eat" và "All you can drink"	Thứ 7, 21:01 đến thứ 7, 23:59

Giờ anh ấy muốn thực hiện được tối đa các hoạt động trong thời gian biểu trên. Mặc dù để lên kế hoạch hiệu quả thì cần phải có chút lý trí, nhưng giờ thì hồn anh ấy đã đắm chìm vào kỳ nghỉ rồi.

Phát biểu bài toán

Liệu ta có thể giúp anh ấy có một kỳ nghỉ tuyệt vời? Ta hoàn toàn có thể! Đầu tiên, ta cần phải trình bày lại bài toán:

- John có N hoạt động. Hoạt động i bắt đầu vào thời gian L_i và kết thúc vào thời gian R_i .
- John có thể thực hiện cả hoạt động i và hoạt động j , nếu 2 khoảng thời gian $[L_i, R_i]$ và $[L_j, R_j]$ không giao nhau.
- Là một lập trình viên tỉ mỉ, một khi đã đặt ra kế hoạch, anh ấy buộc phải thực hiện nó.

Mỗi hoạt động chỉ có hai chọn lựa là có hoặc không. Với mỗi trường hợp chọn lựa cho hoạt động thứ nhất, ta lại có thêm 2 lựa chọn cho hoạt động thứ 2. Phân tích nhanh ta sẽ thấy được rằng có 2^n trường hợp, và trong tình huống này thì sẽ có $2^{10} = 1024$ trường hợp. Với mỗi trường hợp ta sẽ kiểm tra xem có thể thực hiện được các hoạt động đó không: có 2 cặp hoạt động nào bị trùng thời gian hay không. Sau khi xét hết các phương án, ta dễ dàng tìm ra 1 phương án có nhiều hoạt động nhất. Với khá nhiều sự chọn lựa như thế này, John buộc phải nhờ đến sự giúp đỡ của chiếc máy tính đang mệt mỏi. Nhưng điều gì sẽ xảy ra nếu John có tới 50 hoạt động trong danh sách? Thậm chí dùng đến cả siêu máy tính nhanh nhất thế giới thì cũng cần đến vài năm để tìm ra câu trả lời. Thế nên, phương án này khá phi thực tế.

Tiếp cận 1

Chúng ta cần tìm một cách tiếp cận mới. Một phương án tốt có lẽ là thực hiện công việc ngay khi thời cơ đến: Nếu ta có hai hoạt động và chúng bị trùng về thời gian, ta sẽ ưu tiên lựa chọn hoạt động bắt đầu trước nhằm tiết kiệm thời gian (nghĩa là hoạt động có L_i nhỏ hơn). Nếu áp dụng cách này vào 10 hoạt động trên thì:

- John bắt đầu buổi tối T2 bằng việc *Sửa phòng*.
- Buổi sáng T3, anh ấy sẽ lên máy bay đi *Du lịch Hawaii*.

Và chưa đầy một ngày nhưng anh ấy đã thực hiện được 2 hoạt động. Thật tuyệt vời! Nhưng thật ra, đó chỉ là **lựa chọn tốt nhất** lúc này thôi. Và bây giờ thì ta có gì, 5 ngày ăn chơi ở Hawaii và cho đến tận tối thứ 7 thì anh ấy vẫn chỉ thực hiện được 2 hoạt động. Hãy nghĩ xem trong 5 ngày đó anh ta đã có thể thực hiện được những gì. Mặc dù đơn giản và thực thi rất nhanh, song rất không may là thuật toán này lại không chính xác.

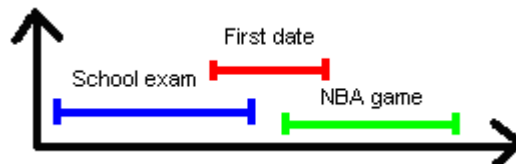
Tiếp cận 2

Hãy thử một mảnh khốe khác. Giờ ta sẽ bỏ những hoạt động tiêu tốn nhiều thời gian như đi *Du lịch Hawaii* bằng cách lựa chọn những hoạt động tốn ít thời gian nhất (nghĩa là có $R_i - L_i$ nhỏ nhất) và kiểm tra xem nó có hợp lý với những hoạt động đã chọn trước đó chưa rồi tiếp tục quá trình. Theo như thời gian biểu ở trên thì hoạt

động đầu tiên được chọn lựa sẽ là tắm. Với thời gian chỉ 15 phút, đây chính là **lựa chọn tối ưu cục bộ (local best)**. Và giờ điều mà ta cần biết đó là có thể giữ được **tối ưu cục bộ** khi mà những hoạt động thích hợp khác được chọn lựa. Thời gian biểu của John sẽ như sau:

- Tắm rửa (15 phút, tối T7, 20:30 - 20:45).
- Tham gia kỳ thi SRM trên Topcoder (2 tiếng, T7, 12:00 - 14:00).
- Tham gia thử thách "All you can eat" và "All you can drink" (2 tiếng 58 phút, T7, 21:01 - 23:59).
- Sửa phòng (3 tiếng, T2, 22:00 - T3, 1:00).
- Thăm dự nhạc hội Rock (4 tiếng, T3, 19:00 - 23:00).
- Chơi trò bắn súng nước sơn (6 tiếng, T5, 10:00 - 16:00).

Trong 10 hành động, ta đã lựa ra được 6 hành động, không tệ chút nào! Giờ thì thuật toán của ta vẫn chạy rất nhanh và đáng tin cậy hơn chút. Và quả thực, đáp án chính xác trong trường hợp này là 6. John rất hài lòng về sự hỗ trợ của chúng ta, nhưng sau khi trở về từ kỳ nghỉ với kế hoạch thông minh này, John đã phải đối mặt với những rắc rối nghiêm trọng khác:

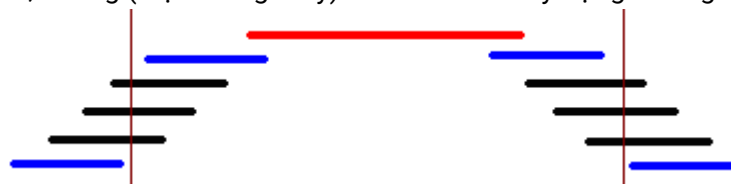


Áp dụng thuật toán của ta, John đã tham gia một cuộc hẹn hò chóng vánh (màu đỏ), để rồi anh ấy đã bỏ lỡ cả bài thi trong trường (màu xanh da trời) lẫn trận đấu bóng rổ của đội anh ấy yêu thích (màu xanh lá). Là một **Topcoder**, chúng ta cần phải viết ra một chương trình hoàn toàn chính xác, chứ không phải chỉ đúng trong 1 số trường hợp. Chỉ cần một trường hợp duy nhất chúng ta không giải quyết được sẽ dẫn tới một thất bại toàn diện.

Tiếp cận 3

Những gì mà chúng ta thường làm trong tình huống này là phân tích điều gì đã gây ra lỗi ở cách làm trước để tránh lặp lại nó trong tương lai. Hãy xem xét lại trường hợp sai. Cuộc hẹn hò trùng thời gian với cả việc làm bài thi lẫn trận đấu bóng rổ, trong khi cả trận đấu bóng rổ lẫn làm bài thi chỉ trùng lặp với một mình cuộc hẹn hò. Vậy thì ý tưởng cũng tự sinh ra từ vấn đề này. Tại sao ta không chọn hoạt động ít bị trùng lặp nhất so với những hoạt động còn lại? Nghe có vẻ hợp lí!

Để đảm bảo rằng phương pháp này hoàn toàn đúng đắn, lần này ta hãy thử chứng minh. Giả sử ta đã lựa chọn hoạt động X, ta sẽ thử kiểm tra xem ta có thể lựa chọn hoạt động A và B (những hoạt động bị trùng lặp với X) thay vì X được hay không. Và A, B cũng không được trùng lặp nhau, nếu không ta cũng không thể tối ưu hóa kết quả. Bây giờ, ta sẽ quay về trường hợp trước đó (X trùng với 2 hoạt động, A và B trùng với 1 hoạt động). Trong trường hợp này, ta sẽ chọn A và B ngay từ đầu tiên. Một trong những cách để phản bác lại giả thiết này chính là cho hoạt động A và B trùng lặp với nhiều hoạt động hơn nữa chứ không chỉ hoạt động X. Nghe nó có vẻ không trực quan cho lắm, nhưng (thật không may) ta vẫn có thể xây dựng trường hợp đó như sau:



Nhưng hoạt động được biểu diễn bằng gạch màu xanh chính là những lựa chọn tối ưu trong thời gian biểu trên. Nhưng hoạt động tô màu đỏ trùng lặp với 2 hoạt động nên nó sẽ được chọn trước. Vẫn còn 4 hoạt động thích hợp khác trước hoạt động đỏ, nhưng chúng đều bị trùng lặp lẫn nhau, thế nên ta chỉ có thể lựa chọn thêm 1

hoạt động. Điều tương tự cũng xảy ra đối với 4 hoạt động sau hoạt động màu đỏ, nhưng ta vẫn chỉ có thể chọn 1. Vậy tổng cộng theo phương pháp này, ta vẫn chỉ có thể chọn 3 hoạt động, trong khi kết quả tối ưu là 4.

Tiếp cận 4

Tổng quát lại, ta đã thử 3 cách khác nhau, và mỗi cách đều có thiếu sót:

- Cách 1: Chọn hoạt động có L_i nhỏ nhất
- Cách 2: Chọn hoạt động có $R_i - L_i$ nhỏ nhất
- Cách 3: Chọn hoạt động giao với ít hoạt động khác nhất.

Có vẻ như chúng ta đang đối mặt với một vấn đề hóc búa. Nhưng thật ra, vấn đề này vẫn có một cách giải quyết đẹp đẽ, và không hề phức tạp. Nếu ta xem xét hình trên một cách kỹ lưỡng hơn nữa, ta sẽ thấy được rằng hoạt động màu xanh nằm ở góc trái dưới là hoạt động duy nhất hoàn thành trước đường vẽ dọc màu đỏ. Vậy, nếu lựa chọn 1 hoạt động đơn lẻ, ta sẽ chọn hoạt động kết thúc sớm nhất (có R_i nhỏ nhất). Gọi thời điểm hoàn thành hoạt động đó là t_1 , thì khoảng thời gian sau t_1 sẽ trống để ta có thể chọn các hoạt động khác. Nếu chúng ta chọn bất kỳ hoạt động nào khác, thì khoảng thời gian còn lại sẽ ngắn đi. Điều này là hiển nhiên, bởi vì khi ta kết thúc với bất kỳ một hoạt động nào khác thì luôn luôn $t_2 > t_1$. Trong trường hợp đầu tiên, ta sẽ có toàn bộ thời gian từ t_1 đến khi kết thúc và bao gồm luôn khoảng từ t_2 đến kết thúc. Bởi vậy mà nó cũng không có khuyết điểm trong việc lựa chọn hoạt động kết thúc sớm. Và nó còn có một ưu điểm đó là ta hoàn toàn có thể chèn thêm một hoạt động bất kỳ vào giữa t_1 và t_2 và kết thúc trước khi hoạt động của t_2 bắt đầu.

Trên đây là ý tưởng để chứng minh thuật toán này đúng. Ta có thể từ đó để chứng minh chặt chẽ là cách làm này đúng.

Được biết tới với tên gọi "**Lựa chọn hoạt động**" (**Activity Selection**), đây là bài toán cơ sở sử dụng "**Phương pháp Tham lam**". Giống như là một gã tham lam luôn muốn chiếm lấy nhiều nhất, thường xuyên nhất mà hắn ta có thể, trong trường hợp này, ở mỗi bước ta sẽ chọn lựa một hoạt động kết thúc đầu tiên và mỗi lần đều không có hoạt động đang trong tiến trình. Và có một sự thật đó là ta luôn luôn áp dụng phương pháp tham lam cho mỗi bước trong cuộc đời của mình. Khi ta đi mua sắm hoặc đi xe hơi, ta đều luôn lựa chọn phương án tốt nhất tại thời điểm hiện tại. Thật ra, phương pháp tham lam có 2 công thức chung:

- Tính lựa chọn tối ưu: Từ những kết quả tối ưu cục bộ ta có thể đi đến kết quả tối ưu toàn cục mà không cần phải xem xét lại các kết quả.
- Tính tối ưu từ bài toán nhỏ: Kết quả tối ưu có được xác định bằng các kết quả tối ưu từ bài toán nhỏ hơn.

Đoạn mã giả dưới đây diễn tả cách lựa chọn tối ưu các hoạt động bằng thuật toán tham lam mà ta vừa chứng minh phía trên:

Đặt N là số hoạt động và
 $\{I\}$ là hoạt động thứ I ($1 \leq I \leq N$)

Với mỗi $\{I\}$, xét $S[I]$ và $F[I]$ lần lượt là thời gian bắt đầu và kết thúc của hoạt động.
 Sắp xếp lại các hoạt động theo thứ tự tăng dần của thời gian kết thúc.

- Có nghĩa là, với $I < J$ ta phải có $F[I] \leq F[J]$

```
// A là tập hợp các hoạt động được chọn
A = {1}
// J là hoạt động cuối cùng được chọn
```

```


12  J = 1
13  For I = 2 to N
14      // ta có thể chọn I nếu nó là hoạt động cuối cùng
15      // việc chọn lựa đã hoàn thành
16      If S [I] >= F [J]
17          // lựa chọn hoạt động 'I'
18          A = A + {I}
19
20      // hoạt động 'I' giờ trở thành hoạt động cuối cùng được lựa chọn
21      J = I
22  Endif
23 Endfor
24
    Return A

```

Sau khi áp dụng thuật toán trên, "*Fun plan*" của Johnny sẽ như thế này:

- Sửa phòng
- Vô địch cuộc thi cờ vua
- Chiến thắng cuộc thi Starcraft
- Tham gia kỳ thi SRM trên Topcoder
- Tắm rửa
- Tham gia thử thách "All you can eat" và "All you can drink"

Vấn đề của John Smith đã được giải quyết, tuy nhiên đây chỉ là một ví dụ mà Tham lam có thể hoạt động.

Bài tập tương tự: [Boxing](#) 

BioScore

Bài toán

- Cho N dãy ký tự dài bằng nhau. Mỗi dãy gồm 4 loại ký tự: A, C, T, G .
- Độ tương đồng của 2 dãy là tổng điểm của các cặp ký tự của cùng vị trí. Ví dụ, độ tương đồng của $ACTA$ và $GATC$ là: $\text{score}(A,G) + \text{score}(C,A) + \text{score}(T,T) + \text{score}(A,C)$.

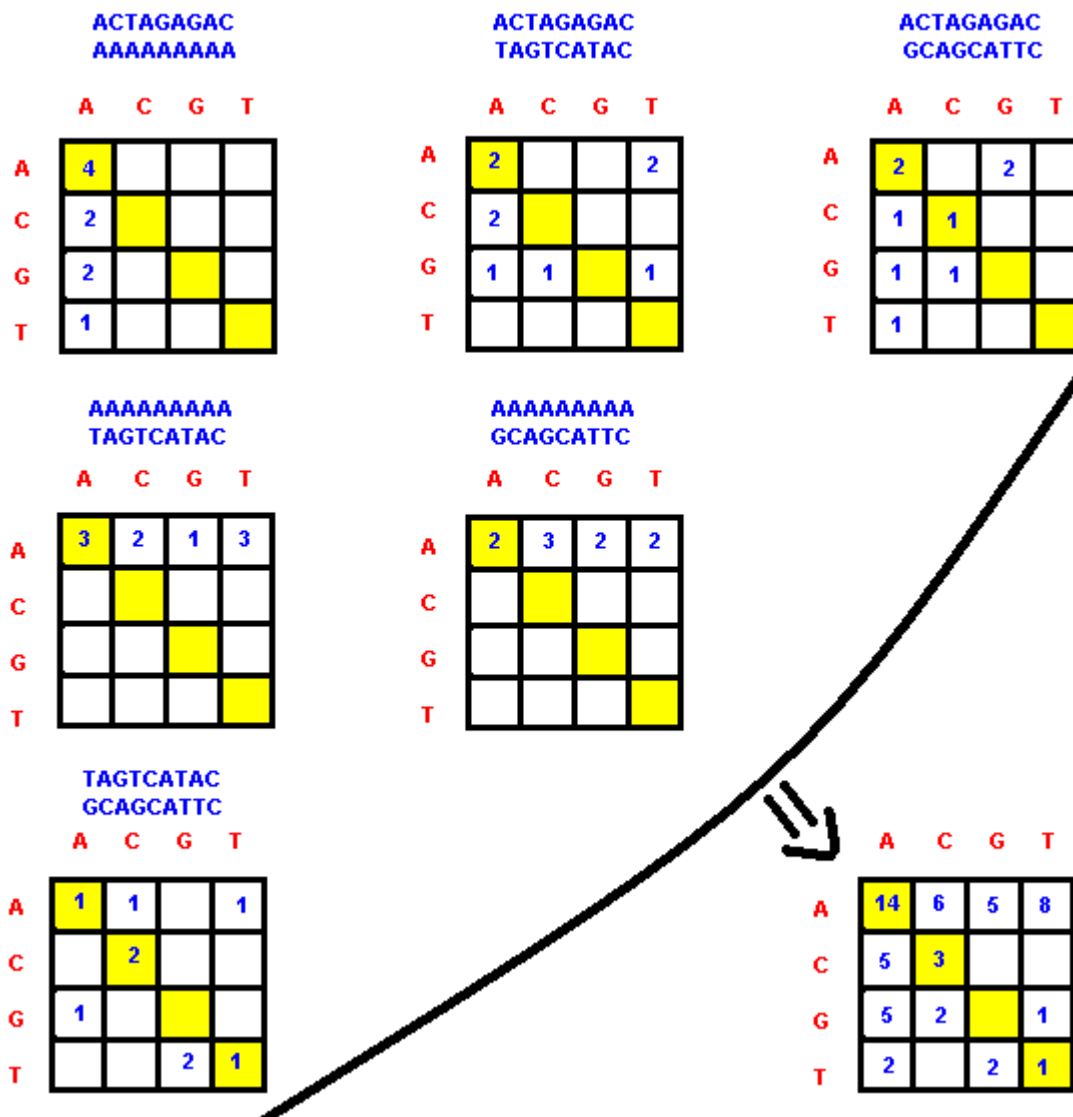
Tìm cách xây dựng bảng S (score) sao cho độ tương đồng của tất cả các cặp 2 xâu trong N xâu là lớn nhất, biết rằng bảng S phải thỏa mãn các tính chất:

- Các giá trị từ -10 đến 10
- Đối xứng: $S(x, y) = S(y, x)$.
- Đường chéo dương: $S(x, x) > 0$.
- Tổng các số trong S phải bằng 0.

Phân tích

Việc đầu tiên mà ta cần làm là xây dựng một ma trận cho biết *số lần lặp* (**ma trận tần số**) của từng cặp 2 ký tự. Đây là một công việc khá nhẹ nhàng khi mà ta chỉ cần ghép từng cặp ký tự ở hai chuỗi tạo rồi đếm số lần xuất hiện của chúng (AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT). Từng cặp ký tự sẽ được xem như một phần tử trong ma trận và giá trị của nó chính là số lần xuất hiện của nó.

Ví dụ, hãy xét bộ {"ACTAGAGAC", "AAAAAAAA", "TAGTCATAC", "GCAGCATTC"} được sử dụng ở ví dụ thứ 2.



Ở góc phải - dưới của hình minh họa trên, ta có thể thấy kết quả của ma trận tần số đối với bộ đã cho. Tạm gọi nó là F . Giờ việc mà ta cần làm là tìm ra một ma trận S sao cho tổng của các tích: $F[i, j] \cdot S[i, j]$ với $(1 \leq i, j \leq 4)$ là lớn nhất.

Giờ ta xét từng điều kiện cho ma trận cần tìm:

1) Tổng của 16 phần tử bằng 0:

Rõ ràng, nếu ta không giới hạn tổng của các phần tử, thì ta có thể đạt được tổng bằng vô cùng, nếu gán tất cả các phần tử của S là vô cùng. Nhưng bởi vì tổng của chúng phải bằng 0, thế nên khi ta tăng giá trị 1 phần tử lên

thì ta sẽ phải giảm đi giá trị của 1 phần tử khác. Thử thách ở điều kiện này chính là phải tìm ra một sự phân bố tối ưu.

2) Giá trị mỗi phần tử chỉ nằm trong khoảng từ -10 đến 10 ($S[i, j] \in [-10, 10]$)

Tương tự, nếu không có điều kiện này, kết quả có thể lên đến vô cùng, nếu ta có 1 cặp không xuất hiện, gán S tương ứng là âm vô cùng, còn các phần tử còn lại trong mảng là dương vô cùng.

3) Mỗi phần tử đối xứng phải có giá trị bằng nhau ($S(x, y) = S(y, x)$)

Bởi vì tính đối xứng, ta phải quy định cho các điểm cho các cặp như **AC** và **CA** bằng nhau. Do tính đối xứng này, ta cập nhật lại mảng tần số xuất hiện để tính cả các cặp đối xứng. Đối với ví dụ trên, ta có bảng tần số mới như sau:

AA: 14	AC+CA: 11	AG+GA: 10	AT+TA: 10
	CC: 4	CG+GC: 2	TC+CT: 0
		GG: 0	GT+TG: 3
			TT: 1

Từ trực giác ta có thể thấy ngay đến phương án như sau: đã sẽ gán điểm số càng cao đối với cặp xuất hiện càng nhiều lần.

4) Các phần tử trên đường chéo phải dương ($S(x, x) > 0$)

Do tính chất này, ta phải xét riêng hai loại phần tử:

- 4 phần tử trên đường chéo (các phần tử đại diện cho **AA**, **CC**, **GG**, **TT**)
- 6 phần tử không nằm trên đường chéo (các phần tử đại diện cho **AC + CA**, **AG + GA**, **AT + TA**, **CG + GC**, **CT + TC**, **GT + TG**).

Ta xét tất cả các trường hợp chọn các phần tử của nhóm thứ nhất. Có $10^4 = 10,000$ trường hợp khác nhau. Với mỗi trường hợp của nhóm thứ nhất, ta sẽ tìm đáp án tối ưu cho nhóm thứ 2.

- Với mỗi trường hợp, ta có tổng các phần tử của nhóm thứ nhất nằm trong khoảng $[4, 40]$, nên tổng các phần tử của nhóm thứ 2 nằm trong khoảng $[-20, -2]$ (chú ý rằng nhóm thứ 2 đối xứng, mỗi phần tử xuất hiện 2 lần trong ma trận S , nên ta phải nhân đôi)
- Vì các phần tử của nhóm 2 có chung tính chất, ta sử dụng Tham lam để tìm kết quả tối ưu cho nhóm thứ 2.

Nhắc lại tư tưởng của Tham lam: tại mỗi bước, ta chọn một lựa chọn tối ưu cục bộ. Trong trường hợp này nghĩa là ta xét lần lượt từng phần tử, với mỗi phần tử, gán cho nó giá trị tối ưu.

Nhận xét:



Nếu ta xét lần lượt các phần tử theo thứ tự giảm dần, ở mỗi bước ta gán giá trị lớn nhất có thể cho phần tử đó, thì kết quả thu được sẽ tối ưu.

Do đó, cách làm là:

- ▶ Sắp xếp 6 phần tử của nhóm 2 theo thứ tự giảm dần (theo ma trận F).
- ▶ Hai giá trị đầu tiên sẽ được gán bằng 10 (vì trường hợp xấu nhất ta cần tổng bằng -20, thì ta vẫn có thể gán 4 giá trị còn lại là -10).
- ▶ Ta biết rằng số điểm cuối cùng sẽ nhỏ hơn 0. Bởi vì ta muốn tối đa hóa số điểm của phần tử thứ 3, nên ba phần tử còn lại sẽ luôn là -10:
 - ▶ Trong trường hợp tốt nhất, tổng bằng -2, ta thu được: $[10, 10, 8, -10, -10, -10]$.
 - ▶ Trong trường hợp xấu nhất, tổng bằng -20, ta thu được: $[10, 10, -10, -10, -10, -10]$.
 - ▶ Ta có thể thu được tất cả các tổng từ -20 đến -2 bằng việc thay đổi giá trị của phần tử thứ 3 từ 8 đến -10.

Giờ ta cần chứng minh rằng phương pháp của mình là đúng.

- ▶ Vì ta xét hết tất cả các trường hợp của nhóm 1, nên ta chỉ cần chứng minh cách chọn các phần tử của nhóm 2 là tối ưu.
- ▶ Vì tổng các phần tử của nhóm 2 cố định, nên nếu ta tăng 1 phần tử, thì phải giảm 1 phần tử khác.
- ▶ Gọi f_1 và f_2 là số lần xuất hiện của 2 số bất kỳ của nhóm 2. Ta có:
 - ▶ $f_1 \setminus * s_1 + f_2 \setminus * s_2 = X$.
- ▶ Không làm mất tính tổng quát, giả sử $f_1 \geq f_2$. Do cách tham, ta có $s_1 \geq s_2$. Ta cũng có s_1 là lớn nhất có thể, nên ta không thể tăng s_1 .
- ▶ Giả sử cách làm của ta không tối ưu, nghĩa là ta có một cách chọn 1 số a sao cho:
 - ▶ $f_1 \setminus * (s_1 - a) + f_2 \setminus * (s_2 + a) = Y$ với a là số dương.
 - ▶ Ta có $Y - X = a * (f_2 - f_1)$. Bởi vì $f_1 \geq f_2$ nên $Y - X$ luôn âm.

Do vậy, cách làm của ta là tối ưu.

Cài đặt

Ý tưởng chính của thuật toán trên sẽ được minh họa trong đoạn mã giả dưới đây:

```

1  Best = -Infinity
2  For S[1] = 1 to 10
3    For S[2] = 1 to 10
4      For S[3] = 1 to 10
5        For S[4] = 1 to 10
6          If (S[1] + S[2] + S[3] + S[4]) mod 2 = 0)
7            S[5] = S[6] = 10
8            S[7] = 10 - (S[1] + S[2] + S[3] + S[4]) / 2
9            S[8] = S[9] = S[10] = -10)
10
11           // biến Best sẽ lưu lại giá trị trung bình lớn nhất tìm được
12           Best = max (Best, score (F, S))
13         Endif
14       Endfor
15     Endfor
16
```



```

17 | Endfor
18 | Endfor
   | Return Best

```

Đối với mảng lưu điểm đã cho (trong trường hợp của chúng ta là mảng S), ta sẽ tính kết quả cuối cùng bằng việc chỉ tính tổng của tích $F[I] \cdot S[I] (1 \leq I \leq 10)$.

GoldMine

Bài toán

Có N mỏ vàng, mỗi mỏ vàng chứa được tối đa 6 công nhân. Bằng cách phân bổ các công nhân, công ty sẽ thu được (hoặc mất đi) số tiền như sau:

- Nếu mỏ vàng có ít nhân công hơn trữ lượng của nó, công ty sẽ thu được 60\$ với mỗi công nhân.
- Nếu mỏ vàng có số công nhân bằng với trữ lượng của nó, công ty sẽ thu được 50\$ cho với mỗi công nhân.
- Nếu mỏ vàng có số công nhân nhiều hơn trữ lượng của nó, công ty sẽ được số tiền là 50 nhân cho trữ lượng mỏ. Với mỗi công nhân bị thừa ra so với sản lượng mỏ, công ty sẽ bị mất đi 20\$.

Dù cho có bị mất tiền đi chăng nữa thì công ty cũng buộc phải phân công đủ các công nhân vào các mỏ.

Phân tích

Bằng việc sử dụng phương pháp tham lam, ta sẽ phân tích xem cách một mỏ vàng bị khai thác triệt. Kinh nghiệm cho thấy, đối với dạng bài như tìm kiếm giá trị tối đa, thường ta có thể giải quyết bằng Tham lam. Trong trường hợp này, mục tiêu của ta là chỉ định những công nhân đến các mỏ vàng sao cho tổng lợi nhuận thu được là tối đa. Phân tích nhanh, ta thấy rằng cần phải biết lợi nhuận thu được từ các mỏ vàng trong tất cả các trường hợp (chỉ có 7 trường hợp - từ 0 đến 6 công nhân). Bảng dưới đây sẽ cho ta thấy lợi nhuận khả thi đối với hai mỏ ở ví dụ 0 trong bài:

	0 người	1 người	2 người	3 người	4 người	5 người	6 người
Mỏ 1	0	57	87	87	67	47	27
Mỏ 2	0	52	66	75	75	66	48

Nếu ta chỉ có duy nhất một công nhân, **lựa chọn tối ưu** chính là cho anh ta vào mỏ nơi mà anh ta mang lại nhiều lợi nhuận nhất. Nhưng nếu ta có nhiều công nhân, ta cần phải kiểm tra xem nếu phân công anh ở mỏ tương tự có mang lại **lợi nhuận cục bộ tối ưu** không.

Trong ví dụ, ta có 4 công nhân cần được phân công. Bảng dưới đây sẽ cho biết lợi nhuận thu được của mỗi mỏ với từng công nhân được thêm vào.

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 1	—	57	30	0	−20	−20	−20

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 2	—	52	14	9	0	−9	−20

Ta để ý rằng, mỏ 1 sẽ tăng thêm 57 nếu ta thêm vào một công nhân, trong khi mỏ 2 chỉ tăng thêm 52. Thế nên, ta sẽ phân bổ người đầu tiên vào mỏ 1.

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 1	—	57	30	0	−20	−20	−20
Mỏ 2	—	52	14	9	0	−9	−20

Giờ, nếu ta thêm công nhân vào mỏ 1, ta chỉ tăng lợi nhuận được thêm 30. Bởi vậy nên ta sẽ thêm công nhân vào mỏ 2, lúc này lợi nhuận ta thu được sẽ tăng thêm 52.

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 1	—	57	30	0	−20	−20	−20
Mỏ 2	—	52	14	9	0	−9	−20

Công nhân thứ 3 sẽ có ích hơn khi làm ở mỏ 1 với lợi nhuận thu được là 30.

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 1	—	57	30	0	−20	−20	−20
Mỏ 2	—	52	14	9	0	−9	−20

Với công nhân thứ 4, ta có thể cho anh ta vào mỏ 1 (với lợi nhuận là 0) hoặc mỏ 2 (với lợi nhuận là 14). Dĩ nhiên, ta sẽ phân công anh ấy vào mỏ hai.

	Ban đầu	Người 1	Người 2	Người 3	Người 4	Người 5	Người 6
Mỏ 1	—	57	30	0	−20	−20	−20
Mỏ 2	—	52	14	9	0	−9	−20

Cuối cùng, hai công nhân còn lại sẽ được phân công bằng cách cho cả hai vào làm ở mỏ 2 hoặc mỗi người làm ở một mỏ riêng. Ví dụ cho ta thấy kết quả mà ta vừa tìm được chính là kết quả tối ưu. Nhưng câu hỏi đặt ra là liệu phương pháp này có luôn đúng hay không?

Khẳng định: Ta luôn luôn thu được tổng lợi nhuận lớn nhất khi lần lượt cho từng công nhân vào mỏ có lợi nhuận cao nhất ở thời điểm hiện tại.

Chứng minh: Gọi A, B lần lượt là mỏ 1 và mỏ 2, $a1, b1, a2, b2$ được định nghĩa như sau:

- $a1$ - lợi nhuận thu được khi phân công thêm một công nhân vào A .
- $a1 + a2$ - lợi nhuận thu được khi phân công thêm hai công nhân vào A .
- $b1$ - lợi nhuận thu được khi phân công thêm một công nhân vào B .
- $b1 + b2$ - lợi nhuận thu được khi phân công thêm hai công nhân vào B .

Thuật toán Tham lam của ta sẽ gia tăng lợi nhuận bằng $a1$ cho công nhân đầu tiên và $(a2 + b1)$ cho công nhân thứ 2. Tổng lợi nhuận lúc này sẽ là $a1 + \max(a2, b1)$. Nếu ban đầu ta chọn $b1$ thì lợi nhuận của công nhân thứ 2 thu được sẽ là $a1$ hoặc $b2$.

Trong trường hợp đầu tiên, ta sẽ có $a1 + b1 \leq a1 + \max(a2, b1)$.

Trong trường hợp thứ hai, tổng lợi nhuận sẽ là $b1 + b2$. Ta cần phải chứng minh $b1 + b2 \leq a1 + \max(a2, b1)$. Mà ta luôn có $b1 \leq b2$ vì lợi nhuận thu được từ việc thêm một công nhân vào một mỏ luôn luôn lớn hơn hoặc bằng lợi nhuận thu được từ việc thêm một công nhân nữa vào mỏ đó.

Trạng thái của mỏ vàng	Lợi nhuận từ việc thêm 1 người	Lợi nhuận từ việc thêm 1 người
Số lượng mỏ số lượng người đào+2	60	60
Số lượng mỏ = số lượng người đào+2	60	50
Số lượng mỏ = số lượng người đào+1	50	-20
Số lượng mỏ số lượng người đào+2	-20	-20

Vì $b1 + b2 \leq a1 + a2 \leq a1 + b1 \leq a1 + \max(a2, b1)$, lựa chọn Tham lam cũng chính là phương án tối ưu.


Cài đặt thuật toán này hoàn toàn không khó, tuy nhiên ta cần phải xử lý thêm một vài trường hợp nữa (tất cả các công nhân đều phải được phân công, chỉ có tối đa sáu người trong một mỏ và nếu một công nhân có thể được đặt tối ưu ở nhiều mỏ, ưu tiên mỏ có chỉ số nhỏ hơn).

WorldPeace 

Bài toán:

Cho n đất nước, mỗi nước có dân số của họ. Hãy chia thành thành các nhóm có k người không có cùng quốc tịch. Hãy cho biết số lượng nhóm tối đa có thể đạt được.

Phân tích:

Những thuật toán Tham lam đều hoạt động tốt ở mọi tình huống bởi ta đã chứng minh được tính đúng đắn của nó. Nhưng còn một lớp bài toán tối ưu hóa nữa mà thuật toán Tham lam có thể được áp dụng. Đây là những bài tập thuộc lớp NP - đầy đủ (như bài toán người đưa thư TSP [Traveling Salesman Problem](#) ) , đối với dạng bài toán này, ta thường sẽ sử dụng phương pháp nhánh cận để giải quyết vấn đề hơn là chờ đợi chương trình thực thi... Lời giải không phải lúc nào cũng là tối ưu, song trong phần lớn trường hợp, nó đã đủ tốt rồi. Với một bài toán không thuộc lớp NP như thế này, đây chính là ví dụ tuyệt vời cho việc một thuật toán Tham lam không chỉ có thể đánh lừa và vượt qua các test mẫu, mà nó còn có thể vượt qua cả những bộ test hệ thống được thiết kế kỹ càng. Thuật toán này không quá khó để nghĩ ra, mà chỉ cần một vài phân tích nhanh ta có thể nhận ra, để tối đa hóa tổng số lượng nhóm, **luôn luôn tối ưu để tạo thiết lập một nhóm từ k quốc gia có dân số đồng nhất**. Chúng ta áp dụng phương pháp này ở từng bước và sau đó sắp xếp lại đoạn để thấy được k quốc gia tiếp theo có dân số đồng nhất. Ý tưởng này sẽ được minh họa trong đoạn mã giả dưới đây:

```

1  Groups = 0
2  Repeat
3    // sắp xếp lại mảng theo thứ tự giảm dần
4    Sort (A)
5    Min = A[K]
6    If Min > 0
7      Groups = Groups + 1
8    For I = 1 to K
9      A[I] = A[I] - 1
10   Endfor
11 Until Min = 0
12 Return Groups

```

Không may thay, mỗi quốc gia lại có tới hàng tỷ người dân, thế nên ta không thể nào thiết lập từng nhóm một được. Về mặt lý thuyết, đối với một tập hợp của k quốc gia, chúng ta sẽ tạo nhóm cho đến khi mà toàn bộ người dân của nước đó đã được phân nhóm. Và điều này sẽ được thực hiện chỉ trong 1 bước:

```

1  Groups = 0
2  Repeat
3    // sắp xếp lại mảng theo thứ tự giảm dần
4    Sort (A)
5    Min= A[K]
6    Groups = Groups + Min
7    For I = 1 to K
8      A[I] = A[I] - Min
9    Endfor
10 Until Min = 0
11 Return Groups

```

Thời gian thực thi giờ không còn là vấn đề nữa, mà vấn đề giờ đây chính là tính đúng đắn của thuật toán! Khi ta thử ví dụ 0 trong bài, thuật toán của chúng ta trả về kết quả là 4 chứ không phải là 5. Nhưng kết quả trả về trong ví dụ 1, 2 và 3 thì đúng. Trong với ví dụ cuối cùng, thay vì tạo ra 3983180234 nhóm, ta chỉ tạo được 3983180207 nhóm mà thôi. Bằng việc chỉ kết quả có sai lệch không đáng kể, ta có thể thấy rằng giải thuật của mình **khá tốt**. Thế nên giờ ta chỉ cần cải tiến nó theo hướng này.

Cho đến hiện tại, ta đã có trong tay hai thuật toán:


- Thuật toán tham lam đầu tiên chính xác, nhưng không đủ nhanh.
- Thuật toán tham lam thứ hai nhanh, nhưng lại không chính xác.

Giờ, điều mà ta cần làm đó chính là tăng độ chính xác của thuật toán này lên nhiều nhất có thể, mà thời gian thực thi vẫn không bị quá giới hạn. Một cách cơ bản, ta đang tìm kiếm sự cân bằng giữa **thời gian thực thi và độ chính xác**. Điểm khác biệt duy nhất giữa hai thuật toán kể trên chính là số lượng nhóm mà chúng ta lựa chọn được. Chúng ta sẽ có một phương án như sau: ta sẽ lựa ra một số lượng lớn nhóm ngẫu nhiên lúc đầu, sau đó sẽ giải quyết đoạn còn lại theo cách tiếp cận an toàn hơn. Khi mà chúng ta chỉ còn lại một số lượng nhỏ người dân chưa được phân nhóm ở các quốc gia, thì lúc này nó hoàn toàn hợp lý khi ta sử dụng phương pháp vét cạn. Với biến **Allowed** được khởi tạo trong thuật toán dưới đây, ta điều khiển số lượng nhóm mà ta mong muốn tạo tại thời điểm được cho.

```

1  Groups = 0
2  Repeat
3  // sắp xếp lại mảng theo thứ tự giảm dần
4    Sort (A)
5    Min = A[K]
6    Allowance = (Min+999) / 1000
7    Groups = Groups + Allowance
8    For I = 1 to K
9      A[I] = A[I] - Allowance
10   Endfor
11 Until Min = 0
12 Return Groups

```

Nếu cách tiếp cận này thật sự đúng, ta hoàn toàn có thể nhận ra được. Mặc dù nó có thể thoát khỏi ánh mắt sắc nhọn của Tomek cũng như là test hệ thống, nhưng có vẻ là nó sẽ không thể nào đưa ra kết quả đúng với mọi bộ test khả thi. Đây chính một ví dụ cho thấy nếu được tinh chỉnh, từ một thuật toán tham lam đơn giản (nhưng vẫn còn khiếm khuyết) cũng cũng có thể trở thành một giải thuật "đúng". Để biết thêm về thuật toán chính xác cho bài này, xem lời giải ở [Match Editorial](#) .


Tổng kết

Tham lam thường dễ nghĩ ra, dễ cài đặt và chạy nhanh, nhưng không phải lúc nào cũng đúng. Khi bạn sử dụng duyệt hoặc quy hoạch động, nó giống như bạn đang di chuyển trên mặt đất an toàn. Còn đối với tham lam, thì giống như bạn đang đi trên một bãi mìn. Như bạn đã thấy qua ví dụ 1, có rất nhiều cách tham khác nhau nhưng chỉ có một cách cho kết quả đúng. Vì vậy, khi làm bài, bạn luôn luôn nên tìm cách chứng minh tính đúng đắn của thuật tham.

Không tồn tại một công thức chung nào cho việc áp dụng Tham lam, tuy nhiên, ta có thể nhìn ra thuật tham bằng việc phân tích các tính chất của bài toán, kinh nghiệm cũng như trực giác.

Một vài lưu ý nhỏ

- Nhưng bài tập mà có vẻ cực kỳ phức tạp (như [TCSocks](#) ) có thể xem như là dấu hiệu để tiếp cận bằng phương pháp Tham lam.

- ▶ Nhưng bài toán mà dữ liệu đầu vào rất lớn (mà kể cả thuật toán có độ phức tạp $O(n^2)$ vẫn không kịp) thường được giải bằng tham lam hơn là quay lui hoặc [quy hoạch động](#) .
- ▶ Mặc dù nó có vẻ rùng rợn, nhưng bạn nên nhìn thuật toán tham lam dưới đôi mắt của một thám tử chứ không phải là dưới cặp kính của một nhà toán học.






Một thám tử giỏi tham lam.



Một người tham lam may mắn.











Một người tham lam không may mắn.






- ▶ Ngoài ra, việc học tập một số thuật toán có sử dụng Tham lam sẽ giúp nắm vững phương pháp này hơn ([thuật toán Prim](#) , [thuật toán Kruskal](#) , [thuật toán Dijkstra](#) )

Bài tập mở rộng





Cấp độ 1

- ▶ [GroceryBagger](#)  – SRM 222
- ▶ [FanFailure](#)  – SRM 195
- ▶ [PlayGame](#)  – SRM 217
- ▶ [SchoolAssembly](#)  – TC004 Round 2
- ▶ [RockStar](#)  – SRM 216
- ▶ [Apothecary](#)  – SRM 204
- ▶ [Boxing](#)  – TC004 Round 3
- ▶ [Unblur](#)  – TC004 Semifinal Room 3

Cấp độ 2

- [Crossroads](#)  – SRM 217
- [TCSocks](#)  – SRM 207
- [HeatDeath](#)  – TC004 Round 4
- [BioScore](#)  – TC004 Semifinal Room 1
- [Rationalization](#)  – SRM 224

Cấp độ 3

- [GoldMine](#)  – SRM 169
- [MLBRecord](#)  – TC004 Round 2
- [RearrangeFurniture](#)  – SRM 220
- [WorldPeace](#)  – SRM 204

Được cung cấp bởi [Wiki.js](#)