Tổng kết kinh nghiệm các bài tập:

Naruto:

- Hướng tiếp cận cho bài toán này là cố gắng **làm giảm số nghiệm thỏa mãn - số lời giải của bài toán:** từ miền 2D -> 1D -> 0D (các đỉnh của miền xác định)
- Đây là một bài toán có thể sử dụng mô hình **Quy hoạch tuyến tính** để giải quyết - một kiến thức cũng rất đặc biệt quan trọng cần nhớ.
- Bài toán có thể được giải bằng việc suy luận các trường hợp a, b (dù không biết gì về quy hoạch tuyến tính), nhưng khi cài đặt **không nên thể hiện điều đó bằng if-else**.
    - Vì rõ ràng rằng hàm Min/Max là một hàm không nghiêm ngặt nên max(f(x1), f(x2)) = max(f(x1), f(x2), f(x1)) = max(f(x1), f(x2), f(x3)) nếu f(x3) luôn <= f(x1)
    - Do đó, một cách an toàn là ta chỉ nên liệt kê ra các nghiệm tối ưu có thể có trong tất cả các trường hợp (các đỉnh) rồi đơn giản đi tính max cho tất cả f(c, k) đó mà thôi.
    - Cài đặt bằng if-else tiềm ẩn nguy cơ sai sót trong cài đặt, hoặc xét trường hợp.

```
a, b, N, X = map(int, input().split())

pairs = [[N,N],[0,X],[X,0],[N,N-X],[N-X,N]]
ans = 0 #Truong hop C = K = 0

for pair in pairs:
    ans = max (ans, a*pair[0] +b*pair[1]) #Xet qua cac truong hop o bien

print(ans)
```

Fairy Tail:

- Đây là một bài toán nhập môn đúng bản chuẩn thuần của phương pháp thiết kế thuật toán tham lam.
- Đơn giản chỉ là nhận ra đặc điểm của hàm mục tiêu từ đó **tìm ra một thứ tự toàn phần - được xem như hàm chọn cục bộ** phù hợp.

```python
n = int(input())
a = []
for i in range(n):
    t, d = map(int, input().split())
    a.append((t, d))
a = sorted(a, key=lambda x: (x[0], -x[1]))
sumTime = 0
ans = 0
for i in range(n):
    sumTime += a[i][0]
    ans += a[i][1] - sumTime
print(ans)
```

Nobita:

- Về cơ bản giống bài Fairy Tail nhưng ở mức độ **khó nhận ra được hàm chọn hơn** - do cần chọn ra một tập hợp con (tổ hợp) thay vì đi xây dựng một thứ tự toàn phần (hoán vị).
- Điều đáng nói duy nhất trong bài tập này có lẽ ở cách làm của bạn **[56]**, cho chúng ta thấy rằng một bài toán có thể giải bằng phương pháp tham lam, hoàn toàn có thể được nhìn nhận dưới góc nhìn của một bài toán Quy hoạch động - dãy con tăng dần.
- **Một bài tập có thể có nhiều phương pháp giải khác nhau theo nhiều góc nhìn khác nhau**.

```python
    global n, res
    n = int(input())
    a =[(0, 0)] * (n + 1)
    for i in range(1, n + 1):
        inp = input().split()
        x, y = map(int, inp)
        a[i] = (x, y)
    a.sort(key=lambda x: x[1])

    def bs(x):
        global n
        l = 1
        r = n
        ans = 0
        while l <= r:
            mid = (l + r) // 2
            if a[mid][1] <= x:
                l = mid + 1
                ans = mid
            else:
                r = mid - 1
        return ans

    for i in range(1, n + 1):
        id = bs(a[i][0])
        maxPrefix[i] = max(maxPrefix[i - 1], maxPrefix[id] + 1)
        res = max(res, maxPrefix[i])
    print(res)
```

- Đối với một người chưa biết về Kỹ thuật chặt nhị phân giải Quy hoạch động Dãy con tăng dần sẽ thật khó để hiểu được bạn [56] đang làm gì và nó giống như là đang dùng "dao mổ trâu đi giết gà"
- Nhưng đối với riêng bạn [56] thì đó lại là cách làm đơn giản vì bạn ấy biết khá nhiều nên sẽ có xu hướng đưa các bài toán mới lạ về các thứ quen thuộc mà mình đã biết, dù cho việc suy luận theo hướng làm mới có thể đơn giản hơn.
- Đây là ví dụ của "**Ưu là nhược - nhược là ưu**"

One Piece:

- Đây là bài tập điển hình cho việc **một bài toán có thể có nhiều phương pháp giải khác nhau** → Nếu cố chấp rằng bài tập tham lam chỉ có thể giải bằng cách tiếp cận tham lam thì chắc chắn sẽ rơi vào bế tắc. → **Close-Mind**

- Tuy nhiên đó là một bài mà thể hiện rất rõ một điều rằng: Một bài tập nhìn vào rất tham lam, nhưng không nên tiếp cận bằng phương pháp tham lam, nhưng rồi lại được giải bằng phương pháp tham lam.
- Chỉ khi chấp nhận thay đổi hướng tiếp cận bài toán, **bài toán tham lam thật sự mới hiện ra - bài toán kiểm tra điều kiện của hàm chặt nhị phân**

DNS:

- Đây là một bài tập không bắt buộc làm, tuy nhiên có khá nhiều điều và kinh nghiệm để nói về bài tập này.
- Trước hết có thể tóm tắt bài toán đơn giản chỉ là một bài toán KNN nhưng với ràng buộc cực kì khổng lồ.

2 Input

- Dòng đầu tiên chứa 3 số nguyên n, K và Q.
- n dòng tiếp theo, mỗi dòng chứa 3 số nguyên x, y, p ($1 <= p <= K$), trong đó (x, y) là tọa độ địa lý của máy chủ thứ i và p là cấp của máy chủ đó.
- Q dòng tiếp theo mỗi dòng chứa 2 số nguyên x và y, (x, y) tọa độ địa lý của các thiết bị đầu cuối tương ứng.

3 Ràng buộc

- $1 <= n <= 10^4$
- $1 <= K <= 5$
- $1 <= Q <= 10^5$
- $0 <= x, y <= 10^9$
- Đảm bảo có đủ K máy có K cấp khác nhau.
- Đảm bảo các điểm dữ liệu được phát sinh ngẫu nhiên có phân bố đồng đều.

4 Output

- Gồm Q dòng, mỗi dòng ghi một **số nguyên duy nhất** tổng thời gian kết nối nhỏ nhất cho thiết bị đầu cuối tương ứng.

5 Notes

- subtask 1: [0, 50) có $N <= 100$
- subtask 2: [50, 80) có $Q <= 10^4$
- subtask 3: [80, 100) không có ràng buộc gì thêm.

- Lời giải cho bài toán:
  - Subtask 1: Duyệt trâu như bình thường
  - Subtask 2: Sử dụng thư viện numpy để tận dụng tính toán song song bằng cách ma trận hóa các điểm thiết bị và các điểm server lại. Khi đó việc tính khoảng cách được rút về việc nhân ma trận bằng numpy thì tốc độ sẽ tăng gấp 10 lần.

Tham khảo hàm fast_dist của sklearn được anh Vũ Hữu Tiệp trình bày trong sách ML cơ bản trong bài KNN để biết thêm chi tiết.

- Subtask 3:
  - Đọc thật kỹ các ràng buộc để thấy có một ràng buộc cực kỳ quan trọng đó là các điểm dữ liệu phân bố ngẫu nhiên đều. Điều này có nghĩa rằng các điểm dữ liệu không được phát sinh một cách đặc biệt.
  - Khi đó đọc lại tư tưởng của phương pháp tham lam: **lấy tối ưu cục bộ để mong đợi tối ưu toàn cục.** Điều này gợi cho ta một cách làm đó là chỉ lấy các điểm server cục bộ - lân cận với điểm thiết bị truy cập chứ không lấy hết toàn bộ các điểm server để đi tính khoảng cách. Tất nhiên, như thế nào được gọi là các điểm cục bộ - lân cận với một điểm ở đây là do ta hoàn toàn tự định nghĩa. → Đây là một **góc nhìn tham lam hoàn toàn mới**, nó khác xa với góc nhìn tham lam cũ theo kiểu cấu trúc con tối ưu và tính lựa chọn tham lam như đã trình bày.
  - Câu hỏi là làm thế nào để định nghĩa các điểm lân cận với một điểm bất kỳ? Có mấy cách sau đây:
    - Một là phân cụm clustering KMeans các điểm server ra thành K cụm, rồi điểm truy cập nằm vào cụm nào thì các điểm thuộc cụm đó được gọi là điểm lân cận. Điều này khá tự nhiên trong thực tế, khi mà một thiết bị truy cập ở HCM thì rõ ràng không nên kết nối với một máy chủ ở HN.
    - Tham khảo thêm chi tiết về kỹ thuật này được viết khoảng 10 dòng trong phần cuối cùng của bài KMeans trong sách ML cơ bản của anh Vũ Hữu Tiệp. Đây là một kỹ thuật được dùng trong các cơ sở dữ liệu Big Data nhằm truy vấn nhanh.
  - Cách lựa chọn trên có lẽ là chuẩn chỉ nhất, nhưng cài đặt KMeans không thư viện thì cũng hơi cực nên có một cách cài đặt tà đạo hơn. Đó là cứ lấy 100 điểm có tọa độ x gần điểm truy cập nhất (sort lại theo x), Tương tự như vậy cho tọa độ y. Đây cũng là một cách phân cụm. Tham khảo bài làm của **[Trần Tuấn Khoa]**

- Thông điệp qua bài tập trên [**Đây là kinh nghiệm từ một người anh đã chỉ lại**]:
  - **Một vấn đề/ lý thuyết có nhiều cách khai thác, cách hiểu khác nhau**. Như Linear Regression có thể hiểu theo kiểu giải tích, đại số, MLE đều được. Thì tham lam cũng có nhiều cách hiểu khác nhau. → Đó là lý do không thể dùng cách nhìn cụ thể để thay thế một cách nhìn khái quát.
  - Hai là: Accuracy trong học thuật thì đó là một chỉ số rất quan trọng, một phương pháp, thuật toán, mô hình mà có có acc 100% chắc chắn tốt hơn cái mà chỉ có 99%. Nhưng trong thực tế công nghiệp điều đó chưa chắc.
  - Nếu không liên quan đến tính mạng con người, tài sản, an ninh quốc gia,... thì đối với khách hàng acc 100% hay 99% cũng chỉ là **perception** - dù bạn có chỉ cho họ đường đi ngắn nhất với 100ms hay đường đi ngắn nhì với 110ms thì họ cũng không cảm nhận được điều khác biệt và họ vẫn nghĩ rằng đó là đường đi ngắn nhất.

Tuy nhiên nếu hệ thống của bạn không tải được số lượng request lớn, dẫn đến lag, sập,.. Thì chắc chắn bạn sẽ bị chửi.

- Thứ ba là mấu chốt để giải bài toán trên bằng một cách khá độc lạ đó là bạn phải để ý đến ngữ cảnh, ràng buộc đặc biệt của bài toán. Điều này có nghĩa là khi giải quyết vấn đề, bạn không nên chỉ dựa vào các công cụ, phương thức, mô hình mạnh. (Tất nhiên việc trang bị các điều trên luôn cần thiết). Mà còn cần biết quan sát, tinh tế, khéo léo, phân tích các **ngữ cảnh, bối cảnh, yếu tố bên ngoài như: đối tượng khách hàng, đặc trưng dữ liệu,... để mà khai thác, lợi dụng sẽ tạo ra được tác động vô cùng to lớn.**

56:

- One Piece:
  - Nhìn một vấn đề - bài toán dưới nhiều khía cạnh góc nhìn khác nhau, nên nhanh chóng có nhiều phương pháp giải khác nhau cho bài toán và tìm được phương án phù hợp, hiệu quả.
  - Tư duy deep thinking nhạy bén, phát hiện vấn đề nhanh chóng.
- Nobita:
  - Biết đa dạng các phương pháp, kỹ thuật và ctdl khác nhau. Biết DP dãy con tăng dần, biết segmentree, biết kỹ thuật chặt nhị phân trong dãy con tăng dần,...
  - Sáng tạo trong cách nhìn một bài toán dưới nhiều phương pháp giải. Đưa một bài toán có thể giải bằng tham lam thành một bài toán DP quen thuộc là Dãy con tăng dần. → Sáng tạo
  - Nhược điểm: Biết quá nhiều nên tự làm khó chính mình.
- Fairy Tail:
  - Phát hiện đặc điểm của bài toán và nhanh chóng tìm ra được một thứ tự các công việc tốt bằng phương pháp tham lam.
- Naruto:
  - Thiết kế thuật toán phù hợp theo từng ràng buộc của bài toán → Tiết kiệm chi phí thời gian.
  - Phát hiện nhanh chóng vấn đề bằng suy luận trường hợp.
  - Nhược điểm:
    - Không đưa bài toán về mô hình quy hoạch tuyến tính
    - Lạm dụng if - else với hàm Min Max không phải là một cách hiệu quả.
- Idea & Solution:
  - Biết đa dạng các *phương pháp* giải, *kỹ thuật*, *ctdl* khác nhau
  - Có sự vận dụng linh hoạt, phù hợp và vô cùng sáng tạo
  - Nhìn một bài toán dưới nhiều góc nhìn khía cạnh khác nhau và chuyển hóa được bài toán đã cho thành bài toán đơn giản hơn
- Design & Coding:
  - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
  - Coding style: Functional + Sequential
- Notes:

    - Highly skilled

20:

- One Piece:
    - Khả năng search and abstraction mạnh
- Nobita:
    - Đọc nhiều tài liệu và biết đa dạng kiến thức
    - Nắm vững cơ sở logic & toán học của vấn đề
- Fairy Tail:
    - Phát hiện đặc điểm của bài toán và nhanh chóng tìm ra được một thứ tự các công việc tốt bằng phương pháp tham lam.
- Naruto:
    - Biết, nắm vững và nhớ rất tốt đa dạng kiến thức
    - Ngay lập tức nhìn ra được vấn đề là một bài toán quy hoạch tuyến tính quen thuộc
- Idea & Solution:
    - Khả năng đọc tài liệu, hiểu và nắm bắt kiến thức, research mạnh
    - Nắm rất chắc các cơ sở logic & toán học
- Design & Coding:
    - Thiết kế coding theo phong cách OOP → respect
- Notes:
    - Research

91:

- One Piece:
    - Sự kiên trì trong việc giải bài khi quyết tâm giải bài với 16 tries để AC
- Nobita + Fairy Tail + Naruto:
    - Khả năng quan sát + trực giác mạnh → giúp giải quyết 3 bài với tốc độ nhanh chóng
- Idea & Solution:
    - Khả năng quan sát, trực giác phát hiện vấn đề nhạy bén
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Kiên trì + Trực giác nhạy bén

66:

- Idea & Solution:
    - AC tất cả các bài với 1 lần nộp duy nhất và thời gian làm rất ngắn → Sự cẩn thận, tỉ mỉ, chắc chắn
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn

    - Coding style: Functional + Sequential
- Notes:
    - Sự cẩn thận, tỉ mỉ, chắc chắn

76:

- Idea & Solution:
    - Gặp một số vấn đề ở một số bài nhưng nhanh chóng tìm ra được cách giải quyết vấn đề. → Gỡ incident (bug) tốt
    - Bài Naruto gặp vấn đề với việc suy luận và xử lý theo trường hợp (if-else) nhưng tìm ra được cách khắc phục hiệu quả
    - Bài Nobita lúc đầu có chút ngộ nhận (hiểu lầm) trong hướng giải nhưng sau đó tìm được cách giải đúng.

- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Quản lý thời gian làm các bài tập một cách hợp lý, trải đều cho từng bài chứ không all-in một lúc
    - Linh hoạt trong việc giải quyết các vấn đề phát sinh

96:

- Idea & Solution:
    - Làm tất cả các bài khá nhẹ nhàng, nhanh chóng và không gặp quá nhiều sự cố.
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
    - Gặp vấn đề trong việc tiếp xúc với nền tảng mới python - gặp bug ở nhập xuất.
- Notes:
    - Giải quyết công việc một cách nhẹ nhàng

51:

- Idea & Solution:
    - Làm tất cả các bài khá nhẹ nhàng, nhanh chóng và không gặp quá nhiều sự cố.
    - Gặp vấn đề trong việc ngộ nhận cách làm cho bài One Piece (làm theo hướng tham lam) nhưng sau đó đã nhận ra được chân lý rằng một bài toán có thể giải bằng nhiều phương pháp khác nhau và nhanh chóng tìm ra được phương pháp phù hợp.
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Giải quyết công việc một cách nhẹ nhàng

17:

- Idea & Solution:
  - Giải quyết công việc một cách nhanh chóng và chắc chắn
  - Khi gặp vấn đề ở bài One Piece thì nhanh chóng tìm được giải pháp thay thế hiệu quả hơn
- Design & Coding:
  - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
  - Coding style: Functional + Sequential
- Notes:
  - Chắc chắn, nhanh chóng và linh hoạt

92:

- Idea & Solution:
  - Gặp vấn đề trong việc suy luận theo trường hợp và xử lý if-else ở bài Naruto nhưng nhanh chóng tìm ra cách khắc phục rất hiệu quả và hợp lý.
  - Phát hiện nhanh chóng được các đặc trưng đặc biệt của bài toán
- Design & Coding:
  - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
  - Coding style: Functional + Sequential
  - Gặp vấn đề trong việc sử dụng ngôn ngữ lập trình (c++) nhưng không khó khăn để khắc phục (python)
- Notes:
  - Phát hiện nhanh chóng được các đặc trưng đặc biệt của bài toán
  - Giải quyết vấn đề phát sinh một cách linh hoạt và hiệu quả

90:

- Idea & Solution:
  - Gặp vấn đề trong việc ngộ nhận cách làm cho bài One Piece (làm theo hướng tham lam) nhưng sau đó đã nhận ra được chân lý rằng một bài toán có thể giải bằng nhiều phương pháp khác nhau và nhanh chóng tìm ra được phương pháp phù hợp.
- Design & Coding:
  - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
  - Coding style: Functional + Sequential
- Notes:
  - Chia bài tập cho đồng đội một cách đồng đều và hiệu quả - teamwork → respect

33:

- Idea & Solution:
  - Gặp một chút vấn đề trong do không cẩn thận dò lại code trước khi nộp bài nhưng không đáng kể.
- Design & Coding:
  - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn

    - Coding style: Functional + Sequential
- Notes:
    - Chia bài tập cho đồng đội một cách đồng đều và hiệu quả - teamwork → respect

61:

- Idea & Solution:
    - Gặp vấn đề trong việc ngộ nhận rằng bài tập tham lam thì phải giải bằng phương pháp tham lam
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Chia bài tập cho đồng đội một cách đồng đều - teamwork → respect

93:

- Idea & Solution:
    - Giải quyết cả 4 bài một cách nhanh chóng
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Ngoài việc nộp muộn ra thì mọi thứ còn lại đều ổn cả.

69:

- Idea & Solution:
    - Giải quyết cả 3 bài một cách nhanh chóng
    - Gặp vấn đề trục trặc trong việc suy luận theo trường hợp và xử lý if-else ở bài tập Naruto nhưng vẫn kiên trì giải quyết đến cùng với 13 lần tries.
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Kiên trì
    - Làm bài tập DNS

90:

- Idea & Solution:
    - Giải quyết cả 3 bài một cách nhanh chóng
    - Gặp vấn đề trong việc tìm kiếm giải pháp phù hợp cho bài One Piece và cuối cùng đã tìm ra.
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:

    - Kiên trì
    - Làm bài tập DNS

04:

- Idea & Solution:
    - Giải quyết cả 4 bài một cách nhanh chóng
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Chú ý deadline cẩn thận hơn là được

92:

- Idea & Solution:
    - Có tư duy "độc lạ" khi là người duy nhất nộp đúng cho bài tập DNS ngay từ lần submit đầu tiên
    - Bài DNS là bài cần tư duy "lạ" khi không tư duy theo lối thông thường
- Design & Coding:
    - Thiết kế thuật toán đơn giản, dễ hiểu, ngắn gọn
    - Coding style: Functional + Sequential
- Notes:
    - Khó hiểu khi không lựa chọn làm 4 bài còn lại mà lựa chọn làm bài khó nhất và làm đúng được bài đó.