- Ứng dụng bài toán trong thực tế

- Sử dụng thuật toán gì để xây dựng.

- Ứng dụng thuật toán đấy vào trong xây dựng.

- Độ phức tạp của thuật toán.

Bài 1:

- Ứng dụng thực tế:

+ Quản lí ưu tiên các mục gợi ý.

+ Tìm kiếm các giá trị tối ưu hoặc để quản lý các hàng đợi ưu tiên của các yếu tố trong quá trình tính toán.

- Sử dụng:

+ Cấu trúc vun đống nhị phân (binary heap). Heap nhị phân là một cây nhị phân hoàn chỉnh, nơi mỗi nút cha có giá trị lớn hơn tất cả các nút con của nó.

+ Thuật toán swim, sink hỗ trợ xây dựng giúp duy trì heap độ phức tạp là O(logN) mỗi khi thêm hay xóa phần tử.

- Độ phức tạp chủ yếu là: O(logN).

Bài 2: Tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t trên đồ thị có hướng, trọng số không âm.

- Ứng dụng thực tế:

+ Hệ thống định vị điều hướng gg map, mạng máy tính, viễn thông trong việc giảm độ trễ tăng băng thông.

- Sử dụng: phương pháp tham lam

+ Tham lam ở chỗ chọn đỉnh có khoảng cách ngắn nhất với đỉnh nguồn và cập nhật khoảng cách ngắn nhất bằng pp relax().

+ Hàng đợi ưu tiên (IndexMaxPQ): chọn đỉnh có khoảng cách ngắn nhất từ đỉnh nguồn trong mỗi bước của thuật toán, dựa trên đống (heap) nhị phân, cho phép chèn, xóa và giảm giá trị khóa với độ phức tạp O(logN).

- Độ phức tạp: O((V+E)log V)

+ Lặp V lần để xóa đỉnh có khoảng cách ngắn nhất từ hàng đợi ưu tiên, mỗi lần O(log V): tổng O(V log V).

+ Mỗi cạnh được thư giãn một lần. Thư giãn mỗi cạnh có thể bao gồm thao tác chèn hoặc cập nhật giá trị trong hàng đợi ưu tiên, mỗi lần O(log V). Tổng cho tất cả các cạnh là O(E log V).

Bài 3: Tìm đường đi ngắn nhất giữa hai đỉnh bất kì.

Bài 4: Cài đặt cây khung nhỏ nhất PrimMST không có chu trình và liên thông.

- Ứng dụng thực tế:

+ Thiết kế mạng lưới dây diện với tổng chi phí thấp nhất.

+ Thiết kế mạng máy tính với chi phí thấp nhất

- Sử dụng: phương pháp tham lam

+ Hàng đợi ưu tiên với chỉ số (IndexMinPQ) để quản lý và lấy ra đỉnh có trọng số nhỏ nhất một cách hiệu quả.

+ Phương pháp tham lam (Greedy Method) để xây dựng cây khung nhỏ nhất, đảm bảo mỗi lần chỉ thêm cạnh có trọng số nhỏ nhất kết nối một đỉnh mới vào cây hiện tại.

- Độ phức tạp: O(E log V)

+ Trong mỗi lần quét một đỉnh, thuật toán cần quét qua tất cả các cạnh kề của đỉnh đó và thực hiện các thao tác như cập nhật trọng số và cạnh nối của các đỉnh kề. Mỗi cạnh được quét một lần, do đó tổng số lần quét cạnh trong toàn bộ thuật toán là \(O(E)\).

+ Trong quá trình này, thuật toán sử dụng một hàng đợi ưu tiên để lựa chọn đỉnh có trọng số nhỏ nhất để thêm vào cây khung. Mỗi lần thêm hoặc cập nhật đỉnh trong hàng đợi ưu tiên mất (O(log V)) thời gian.

Bài 5: Bellman-Ford đích tìm đường đi ngắn nhất từ mỗi đỉnh đến đích đồ thị có thể có trọng số âm

- Ứng dụng thực tế:

+ Hệ thống giao thông: Trong các hệ thống giao thông, Bellman-Ford có thể được sử dụng để tính toán đường đi ngắn nhất giữa các điểm đến và điểm đi trong một mạng lưới đường.

+ Tính toán đường đi nhất quán nhất trong các hệ thống phân phối điện: Trong hệ thống phân phối điện, việc tính toán đường dây điện ngắn nhất từ trạm điện đến các địa điểm tiêu thụ là quan trọng để tối ưu hóa việc phân phối điện và tránh quá tải.

- Sử dụng: phuong pháp quy hoạch động

+ Giải các bài toán nhỏ trước, chồng chéo nhau, lưu kết quả lại để trợ giúp giải các bài toán lớn hơn dần và cho đến bài toán ban đầu.

+ Thuật toán Bellman-Ford chạy duyệt tối đa n – 1 bước (vì một chu đường đi n cạnh qua n đỉnh sẽ có một chu trình, có thể loại chu trình đó ra), mỗi bước tương ứng với số cạnh được sử dụng cho các đường đi đang tìm. Ở mỗi bước chúng ta xét đỉnh v, mà d(v) ở bước trước đó được cập nhật

- Độ phức tạp O(VE):

+ Số lần lặp: Thuật toán lặp qua mỗi cạnh trong đồ thị, và cố gắng cập nhật khoảng cách từ đỉnh nguồn đến các đỉnh kề. Điều này được thực hiện qua \( V \) lần lặp.

+ Số lần cập nhật: Trong mỗi lần lặp, thuật toán cố gắng cập nhật khoảng cách từ đỉnh nguồn đến các đỉnh kề. Mỗi cạnh sẽ được kiểm tra một lần. Vậy có tổng cộng \( E \) lần cập nhật.

Bài 6: Đóng gói balo

Định nghĩa: Gọi OPT(i, w) là giá trị lớn nhất của tập con các đồ vật tỏng các đồ vật 1, 2, …, i với tổng trọng lượng không quá w.

Trường hợp 1: OPT(i, w) không chứa đồ vật thứ i. Khi đó

OPT(i, w) chọn các đồ vật trong số {1, 2, …, i-1} với giới hạn khối lượng là w.

Trường hợp 2: OPT(i, w) chọn đồ vật thứ i. Khi đó

Tính giới hạn khối lượng trừ đồ vật i: w’ = w – wi

OPT(i, w) chọn đồ vật i và chọn các đồ vật trong số {1, 2, …, i-1} với giới hạn khối lượng là w’ = w - wi. Do đó ta có công thức truy hồi sau



- Ứng dụng của thuật toán knapsack:

+ tối ưu hóa sự phân phối tài nguyên như thời gian, nguồn lực, và chi phí trong quản lí dự án.

- Sử dụng phương pháp quy hoạch động:

+ (giải các bài toán tương tự từ nhỏ đến lớn và lưu nhớ lại kết quả để dùng cho các bài toán lớn mà không phải tính lại).

+ Các bài toán nhỏ được tăng dần theo số đồ vật được chọn {0, 1, 2, …, i} và giới hạn khối lượng cho phép w cũng tăng từ 1 đến W.

- Độ phức tạp của thuật toán: O(NW) với n là số lượng đồ vật, w là số lượng dung tích giới hạn

+ Lí do cho độ phức tạp này là do thuật toán sử dụng một ma trận hai chiều \( opt \) có kích thước \( (N+1) \times (W+1) \) để lưu trữ các giá trị tối ưu tại mỗi bước lặp của thuật toán. Mỗi ô trong ma trận này cần được cập nhật một lần, và có \( N \times W \) ô trong ma trận, do đó độ phức tạp là \( O(N \times W) \).

Bài 7: Nqueen backtracking

Backtracking Algorithm

- Ý tưởng là đặt các quân hậu lần lượt từng quân ở các cột khác nhau, bắt đầu từ cột bên trái nhất. Khi chúng ta đã đặt một quân hậu ở một cột, chúng ta kiểm tra xung đột với các quân hậu đã được đặt. Trong cột hiện tại, nếu chúng ta tìm được hàng mà không có xung đột, chúng ta đánh dấu hàng và cột này như một phần của lời giải đang tìm. Nếu chúng ta không tìm thấy hàng vì có các xung đột, thì chúng ta phải quay lại và trả về false.

1. Bắt đầu cột trái nhất

2. Nếu mọi hậu đã được đặt, trả về true

3. Thử mọi hàng trong cột hiện tại

Thực hiện điều sau cho mỗi hàng đang thử

a. Nếu hậu có thể đặt an toàn ở hàng đó, thì đánh dấu [hàng, cột] như một phần của lời giải và đệ qui kiểm tra nếu đặt hậu ở đây có dẫn đến một lời giải không.

b. Nếu đặt hậu ở [hàng, cột] dẫn đến một lời giải thì trả về true

c. Nếu đặt hậu ở đó không dẫn đến một lời giải thì không đánh dấu [hàng, cột] này (quay lui) và trở về bước a. để thử hàng khác.

4. Nếu mọi hàng đã được thử và không có gì tiến triển tiếp, trả về false để kích hoạt quay lui.

- Độ phức tạp: O(n!)

+ Với mỗi cột, có \(N\) hàng để thử đặt nữ hoàng, do đó số lần kiểm tra sẽ tăng một cách mũi nhọn khi kích thước của bàn cờ \(N\) tăng lên. Điều này dẫn đến độ phức tạp \(O(N!)\), với mỗi bước đệ quy tăng lên theo cấp số nhân, khiến cho thuật toán trở nên rất chậm khi \(N\) lớn.

Bài 15: FordFukerson

- Ứng dụng thực tế:

+ Luồng Mạng: Thuật toán Ford-Fulkerson được sử dụng để tìm luồng tối ưu trong mạng dẫn (đồ thị có hướng có trọng số). Điều này có thể áp dụng trong các hệ thống vận chuyển hàng hóa, mạng điện, mạng dữ liệu, và nhiều ứng dụng khác.

+ Định tuyến mạng: Trong mạng máy tính và mạng điện thoại di động, thuật toán Ford-Fulkerson có thể được sử dụng để tối ưu hóa việc định tuyến dữ liệu trên các đường truyền mạng.

- Sử dụng:

 **Cấu trúc dữ liệu FlowNetwork**: Là đồ thị có hướng có trọng số, trong đó các đỉnh đại diện cho các điểm kết nối, các cạnh đại diện cho các kênh hoặc đường truyền dữ liệu giữa các đỉnh, và trọng số của mỗi cạnh biểu thị dung lượng của kênh đó.

 **Cấu trúc dữ liệu FlowEdge**: Là một cạnh trong đồ thị dòng, chứa thông tin về đỉnh bắt đầu, đỉnh kết thúc, dung lượng của kênh, và dòng dữ liệu hiện tại trên kênh đó.

 **Cấu trúc dữ liệu Queue**: Được sử dụng để thực hiện tìm kiếm theo chiều rộng trong quá trình tìm kiếm đường đi bổ sung.

- Độ phức tạp : O(E \* f)

+ Khi khả năng thông qua là các số tự nhiên, thời gian chạy của thuật toán Ford-Fulkerson bị chặn bởi O(E\*f), trong đó E là số cạnh của đồ thị và f là luồng cực đại trên đồ thị. Điều này là bởi vì mỗi đường tăng được tìm ra trong thời gian O(E) và nó làm tăng luồng với một lượng có giá trị nguyên không nhỏ hơn 1.

Bài 17: Inversions đếm số nghịch thế

- Ứng dụng thực tế:

+ Lý thuyết bầu cử.

+ Lọc cộng tác.

+Đo lường mức độ sắp xếp của một mảng.

+Phân tích độ nhạy của chức năng xếp hạng của Google.

+Tổng hợp thứ hạng để tìm kiếm meta trên Web.

- Sử dụng phương pháp chia để trị

+ Chia để trị: Chia bài toán thành các bài toán con độc lập, giải mỗi bài toán con cũng bằng thuật toán đó và kết hợp các lời giải cúa các bài toán con thành lời giải của bài toán ban đầu.

1. \*\*Chia và Trị (Divide and Conquer)\*\*:

- \*\*Chia (Divide)\*\*: Mảng được chia thành hai nửa.

- \*\*Trị (Conquer)\*\*: Đếm số nghịch thế trong từng nửa một cách đệ quy.

- \*\*Kết hợp (Combine)\*\*: Đếm số nghịch thế giữa hai nửa và hợp nhất hai nửa lại thành một mảng đã sắp xếp.

2. \*\*Đếm Nghịch Thế Trong Quá Trình Merge\*\*:

- Khi hợp nhất hai nửa đã được sắp xếp, nếu một phần tử ở nửa phải nhỏ hơn một phần tử ở nửa trái, thì tất cả các phần tử còn lại ở nửa trái sẽ tạo thành một nghịch thế với phần tử này.

- Độ phức tạp của thuật toán: O(nlogN)

#### 1. Độ Phức Tạp của Việc Chia Mảng

- Mỗi lần chia, mảng được chia đôi. Việc chia này diễn ra logarit theo cơ số 2, do đó có \( \log n \) lần chia.

#### 2. Độ Phức Tạp của Việc Kết Hợp Mảng

- Mỗi lần kết hợp hai mảng con có tổng số phần tử là \(n\), việc này tốn \(O(n)\) thời gian.

- Đối với mỗi cấp độ của đệ quy, tổng thời gian để kết hợp các mảng con lại với nhau vẫn là \(O(n)\).

### Tổng Độ Phức Tạp

- Mỗi lần kết hợp tốn \(O(n)\) thời gian.

- Có \( \log n \) lần chia (cấp độ của đệ quy)

Bài 20: Tìm tập con tối đa cho công việc không bị chồng chéo

- Sử dụng phương pháp tham lam sắp xếp theo thời gian kết thúc sớm nhất.

+ \*\*Tối ưu cục bộ\*\*: Mỗi bước, chúng ta chọn công việc tốt nhất (kết thúc sớm nhất) có thể để tối ưu số lượng công việc được chọn trong thời gian còn lại.

+ \*\*Tối ưu toàn cục\*\*: Việc chọn công việc kết thúc sớm nhất tạo điều kiện tối đa để chọn thêm các công việc khác sau đó, dẫn đến việc chọn được nhiều công việc nhất có thể.

- Độ phức tạp là O(nlogn) với n là số lượng công việc. Độ phức tạp như vậy do sắp xếp công việc theo thời gian kết thúc.

Bài 21: tìm tập con các công việc không xung đột sao cho tổng lợi nhuận là lớn nhất. Mỗi công việc có thời gian bắt đầu, thời gian kết thúc và lợi nhuận.

- Sử dụng phương pháp quy hoạch động được sử dụng để tối ưu hóa tối đa trong việc lựa chọn công việc. Cụ thể, bài toán này được giải quyết bằng cách xây dựng một bảng (mảng) để lưu trữ lợi nhuận tối đa có thể đạt được khi xét đến từng công việc.

+ Sắp xếp theo thời gian kết thúc thuận tiện cho việc áp dụng quy hoạch động.

+ \*\*Khởi tạo mảng lưu trữ lợi nhuận tối đa\*\*:

- Tạo mảng `maxProfit` để lưu trữ lợi nhuận tối đa có thể đạt được khi xét đến từng công việc.

- Tạo mảng `ans` để lưu các công việc cụ thể tương ứng với lợi nhuận tối đa.

+ \*\*Tính toán lợi nhuận tối đa cho từng công việc\*\*:

- Duyệt qua từng công việc và tính toán lợi nhuận tối đa khi chọn hoặc không chọn công việc đó.

- Sử dụng hàm `findLastNonConflictingJob` để tìm công việc cuối cùng không xung đột với công việc hiện tại.

- Độ phức tạp của thuật toán:

- \*\*Sắp xếp công việc\*\*: Độ phức tạp của việc sắp xếp các công việc theo thời gian kết thúc là \(O(n \log n)\).

- \*\*Tính toán lợi nhuận tối đa\*\*: Đối với mỗi công việc, việc tìm kiếm công việc không xung đột cuối cùng sử dụng tìm kiếm nhị phân có độ phức tạp là \(O(\log n)\). Tổng cộng cho \(n\) công việc, độ phức tạp là \(O(n \log n)\).

Do đó, tổng độ phức tạp của thuật toán là \(O(n \log n)\).