CHƯƠNG VI: CHIẾN LƯỢC THAM LAM (GREEDY)

1. Nguyên tắc tham lam

Các thuật toán tham ăn (greedy algorthims) cũng giống như các thuật toán qui hoạch động thường được sử dụng để giải các bài toán tối ưu (tìm nghiệm của bài toán tốt nhất theo một tiêu chí nào đó). Các thuật toán qui hoạch động luôn cho một nghiệm tối ưu.

Các thuật toán tham ăn thực hiện các lựa chọn tối ưu cục bộ với hy vọng các lựa chọn đó sẽ dẫn đến một nghiệm tối ưu toàn cục cho bài toán cần giải quyết. Các thuật toán tham ăn thường rất dễ cài đặt, nhanh (độ phức tạp thời gian thường là hàm tuyến tính hoặc cùng lắm là bậc 2) , dễ gỡ lỗi và sử dụng ít bộ nhớ. Nhưng thật không may là không phải lúc nào chúng cũng cho các lời giải tối ưu.

Qui hoạch động không hiệu quả

Các thuật toán mà chúng ta đã mới học về qui hoạch động chẳng hạn như thuật toán nhân ma trận ($\boldsymbol{O}(N^2)$), thuật toán tìm xâu con dài nhất ($\boldsymbol{O}(mn)$), thuật toán đối với cây nhị phân tối ưu ($\boldsymbol{O}(N^3)$) đều là các thuật toán xét trên một khía cạnh nào đó vẫn có thể cho là không hiệu quả. Tai sao vậy? Câu trả lời là đối với các thuật toán này chúng ta có rất nhiều lựa chọn trong việc tính toán để tìm ra một giải pháp tối ưu và cần phải thực hiện kiểm tra theo kiểu exshauted đối với tất cả các lựa chọn này. Chúng ta mong muốn có một cách nào đó để quyết định xem lựa chọn nào là tốt nhất hoặc ít nhất cũng hạn chế số lượng các lựa chọn mà chúng ta cần phải thử.

Các thuật toán tham ăn (greedy algorithms) được dùng để giải quyết các bài toán mà chúng ta có thể quyết định đâu là lựa chọn tốt nhất.

Thực hiện lựa chọn theo kiểu tham lam (Greedy Choice).

Mỗi khi cần quyết định xem sẽ chọn lựa chọn nào chúng ta sẽ chọn các lựa chọn tốt nhất vào thời điểm hiện tại (Mỗi khi cần chọn chúng ta sẽ chọn một cách tham lam để maximize lợi nhuận của chúng ta). Tất nhiên các lựa chọn như vậy không phải bao giờ cũng dẫn đến một kết quả đúng. Chẳng hạn cho dãy số <3, 4, 5, 17, 7, 8, 9> cần chọn ra dãy con của nó sao cho dãy con đó là một dãy đơn điệu tăng. Dễ dàng thấy kết quả đúng là <3, 4, 5, 7, 8, 9>. Tuy nhiên theo các chọn tham ăn sau khi chọn xong 3 phần tử đầu là 3, 4, 5 sẽ chọn tiếp phần tử 17, phần tử hợp thành một dãy tăng dài hơn đối với các phần tử đã được chọn trước đó và kết quả sẽ là <3, 4, 5, 17>, tất nhiên kết quả này không phải là kết quả đúng.

2. Bài toán đổi tiền

Bài toán phát biểu như sau: có một lượng tiền n đơn vị (đồng) và k đồng tiền mệnh giá lần lượt là m1, m2, .., mk đơn vị. Với giả thiết số lượng các đồng tiền là không hạn chế, hãy đổi n đồng tiền thành các đồn tiền đã cho với số lượng các đồng tiền sử dụng là ít nhất. Một cách đơn giản theo nguyên lý tham ăn chúng ta thấy rằng để đảm bảo số đồng tiền sử dụng là ít nhất chúng ta sẽ cố gắng sử dụng các đồng tiền có mệnh giá lớn nhiều nhất có thể. Ví dụ với 289 đồng và các đồng tiền mệnh giá 1, 5, 10, 25, 100 chúng ta có thể có một cách đổi là 2 đồng 100, 3 đồng 25, 1 đồng 10 và 4 đồng 1.

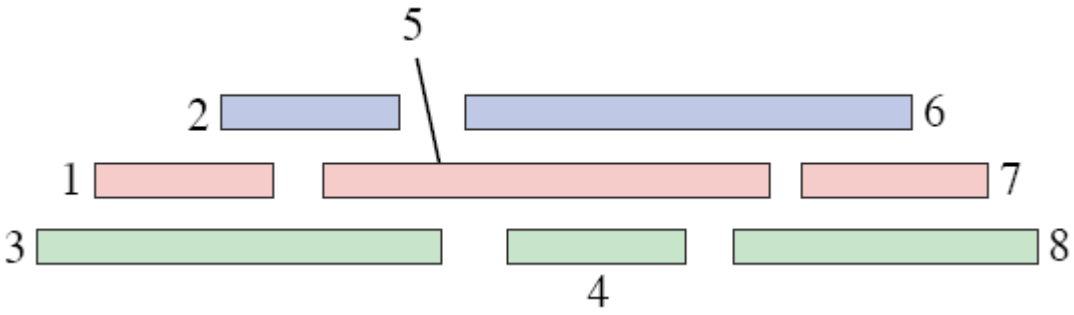
Tuy nhiên trên thực tế bài toán đổi tiền là một bài toán dạng xếp ba lô (knapsack) và thuộc lớp bài toán NP đầy đủ nên thực chất chỉ có 2 cách giải quyết: dùng qui hoạch động với các giá trị nhỏ và duyệt toàn bộ.

3. Bài toán lập lịch

Một phòng học chỉ có thể sử dụng cho một lớp học tại một thời điểm. Có n lớp học muốn sử dụng phòng học, mỗi lớp học có một lịch học được cho bởi một khoảng thời gian $I_j = [s_j, fj]$ có nghĩa là lớp học sẽ học bắt đầu từ thời điểm s_j tới thời điểm f_j. Mục đích của bài toán là tìm một lịch học sao cho số lượng lớp học có thể sử dụng phòng học là lớn nhất có thể được và tất nhiên không có hai lớp nào cùng sử dụng phòng học tại một thời điểm (không có hai lớp trùng lịch học).

Giả sử lịch học của các lớp được sắp theo thứ tự tăng của thời gian kết thúc như sau:

$$f_1 < f_2 < ... < f_n$$



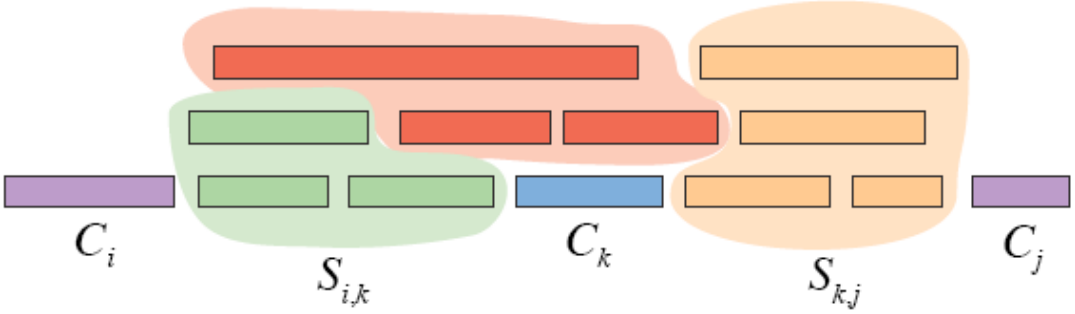
Cấu trúc của một lịch học tối ưu

Gọi $S_{i,j}$ là tập các lớp học bắt đầu sau thời điểm f_i và kết thúc trước thời điểm s_j, có nghĩa là các lớp này có thể được sắp xếp giữa các lớp C_i và C_j.

Chúng ta có thể thêm vào 2 lớp giả định C_0 và C_{n+1} với $f_0 = - \infty$ và $S_{n+1} = +\infty$. Khi đó giá trị $S_{0,n+1}$ sẽ là tập chứa tất cả các lớp.

Giả sử lớp C_k là một phần của lịch học tối ưu của các lớp nằm trong $S_{i,j}$.

Thế thì $i < k < j$ và lịch học tối ưu bao gồm một tập con lớn nhất của $S_{i,k}$, $\{C_k\}$, và một tập con lớn nhất của $S_{k,j}$.



Do đó nếu gọi Q(i, j) là kích thước của một lịch học tối ưu cho tập $S_{i,j}$ chúng ta có công thức sau:

$$Q(i,j) = \begin{cases} 0 & \text{if j = i+1} \\ \max_{i<k<j}(Q(i,k)+Q(k,j)+1) & \text{if j > i+1} \end{cases}$$

Thực hiện một lựa chọn tham lam

Bổ đề 1: Tồn tại một lịch học (thời khóa biểu) tối ưu cho tập $S_{i,j}$ chứa lớp C_k trong $S_{i,j}$ kết thúc đầu tiên, có nghĩa là lớp C_k trong $S_{i,j}$ với giá trị chỉ số k nhỏ nhất.

Bổ đề 2: Nếu chọn lớp C_k như bổ đề 1 tập $S_{i,k}$ sẽ là tập rỗng.

Thuật toán tham lam đệ qui

Recursive-Schedule(*S*)

1 **if** $|S| = 0$

2 **then return**

3 Gọi C_k là lớp có thời gian kết thúc nhỏ nhất trong S

4 Loại bỏ C_k và tất cả các lớp bắt đầu trước thời gian kết thúc của C_k khỏi S;

Gọi S' là tập kết quả

5 O = Recursive-Schedule(S')

6 **return** $O \cup \{C_k\}$

Dựa trên cấu trúc dữ liệu sử dụng để chứa S, thuật toán sẽ có độ phức tạp thời gian là $\boldsymbol{O}(n^2)$ hoặc $\boldsymbol{O}(\text{nlog(n)})$. Thêm vào đó thuật toán sẽ mất một chi phí cho việc bảo trì ngăn xếp vì nó là đệ qui.

Thuật toán tham ăn thời gian tuyến tính theo kiểu lặp

Iterative-Schedule(*S*)

1 $n = |S|$

2 $m = -\infty$

3 $O = \{\}$

3 **for** $i = 1..n$

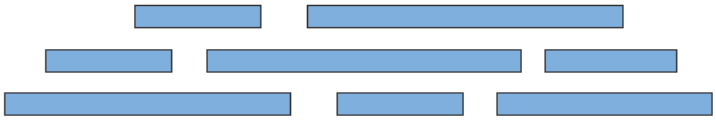
4 **do if** $s_i \geq m$

5 **then** $O = O \cup \{C_i\}$

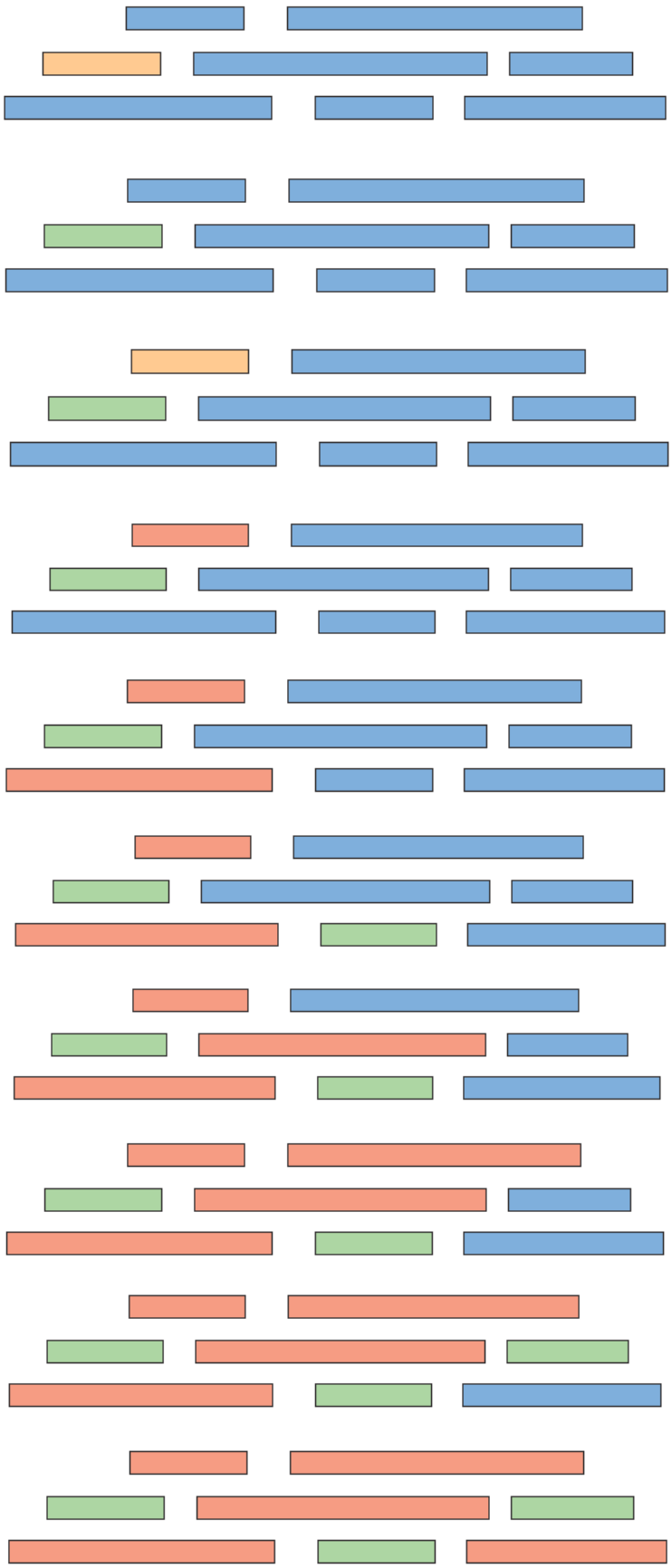
6 $m = f_i$

7 **return** O

Minh họa quá trình thực hiện của thuật toán



:



Thuật toán trên hiển nhiên có thời gian thực hiện tuyến tính và nó là đúng đắn dựa trên bổ đề sau: "Gọi O là tập các lớp hiện tại, và C_k là lớp cuối cùng được thêm vào O. Thế thì đối với lớp C_l bất kỳ mà $l > k$, nếu C_l xung đột với một lớp trong O nó sẽ xung đột với C_k.

Các biến thể của bài toán: Thay vì tìm số lượng lớn nhất các lớp có thể sử dụng phòng học chúng ta có thể thay đổi yêu cầu của bài toán là tìm thời gian lớn nhất mà phòng học được sử dụng. Cách tiếp cận qui hoạch động khi chúng ta thay đổi mục tiêu của bài toán là không đổi nhưng các lựa chọn tham ăn của theo kiểu như trên sẽ không làm việc được đối với bài toán này.

4. So sánh chiến lược tham lam và qui hoạch động

Vậy khi nào thì thuật toán tham ăn sẽ cho nghiệm tối ưu?

Bài toán được giải quyết bằng thuật toán tham ăn (thường là các bài toán tối ưu) nếu như nó có hai đặc điểm sau:

+ Tính lựa chọn tham ăn (greedy choice property): Một nghiệm tối ưu có thể nhận được bằng cách thực hiện các lựa chọn có vẻ như là tốt nhất tại mỗi thời điểm mà không cần quan tâm tới các gợi ý của nó đối với các nghiệm của các bài toán con. Hay nói một cách dễ hiểu là một nghiệm tối ưu của bài toán có thể nhận được bằng cách thực hiện các lựa chọn tối ưu cục bộ.

+ Một nghiệm tối ưu có thể nhận được bằng cách gia tăng (augmenting) các nghiệm thành phần đã được xây dựng với một nghiệm tối ưu của bài toán con còn lại. Có nghĩa là một nghiệm tối ưu sẽ chứa các nghiệm tối ưu đối với các bài toán con kích thước nhỏ hơn. Tính chất này được gọi là cấu trúc con tối ưu (optimal substructure).

Sự khác nhau cơ bản giữa các thuật toán qui hoạch động và các thuật toán tham ăn đó là đối với các thuật toán tham ăn chúng ta không cần biết các nghiệm của các bài toán con để có thể thực hiện một lựa chọn nào đó. Và vì một bài toán có thể có rất nhiều các nghiệm tối ưu khác nhau nên các thuật toán tham ăn có thể hiệu quả hơn các thuật toán qui hoạch động.

Chú ý: Trong một số bài toán nếu xây dựng được hàm chọn thích hợp có thể cho nghiệm tối ưu. Trong nhiều bài toán, thuật toán tham ăn chỉ cho nghiệm gần đúng với nghiệm tối ưu.