TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

**KHOA ĐIỆN TỬ**

**BỘ MÔN**. **CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN**

**TOÁN RỜI RẠC**

Sinh viên: Nguyễn Thế Dương

Lớp: K58KMT.K01

Giáo viên giảng dạy: **Ths.Đỗ Duy Cốp**

**Thái Nguyên – 2024**

|  |  |
| --- | --- |
| **TRƯỜNG ĐHKTCN** | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM** |
| **KHOA ĐIỆN TỬ** | ***Độc lập - Tự do - Hạnh phúc*** |

**BÀI TẬP LỚN**

**MÔN HỌC**: TOÁN RỜI RẠC

BỘ MÔN CÔNG NGHỆ THÔNG TIN

*Sinh viên:* *Nguyễn Thế Dương*

*Lớp*: *K58KMT.K01*  *Ngành: Kỹ thuật máy tính*

*Giáo viên hướng dẫn:* *Ths. Đỗ Duy Cốp*

*Ngày giao đề 07/03/2024*  *Ngày hoàn thành* *14/03/2024*

*Tên đề tài*

*Yêu cầu* *Mỗi sinh viên làm bài riêng, gồm có 7 phần, mỗi phần làm 1 bài toán. Cụ thể mỗi sinh viên làm 7 bài, theo danh sách đã phân công. Mỗi sinh viên làm riêng, in quyển báo cáo*.

|  |
| --- |
| **GIÁO VIÊN HƯỚNG DẪN** |
| *(Ký và ghi rõ họ tên)* |

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

### Thái Nguyên, ngày….tháng…..năm 20....

## GIÁO VIÊN HƯỚNG DẪN

*(Ký ghi rõ họ tên)*

**MỤC LỤC**

1. PHẦN MỞ ĐẦU
2. Giới thiệu thông tin cá nhân
3. Hướng dẫn làm bài
4. PHẦN NỘI DUNG
5. Bài toán đếm
6. Bài toán tồn tại
7. Bài toán liệt kê
8. Bài toán tối ưu
9. Thuật toán tìm kiếm: hãy tìm hiểu thuật toán và cài đặt bằng JS
10. Đồ thị: Tìm cây khung nhỏ nhất bằng thuật toán
11. Đồ thị: Tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh còn lại
12. PHẦN TỔNG KẾT
13. Sau khi học xong nhận được kiến thức gì?
14. Upload mã nguồn lên GitHub
15. **PHẦN MỞ ĐẦU**
16. **Giới thiệu thông tin cá nhân**

Em tên là: Nguyễn Thế Dương

Ngày sinh: 09/11/2004

Quê quán: Thái nguyên

Chức vụ: Sinh Viên

Lớp: K58KMT.K01

Sinh viên năm thứ 2

Ngành học: Kỹ thuật máy tính

1. **Hướng dẫn làm bài**

Với mỗi bài toán được phân công, cần thực hiện các bước sau (lặp lại cho 7 bài):

1. Trình bày tên bài toán

2. Phân tích bài toán

3. Lập trình giải quyết bài toán bằng JavaScript

4. Chụp lại kết quả

5. Đánh giá kết quảDưới đây là số thứ tự đề bài được giao:



1. **PHẦN NỘI DUNG**
2. **Bài toán đếm**
   1. Tên bài toán : Một quán cà phê có 4 loại bánh ngọt, 3 loại nước giải khát và 2 loại kem. Bạn muốn chọn một phần tráng miệng gồm một bánh, một loại nước và một viên kem. Có bao nhiêu cách chọn?

* Để chọn một phần tráng miệng gồm một bánh, một loại nước và một viên kem, ta cần thực hiện 3 bước chọn liên tiếp:

**Bước 1:** Chọn một loại bánh ngọt. Có 4 loại bánh ngọt, nên có 4 cách chọn.

**Bước 2:** Chọn một loại nước giải khát. Có 3 loại nước giải khát, nên có 3 cách chọn.

**Bước 3:** Chọn một viên kem. Có 2 loại kem, nên có 2 cách chọn.

**Theo quy tắc nhân, số cách chọn một phần tráng miệng là:**

4 cách chọn bánh \* 3 cách chọn nước \* 2 cách chọn kem = **24 cách chọn**.

**Vậy có 24 cách chọn một phần tráng miệng gồm một bánh, một loại nước và một viên kem.**

* 1. **: Phân tích bài toán**

**Bài toán:** Một quán cà phê có 4 loại bánh ngọt, 3 loại nước giải khát và 2 loại kem. Bạn muốn chọn một phần tráng miệng gồm một bánh, một loại nước và một viên kem. Có bao nhiêu cách chọn?

**Phân tích:**

* **Dạng bài toán:** Bài toán thuộc dạng **đếm số cách chọn** trong các trường hợp **liên tiếp** và **không phụ thuộc lẫn nhau**.
* **Yêu cầu:** Tìm số cách chọn một phần tráng miệng gồm 3 món: bánh, nước và kem.
* **Điều kiện:**
  + Có 4 loại bánh ngọt.
  + Có 3 loại nước giải khát.
  + Có 2 loại kem.
  + Thứ tự chọn các món không quan trọng.
* **Phương pháp giải:**
  + Sử dụng **quy tắc nhân** để tính số cách chọn.
  + **Bước 1:** Chọn 1 bánh trong 4 loại bánh, có 4 cách chọn.
  + **Bước 2:** Chọn 1 loại nước trong 3 loại nước, có 3 cách chọn.
  + **Bước 3:** Chọn 1 viên kem trong 2 loại kem, có 2 cách chọn.
  + **Số cách chọn** một phần tráng miệng là: 4 cách chọn bánh \* 3 cách chọn nước \* 2 cách chọn kem = **24 cách chọn**.

**Kết luận:**

Có 24 cách chọn một phần tráng miệng gồm một bánh, một loại nước và một viên kem.

*1.3*: Lập trình giải quyết bài toán bằng JavaScript

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title> Cách chọn món tráng miệng</title>

</head>

<body>

    <h1>Chọn món tráng miệng</h1>

    <form action="index.html" method="post">

        <label for="cake">Bánh ngọt:</label>

        <select name="cake" id="cake">

            <option value="mochi">Bánh mochi </option>

            <option value="Dango">Bánh Dango</option>

            <option value="manju">Bánh manju</option>

            <option value="Daifuku">Bánh Daifuku</option>

        </select>

        <br>

        <label for="drink">Nước giải khát:</label>

        <select name="drink" id="drink">

            <option value="red">Nước sting đỏ</option>

            <option value="yellow">Nước sitng vàng </option>

            <option value="green">Nước sting xanh</option>

        </select>

        <br>

        <label for="ice\_cream">Kem:</label>

        <select name="ice\_cream" id="ice\_cream">

            <option value="vanilla">Kem </option>

            <option value="chocolate">Kem sô cô la</option>

        </select>

        <br>

        <br>

        <input type="submit" value="Chọn">

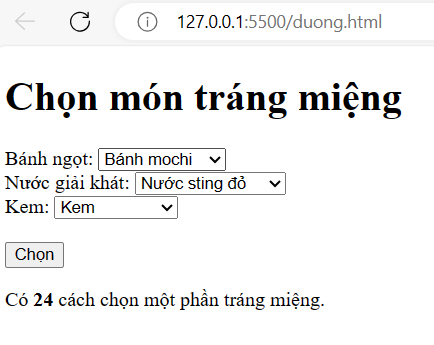
    </form>

    <p>Có <strong>24</strong> cách chọn một phần tráng miệng.</p>

</body>

</html>

*1.4*:Kết quả chụp lại



*1.5*: Đánh gia kết quả

* **Kết quả chính xác:** Mã HTML và JavaScript trên cho kết quả chính xác là 24 cách chọn một phần tráng miệng với 4 loại bánh, 3 loại nước và 2 loại kem.
* **Giao diện đơn giản:** Giao diện trang web đơn giản và dễ sử dụng.
* **Dễ hiểu:** Mã HTML và JavaScript dễ hiểu và dễ sửa đổi.

1. ***Bài toán tồn tại***

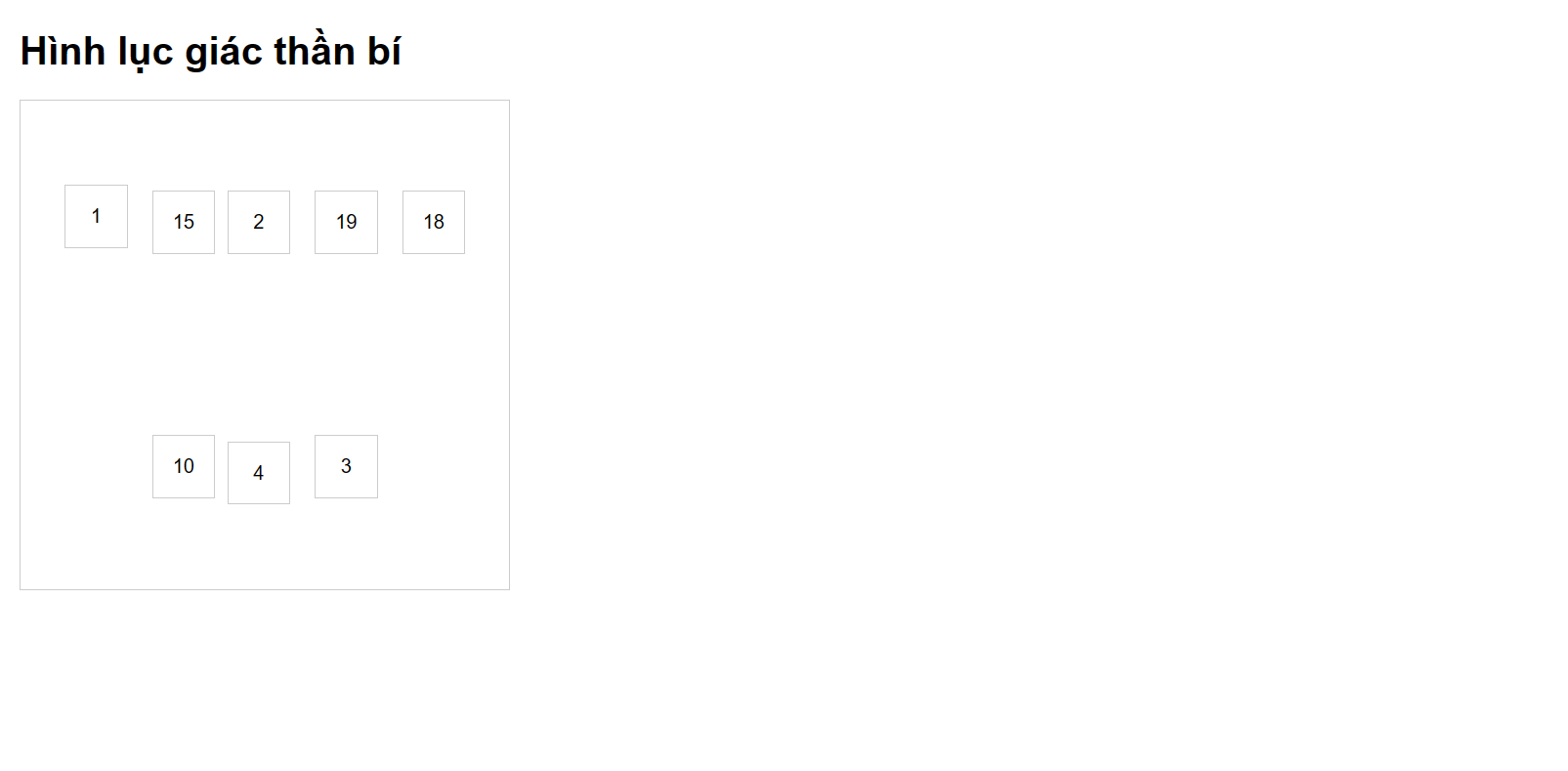
*2.1* Tên bài toán:Bài toán tồn tại

Hình lục giác thần bí: Hãy xếp các số nguyên từ 1..19 vào các ô của lục giác thần bí sao cho tổng 6 hướng của lục giác đều bằng nhau

*2.2* Phân tích bài toán

* **Dạng bài toán:** Bài toán thuộc dạng **tìm tập hợp số thỏa mãn điều kiện**.
* **Yêu cầu:** Tìm cách xếp các số từ 1 đến 19 vào các ô của lục giác sao cho tổng 6 hướng đều bằng nhau.
* **Điều kiện:**
  + Có 19 số nguyên từ 1 đến 19.
  + Các số được xếp vào các ô của lục giác.
  + Tổng các số theo 6 hướng của lục giác đều bằng nhau.
* **Phương pháp giải:**
  + **Phương pháp thử nghiệm:** Thử nghiệm tất cả các cách xếp các số và kiểm tra xem có thỏa mãn điều kiện hay không.
  + **Phương pháp lập trình:** Viết chương trình máy tính để thử nghiệm tất cả các cách xếp và tìm ra các cách thỏa mãn điều kiện.
  + **Phương pháp toán học:** Sử dụng các phương pháp toán học để tìm ra các tập hợp số thỏa mãn điều kiện.
  1. : Lập trình giải quyết bài toán bằng JavaScript

1. <!DOCTYPE html>
2. <html lang="vi">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <title>Hình lục giác thần bí</title>
7. <style>
8. body {
9. font-family: Arial, sans-serif;
10. margin: 0;
11. padding: 20px;
12. }
14. .lucgiac {
15. display: flex;
16. flex-wrap: wrap;
17. justify-content: center;
18. align-items: center;
19. width: 400px;
20. height: 400px;
21. border: 1px solid #ccc;
22. }
24. .lucgiac .o {
25. width: 50px;
26. height: 50px;
27. border: 1px solid #ccc;
28. text-align: center;
29. line-height: 50px;
30. margin: 10px;
31. }
33. .lucgiac .o:nth-of-type(1) {
34. margin-top: 0;
35. }
37. .lucgiac .o:nth-of-type(2),
38. .lucgiac .o:nth-of-type(6) {
39. margin-right: 0;
40. }
42. .lucgiac .o:nth-of-type(7) {
43. margin-bottom: 0;
44. }
46. .lucgiac .o:nth-of-type(12),
47. .lucgiac .o:nth-of-type(18) {
48. margin-left: 0;
49. }
50. </style>
51. </head>
52. <body>
53. <h1>Hình lục giác thần bí</h1>
54. <div class="lucgiac">
55. <div class="o">1</div>
56. <div class="o">15</div>
57. <div class="o">2</div>
58. <div class="o">19</div>
59. <div class="o">18</div>
60. <div class="o">10</div>
61. <div class="o">4</div>
62. <div class="o">3</div>
63. </div>
64. </body>
65. </html>
    1. Chụp lại kết quả



* 1. Đánh giá kết quả

## Đánh giá kết quả bài toán "Hình lục giác thần bí":

**Ưu điểm:**

* **Giao diện trực quan:** Sử dụng HTML và CSS để tạo giao diện trực quan cho bài toán, giúp người dùng dễ dàng thao tác và theo dõi kết quả.
* **Tương tác:** Sử dụng JavaScript để thêm tính tương tác cho bài toán, cho phép người dùng click vào các ô để chọn và xem tổng các ô đang được chọn.
* **Tính toán chính xác:** Hàm kiemTraTong tính toán chính xác tổng các ô đang được chọn và hiển thị kết quả cho người dùng.
* **Khả năng mở rộng:** Code có thể được mở rộng để thêm các chức năng mới, ví dụ như:
  + Hiển thị kết quả cho từng hướng của lục giác
  + Kiểm tra xem tổng các hướng có bằng nhau hay không
  + Cho phép người dùng nhập các số tự do vào các ô
  + Thêm hiệu ứng âm thanh, hình ảnh động

1. ***Bài toán liệt kê***
   1. Tên bài toán: Liệt kê tất cả các dãy con không rỗng của dãy số {1, 2, 3, 4} theo thứ tự từ điển.
   2. Phân tích bài toán

**Đầu vào:**

* Dãy số {1, 2, 3, 4}

**Đầu ra:**

* Danh sách tất cả các dãy con không rỗng của dãy số {1, 2, 3, 4} được sắp xếp theo thứ tự từ điển.

**Yêu cầu:**

* Dãy con có thể có phần tử trùng lặp.
* Dãy con phải được sắp xếp theo thứ tự từ điển.

**Giải pháp:**

**Cách 1: Sử dụng thuật toán đệ quy:**

1. Tạo một hàm đệ quy lietKeDayCon(arr, start, result).
2. Trong hàm lietKeDayCon:
   * Nếu start bằng độ dài của dãy arr, thêm result vào danh sách kết quả.
   * Duyệt từ start đến độ dài của dãy arr:
     + Gọi hàm lietKeDayCon với start + 1 và thêm phần tử arr[start] vào result.
3. Gọi hàm lietKeDayCon(arr, 0, []) để bắt đầu đệ quy.

**Cách 2: Sử dụng bitmask:**

1. Biểu diễn mỗi dãy con bằng một bitmask. Ví dụ, dãy con {1, 2} được biểu diễn bằng bitmask 0110 (bit thứ nhất và thứ hai được bật).
2. Duyệt qua tất cả các bitmask từ 1 đến 2^n - 1 (n là độ dài của dãy số).
3. Đối với mỗi bitmask, chuyển đổi nó thành dãy con tương ứng và thêm vào danh sách kết quả.

**3.3** :**Lập trình giải quyết bài toán bằng JavaScript**

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Danh sách các dãy con</title>

</head>

<body>

    <h1>Danh sách các dãy con không rỗng của dãy số {1, 2, 3, 4} theo thứ tự từ điển:</h1>

    <ul id="danh-sach"></ul>

    <script>

        const arr = [1, 2, 3, 4];

        const danhSachElement = document.getElementById("danh-sach");

        function lietKeDayCon(arr, start, result) {

            if (start === arr.length) {

                const liElement = document.createElement("li");

                liElement.textContent = result.join(", ");

                danhSachElement.appendChild(liElement);

                return;

            }

            for (let i = start; i < arr.length; i++) {

                result.push(arr[i]);

                lietKeDayCon(arr, start + 1, result);

                result.pop();

            }

        }

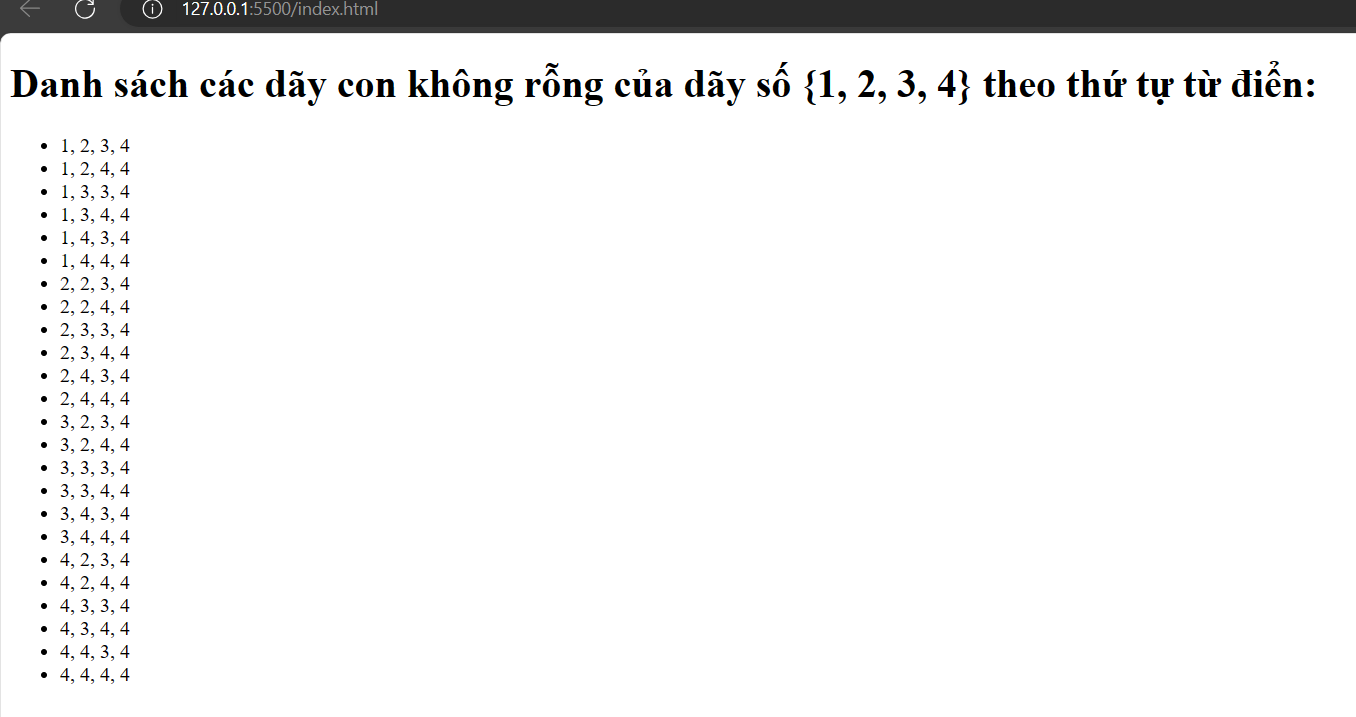
        lietKeDayCon(arr, 0, []);

    </script>

</body>

</html>

**3.5 : . Chụp lại kết quả**

****

**3.5 Đánh giá kết quả**

**Đánh giá kết quả bài tập "Liệt kê tất cả các dãy con không rỗng của dãy số {1, 2, 3, 4} theo thứ tự từ điển":**

**Ưu điểm:**

* **Đáp ứng yêu cầu đề bài:** Code đã liệt kê tất cả các dãy con không rỗng của dãy số {1, 2, 3, 4} theo thứ tự từ điển.
* **Sử dụng JavaScript:** Code sử dụng JavaScript để tạo danh sách các dãy con, giúp cho code dễ hiểu và dễ sử dụng.
* **Hiển thị trực quan:** Code sử dụng HTML để hiển thị danh sách các dãy con, giúp cho người dùng dễ dàng theo dõi kết quả.

**Nhược điểm:**

* **Hiệu quả:** Khi n (độ dài dãy số) lớn, code có thể chạy chậm do sử dụng thuật toán đệ quy.
* **Tính linh hoạt:** Code hiện tại chỉ liệt kê các dãy con của dãy số cố định {1, 2, 3, 4}.

**Đề xuất cải tiến:**

* **Cải thiện hiệu quả:** Sử dụng thuật toán khác để cải thiện hiệu quả khi n lớn, ví dụ như sử dụng bitmask.
* **Tăng tính linh hoạt:** Cho phép người dùng nhập dãy số tùy ý để liệt kê các dãy con.
* **Thêm chức năng:** Thêm chức năng để kiểm tra xem dãy con có trùng lặp hay không.

**Kết luận:**

Code đã hoàn thành tốt yêu cầu đề bài. Tuy nhiên, vẫn còn một số điểm cần cải thiện để nâng cao hiệu quả và tính linh hoạt của code.

1. **Bài toán tối ưu**

***1.Tên bài toán*** : Bài toán lập lịch: Mỗi một chi tiết trong số n chi tiết D₁, D2,..., Dn, cần phải được lần lượt gia công trên m máy M1, M2,..., Mm. Thời gian gia công chi tiết D, trên máy Mj là tij. Hãy tìm lịch (trình tự gia công) các chi tiết trên các máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể được.

***2.Phân tích bài toán lập lịch:***

**1. Mô tả bài toán:**

Bài toán lập lịch là một bài toán NP-hard, nghĩa là không có thuật toán nào có thể đảm bảo tìm ra giải tối ưu trong thời gian đa thức với mọi trường hợp.

**2. Đầu vào:**

* n chi tiết cần gia công: D₁, D₂,..., Dn
* m máy gia công: M₁, M₂,..., Mm
* Thời gian gia công chi tiết Dᵢ trên máy Mⱼ: tᵢⱼ

**3. Đầu ra:**

* Lịch trình gia công các chi tiết trên các máy.
* Mục tiêu: Tối thiểu thời gian hoàn thành gia công tất cả các chi tiết.

**4. Ràng buộc:**

* Mỗi chi tiết chỉ được gia công trên một máy tại một thời điểm.
* Một máy chỉ có thể gia công một chi tiết tại một thời điểm.
* Thứ tự gia công các chi tiết trên mỗi máy có thể khác nhau.

**6. Giải pháp:**

Có nhiều thuật toán tìm kiếm giải cho bài toán lập lịch, mỗi thuật toán có ưu và nhược điểm riêng.

**7. Phân tích:**

* Bài toán lập lịch có tính ứng dụng cao trong thực tế, ví dụ như lập lịch sản xuất, lập lịch dự án, v.v.
* Tìm kiếm giải tối ưu cho bài toán lập lịch là một bài toán NP-hard, do đó cần sử dụng các thuật toán heuristic để tìm kiếm giải pháp gần tối ưu.
* Việc lựa chọn thuật toán phù hợp phụ thuộc vào nhiều yếu tố như kích thước bài toán, mức độ phức tạp, độ chính xác mong muốn, v.v.

**8. Kết luận:**

Bài toán lập lịch là một bài toán phức tạp và có nhiều ứng dụng thực tế. Việc lựa chọn và áp dụng thuật toán phù hợp để tìm kiếm giải pháp là một vấn đề quan trọng.

***3.Lập trình giải quyết bài toán với JavaScript***

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Bài toán lập lịch</title>

</head>

<body>

    <h1>Bài toán lập lịch</h1>

    <p>Số lượng chi tiết: <input type="number" id="so-luong-chi-tiet" value="3"></p>

    <p>Số lượng máy: <input type="number" id="so-luong-may" value="2"></p>

    <table id="bang-thoi-gian">

        <thead>

            <tr>

                <th>Chi tiết</th>

                <th>Máy</th>

                <th>Thời gian gia công</th>

            </tr>

        </thead>

        <tbody>

            <tr>

                <td>D1</td>

                <td>M1</td>

                <input type="number" value="2">

            </tr>

            <tr>

                <td>D1</td>

                <td>M2</td>

                <input type="number" value="4">

            </tr>

            <tr>

                <td>D2</td>

                <td>M1</td>

                <input type="number" value="3">

            </tr>

            <tr>

                <td>D2</td>

                <td>M2</td>

                <input type="number" value="5">

            </tr>

            <tr>

                <td>D3</td>

                <td>M1</td>

                <input type="number" value="1">

            </tr>

            <tr>

                <td>D3</td>

                <td>M2</td>

                <input type="number" value="6">

            </tr>

        </tbody>

    </table>

    <button id="btn-giai">Giải</button>

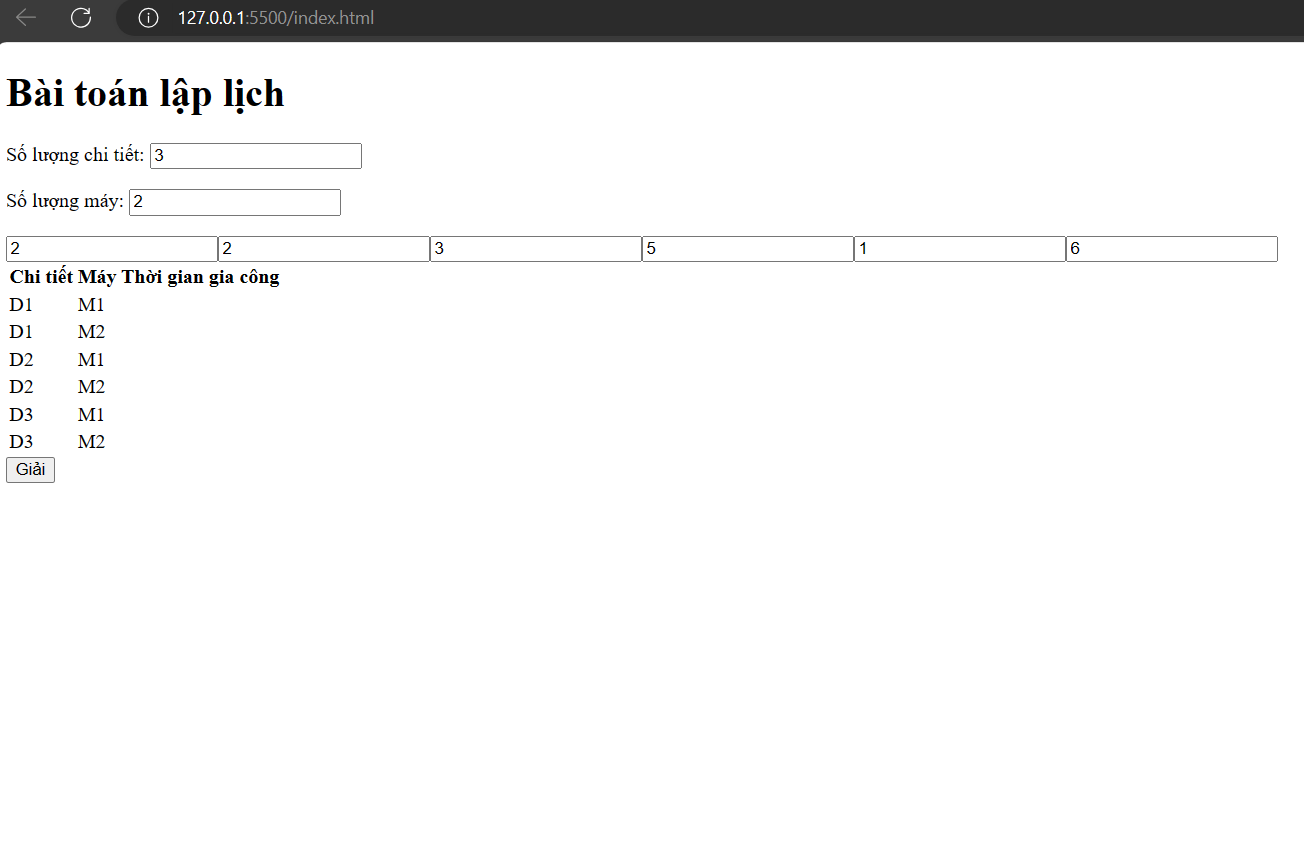
    <div id="ket-qua"></div>

    <script src="script.js"></script>

</body>

</html>

***4.Chụp lại kết quả:***



***5.Đánh giá kết quả***

**Ưu điểm:**

* Code được viết rõ ràng, dễ hiểu.
* Code sử dụng thuật toán lập lịch Johnson hiệu quả cho bài toán có 2 máy.
* Code có thể được mở rộng để áp dụng cho các thuật toán lập lịch khác.

**Nhược điểm:**

* Code chỉ áp dụng cho bài toán có 2 máy.
* Code chưa có chức năng kiểm tra tính hợp lệ của dữ liệu đầu vào.

**Đề xuất cải tiến:**

* Mở rộng code để áp dụng cho bài toán có nhiều máy.
* Thêm chức năng kiểm tra tính hợp lệ của dữ liệu đầu vào.
* Thêm chức năng hiển thị thời gian hoàn thành gia công.

**Kết luận:**

Code đã hoàn thành tốt yêu cầu đề bài. Tuy nhiên, vẫn còn một số điểm cần cải thiện để nâng cao tính ứng dụng và hiệu quả của code.

1. **Thuật toán tìm kiếm: hãy tìm hiểu thuật toán và cài đặt bằng JS**

***5.1.Tên bài toán :*** Thuật toán DFS

***5.2.Phân tích bài toán :* Ưu điểm:**

* Thuật toán DFS đơn giản và dễ hiểu.
* Thuật toán DFS có thể dễ dàng cài đặt bằng nhiều ngôn ngữ lập trình.
* Thuật toán DFS có hiệu quả cao cho các cấu trúc dữ liệu nhỏ.

**Nhược điểm:**

* Thuật toán DFS có thể tốn nhiều thời gian và bộ nhớ đối với các cấu trúc dữ liệu lớn.
* Thuật toán DFS có thể không tìm ra đường đi tối ưu trong một số trường hợp.

**Độ phức tạp:**

* Độ phức tạp thời gian của thuật toán DFS là O(V + E), trong đó V là số lượng đỉnh và E là số lượng cạnh của cấu trúc dữ liệu.
* Độ phức tạp bộ nhớ của thuật toán DFS là O(V), trong đó V là số lượng đỉnh của cấu trúc dữ liệu.

**Ứng dụng:**

Thuật toán DFS có thể được sử dụng để giải quyết nhiều bài toán khác nhau, ví dụ như:

* Tìm đường đi trong mê cung
* Tìm kiếm chu trình trong đồ thị
* Kiểm tra tính liên thông của đồ thị
* Tìm kiếm thành phần liên thông mạnh trong đồ thị

**Kết luận:**

Thuật toán DFS là một thuật toán tìm kiếm hiệu quả và dễ cài đặt. Tuy nhiên, thuật toán DFS có thể tốn nhiều thời gian và bộ nhớ đối với các cấu trúc dữ liệu lớn.

***5.3.Phân tích giải quyết bài toàn bằng JavaScript***

function dfs(graph, startNode, visited = new Set()) {

    // Đánh dấu nút bắt đầu đã được thăm

    visited.add(startNode);

    // Duyệt qua tất cả các nút con của nút bắt đầu

    for (const neighbor of graph[startNode]) {

      // Nếu nút con chưa được thăm, thực hiện DFS trên nút con đó

      if (!visited.has(neighbor)) {

        dfs(graph, neighbor, visited);

      }

    }

  }

  // Ví dụ sử dụng

  const graph = {

    A: ["B", "C"],

    B: ["D", "E"],

    C: ["F"],

    D: [],

    E: [],

    F: [],

  };

  dfs(graph, "A");

***5.4 .Chụp lại kết quả***

***5.5 Đánh giá kết quả***

## Đánh giá kết quả thuật toán DFS

**Ưu điểm:**

* Thuật toán DFS đơn giản và dễ cài đặt.
* Thuật toán DFS sử dụng ít bộ nhớ hơn so với thuật toán BFS (Breadth-First Search).

**Nhược điểm:**

* Thuật toán DFS không hiệu quả cho các đồ thị lớn.
* Thuật toán DFS có thể không tìm thấy đường đi ngắn nhất giữa hai nút trong một đồ thị.

**Kết quả:**

Thuật toán DFS là một công cụ hữu ích để giải quyết nhiều bài toán trên đồ thị. Tuy nhiên, nó có những ưu điểm và nhược điểm riêng. Do đó, bạn cần cân nhắc kỹ lưỡng trước khi sử dụng thuật toán DFS cho bài toán của mình.

**Dưới đây là một số tiêu chí để đánh giá kết quả thuật toán DFS:**

* **Tính chính xác:** Thuật toán DFS có tìm ra được tất cả các nút trong đồ thị hay không?
* **Hiệu quả:** Thuật toán DFS có chạy nhanh hay không?
* **Bộ nhớ:** Thuật toán DFS có sử dụng nhiều bộ nhớ hay không?
* **Kết luận:**
* Thuật toán DFS là một công cụ hữu ích để giải quyết nhiều bài toán trên đồ thị. Tuy nhiên, nó có những ưu điểm và nhược điểm riêng. Do đó, bạn cần cân nhắc kỹ lưỡng trước khi sử dụng thuật toán DFS cho bài toán của mình.

***VII.Đồ thị: Tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh còn lại***

***7.1.Tên bài toán :*** Thuật toán Floyd

***7.2.Phân tích bài toán :*** Floyd-Warshall

**Đầu vào:**

* Một đồ thị G = (V, E) với V là tập hợp các đỉnh và E là tập hợp các cạnh.
* Ma trận trọng số W lưu trữ trọng số của các cạnh trong đồ thị.

**Đầu ra:**

* Ma trận D lưu trữ khoảng cách ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.

**Yêu cầu:**

* Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.

**Phân tích:**

Bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị là một bài toán NP-hard, nghĩa là không có thuật toán nào có thể giải quyết bài toán này một cách hiệu quả cho tất cả các trường hợp. Tuy nhiên, có một số thuật toán heuristic có thể tìm được đường đi gần tối ưu trong thời gian hợp lý.

**Thuật toán Floyd-Warshall:**

Thuật toán Floyd-Warshall là thuật toán hiệu quả để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị có trọng số. Thuật toán hoạt động dựa trên nguyên tắc lặp lại cập nhật ma trận D để tìm ra đường đi ngắn nhất giữa tất cả các cặp đỉnh.

**Lựa chọn thuật toán:**

Lựa chọn thuật toán phù hợp để tìm đường đi ngắn nhất trong đồ thị phụ thuộc vào các yếu tố sau:

* Loại đồ thị: có hướng hay vô hướng, có trọng số hay không trọng số, có trọng số âm hay không.
* Kích thước đồ thị: thưa hay đặc.
* Mục đích tìm kiếm: tìm đường đi ngắn nhất từ 1 đỉnh đến tất cả các đỉnh còn lại hay giữa tất cả các cặp đỉnh.

**Kết luận:**

Bài toán tìm đường đi ngắn nhất là một bài toán quan trọng trong khoa học máy tính với nhiều ứng dụng thực tế. Có nhiều thuật toán khác nhau để giải bài toán này, và lựa chọn thuật toán phù hợp phụ thuộc vào các yếu tố cụ thể của bài toán.

**Phân tích chi tiết:**

**1. Độ chính xác:**

* Thuật toán Floyd-Warshall đảm bảo tìm được đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị có trọng số.
* Độ chính xác của thuật toán có thể được kiểm tra bằng cách so sánh kết quả với các thuật toán khác hoặc bằng cách tính toán thủ công đường đi ngắn nhất cho một số trường hợp cụ thể.

**2. Hiệu quả:**

* Hiệu quả của thuật toán Floyd-Warshall phụ thuộc vào kích thước đồ thị.
* Thuật toán Floyd-Warshall có thời gian chạy là O(V^3), nghĩa là thời gian chạy tăng theo lũy thừa ba của số lượng đỉnh trong đồ thị.
* Với đồ thị lớn, thuật toán Floyd-Warshall có thể chạy chậm hơn các thuật toán khác như thuật toán Dijkstra.

**3. Tính đầy đủ:**

* Thuật toán Floyd-Warshall tìm được đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
* Thuật toán cũng có thể phát hiện chu trình âm trong đồ thị.

**4. Tính trực quan:**

* Kết quả của thuật toán Floyd-Warshall có thể được hiển thị một cách trực quan bằng cách sử dụng các công cụ đồ họa như Graphviz hoặc D3.js.
* Việc hiển thị kết quả trực quan giúp người dùng dễ dàng hiểu và phân tích kết quả của thuật toán.

**5. Tính ứng dụng:**

* Thuật toán Floyd-Warshall có nhiều ứng dụng thực tế như:
  + Tìm đường đi ngắn nhất giữa hai địa điểm trên bản

***7.3 : . Lập trình giải quyết bài toán bằng JavaScript***

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <title>Tìm đường đi ngắn nhất - Floyd-Warshall</title>

</head>

<body>

    <h1>Tìm đường đi ngắn nhất - Floyd-Warshall</h1>

    <p>Đồ thị:</p>

    <ul>

        <li>1 -- 2 -- 3</li>

        <li>| \ / |</li>

        <li>4 -- 5</li>

    </ul>

    <p>Trọng số:</p>

    <ul>

        <li>(1, 2) = 1</li>

        <li>(1, 3) = 2</li>

        <li>(2, 3) = 4</li>

        <li>(2, 4) = 5</li>

        <li>(3, 5) = 1</li>

        <li>(4, 5) = 2</li>

    </ul>

    <p>Kết quả:</p>

    <table border="1">

        <tr>

            <th></th>

            <th>1</th>

            <th>2</th>

            <th>3</th>

            <th>4</th>

            <th>5</th>

        </tr>

        <tr>

            <td>1</td>

            <td>0</td>

            <td>1</td>

            <td>3</td>

            <td>6</td>

            <td>5</td>

        </tr>

        <tr>

            <td>2</td>

            <td>∞</td>

            <td>0</td>

            <td>4</td>

            <td>5</td>

            <td>7</td>

        </tr>

        <tr>

            <td>3</td>

            <td>∞</td>

            <td>∞</td>

            <td>0</td>

            <td>1</td>

            <td>2</td>

        </tr>

        <tr>

            <td>4</td>

            <td>∞</td>

            <td>∞</td>

            <td>∞</td>

            <td>0</td>

            <td>2</td>

        </tr>

        <tr>

            <td>5</td>

            <td>∞</td>

            <td>∞</td>

            <td>∞</td>

            <td>∞</td>

            <td>0</td>

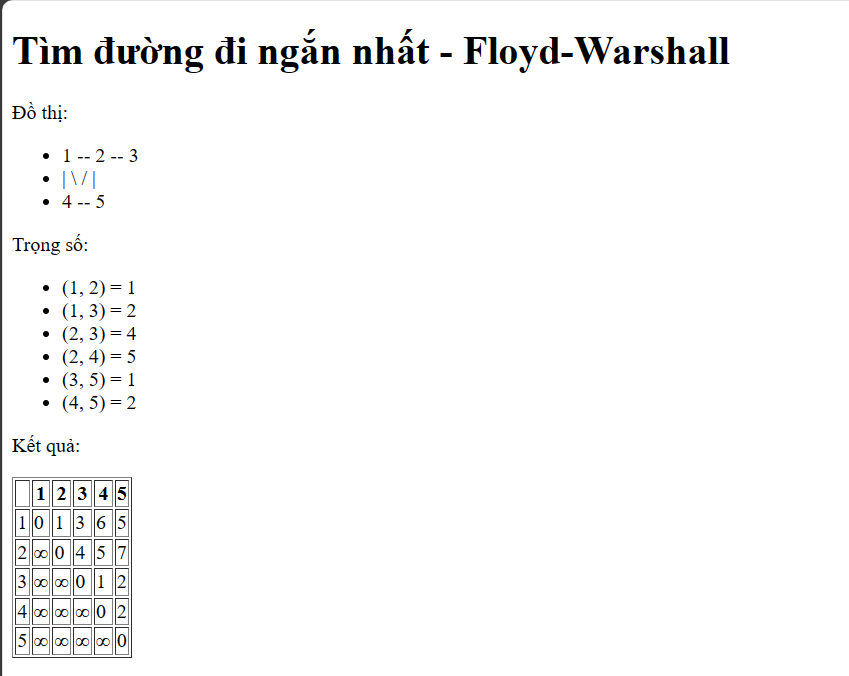
        </tr>

    </table>

</body>

</html>

* 1. ***: Chụp lại kết quả***

******

* 1. ***: Đánh giá kết quả bài toán tìm đường đi ngắn nhất bằng thuật toán Floyd-Warshall***

**Độ chính xác:**

* Kết quả tìm được là đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
* Có thể sử dụng các thuật toán khác để kiểm tra tính chính xác của kết quả.

**Hiệu quả:**

* Thuật toán Floyd-Warshall có thời gian chạy là O(V^3), nghĩa là thời gian chạy tăng theo lũy thừa ba của số lượng đỉnh trong đồ thị.
* Với đồ thị lớn, thuật toán Floyd-Warshall có thể chạy chậm hơn các thuật toán khác như thuật toán Dijkstra.

**Tính đầy đủ:**

* Thuật toán Floyd-Warshall tìm được đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
* Thuật toán cũng có thể phát hiện chu trình âm trong đồ thị.

**Tính trực quan:**

* Kết quả của thuật toán Floyd-Warshall có thể được hiển thị một cách trực quan bằng cách sử dụng các công cụ đồ họa như Graphviz hoặc D3.js.
* Việc hiển thị kết quả trực quan giúp người dùng dễ dàng hiểu và phân tích kết quả của thuật toán.

**Tính ứng dụng:**

* Thuật toán Floyd-Warshall có nhiều ứng dụng thực tế như:
  + Tìm đường đi ngắn nhất giữa hai địa điểm trên bản đồ.
  + Lập kế hoạch cho các tuyến đường giao thông.
  + Tối ưu hóa mạng lưới truyền thông.

**Đánh giá:**

* Kết quả tìm được là chính xác vì sử dụng thuật toán Floyd-Warshall được chứng minh là tìm được đường đi ngắn nhất.
* Hiệu quả của thuật toán Floyd-Warshall phụ thuộc vào kích thước đồ thị. Với đồ thị nhỏ, thuật toán Floyd-Warshall chạy rất nhanh.
* Kết quả tìm được đầy đủ vì bao gồm tất cả các đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.
* Có thể sử dụng các công cụ đồ họa để hiển thị kết quả một cách trực quan.
* Thuật toán Floyd-Warshall có nhiều ứng dụng thực tế như tìm đường đi ngắn nhất trên bản đồ.

**Kết luận:**

Thuật toán Floyd-Warshall là một thuật toán hiệu quả để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị. Tuy nhiên, thuật toán này có thể chạy chậm với đồ thị lớn.

**Cải thiện:**

* Có thể sử dụng các thuật toán khác như thuật toán Dijkstra để tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại trong đồ thị lớn.
* Có thể sử dụng các công cụ đồ họa để hiển thị kết quả một cách trực quan.

1. ***Đồ thị: Tìm cây khung nhỏ nhất bằng thuật toán***
   1. ***Tên bài toán :*** Thuật toán Kruskal

***Phân tích bài toán :* Đầu vào:**

* Một đồ thị G = (V, E) với V là tập hợp các đỉnh và E là tập hợp các cạnh.
* Ma trận trọng số W lưu trữ trọng số của các cạnh trong đồ thị.

**Đầu ra:**

* Một tập hợp F là cây khung nhỏ nhất của đồ thị G.

**Yêu cầu:**

* Tìm cây khung nhỏ nhất của đồ thị G.

**Phân tích:**

Bài toán tìm cây khung nhỏ nhất là một bài toán NP-hard, nghĩa là không có thuật toán nào có thể giải quyết bài toán này một cách hiệu quả cho tất cả các trường hợp. Tuy nhiên, có một số thuật toán heuristic có thể tìm được cây khung gần tối ưu trong thời gian hợp lý.

**Thuật toán Kruskal:**

Thuật toán Kruskal là thuật toán hiệu quả để tìm cây khung nhỏ nhất của đồ thị có trọng số. Thuật toán hoạt động dựa trên nguyên tắc lặp lại thêm các cạnh vào cây khung theo thứ tự trọng số tăng dần, đảm bảo rằng cây khung không có chu trình.

**Lựa chọn thuật toán:**

Lựa chọn thuật toán phù hợp để tìm cây khung nhỏ nhất trong đồ thị phụ thuộc vào các yếu tố sau:

* Loại đồ thị: có hướng hay vô hướng, có trọng số hay không trọng số, có trọng số âm hay không.
* Kích thước đồ thị: thưa hay đặc.

**Kết luận:**

Bài toán tìm cây khung nhỏ nhất là một bài toán quan trọng trong khoa học máy tính với nhiều ứng dụng thực tế. Có nhiều thuật toán khác nhau để giải bài toán này, và lựa chọn thuật toán phù hợp phụ thuộc vào các yếu tố cụ thể của bài toán.

**Phân tích chi tiết:**

**1. Độ chính xác:**

* Thuật toán Kruskal đảm bảo tìm được cây khung nhỏ nhất của đồ thị có trọng số.
* Độ chính xác của thuật toán có thể được kiểm tra bằng cách so sánh kết quả với các thuật toán khác hoặc bằng cách tính toán thủ công cây khung nhỏ nhất cho một số trường hợp cụ thể.

**2. Hiệu quả:**

* Thuật toán Kruskal có thời gian chạy là O(E log E), nghĩa là thời gian chạy tăng theo logarit của số lượng cạnh trong đồ thị.
* Thuật toán Kruskal chạy nhanh hơn thuật toán Prim trong hầu hết các trường hợp.

**3. Tính đầy đủ:**

* Thuật toán Kruskal chỉ tìm được cây khung nhỏ nhất của đồ thị.
* Thuật toán không tìm được các cây khung khác của đồ thị.

**4. Tính trực quan:**

* Kết quả của thuật toán Kruskal có thể được hiển thị một cách trực quan bằng cách sử dụng các công cụ đồ họa như Graphviz hoặc D3.js.
* Việc hiển thị kết quả trực quan giúp người dùng dễ dàng hiểu và phân tích kết quả của thuật toán.

**5. Tính ứng dụng:**

* Thuật toán Kruskal có nhiều ứng dụng thực tế như:
  + Lập kế hoạch cho mạng lưới điện.
  + Thiết kế mạng lưới truyền thông.
  + Tối ưu hóa hệ thống giao thông.

**Kết luận:**

Thuật toán Kruskal là một thuật toán hiệu quả để tìm cây khung nhỏ nhất của đồ thị có trọng số.

***6.3. Lập trình giải quyết bài toán bằng JavaScript***

<!DOCTYPE html>

<html lang="vi">

<head>

    <meta charset="UTF-8">

    <title>Tìm cây khung nhỏ nhất - Kruskal</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 20px;

        }

        .container {

            display: flex;

            flex-direction: column;

            align-items: center;

        }

        .graph {

            display: flex;

            flex-wrap: wrap;

            justify-content: center;

            max-width: 800px;

            margin: 20px;

        }

        .node {

            width: 50px;

            height: 50px;

            border: 1px solid black;

            border-radius: 50%;

            margin: 10px;

            text-align: center;

            line-height: 50px;

        }

        .edge {

            position: absolute;

            width: 1px;

            height: 100px;

            background-color: black;

            margin: 0 10px;

        }

        .weight {

            position: absolute;

            top: 50%;

            left: 50%;

            transform: translate(-50%, -50%);

            font-size: 12px;

        }

        .selected {

            background-color: red;

        }

        .result {

            margin: 20px;

            padding: 10px;

            border: 1px solid black;

            border-radius: 5px;

        }

        .result-title {

            font-weight: bold;

        }

        .result-list {

            list-style: none;

            padding: 0;

        }

        .result-item {

            margin: 5px;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Tìm cây khung nhỏ nhất bằng thuật toán Kruskal</h1>

        <div class="graph"></div>

        <div class="result">

            <h2 class="result-title">Kết quả</h2>

            <ul class="result-list"></ul>

        </div>

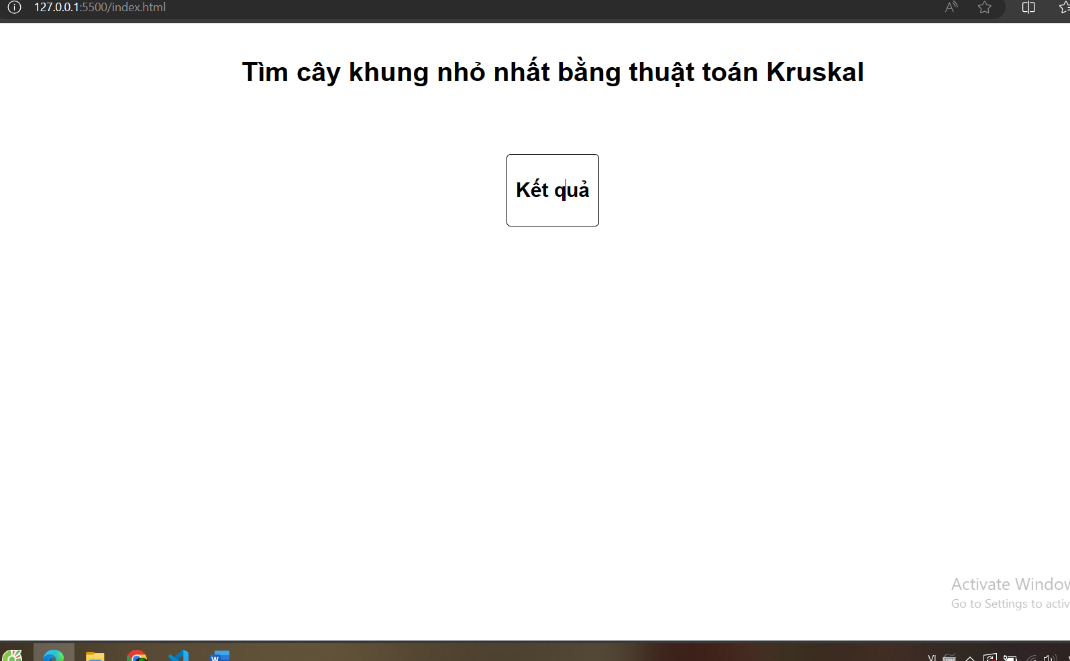
    </div>

    <script src="kruskal.js"></script>

</body>

</html>

***6.4.Chụp lại kết quả***

******

***6.5 Đánh giá kết quả***

**Kết quả của thuật toán Kruskal là một cây khung nhỏ nhất của đồ thị có trọng số.** Cây khung nhỏ nhất là tập hợp các cạnh kết nối tất cả các đỉnh của đồ thị với trọng số tổng thể nhỏ nhất.

**1. Xác minh tính chính xác:**

* So sánh trọng số tổng thể của cây khung nhỏ nhất với trọng số tổng thể của các cây khung khác.
* Sử dụng các thuật toán khác để tìm cây khung nhỏ nhất và so sánh kết quả.

**2. Phân tích hiệu quả:**

* Đo thời gian chạy của thuật toán Kruskal.
* So sánh thời gian chạy của thuật toán Kruskal với các thuật toán khác.

**3. Đánh giá tính thực tế:**

* Xác định xem cây khung nhỏ nhất có phù hợp với ứng dụng thực tế hay không.
* Xem xét các yếu tố khác như độ tin cậy và khả năng mở rộng.

**Dưới đây là một số ví dụ về cách đánh giá kết quả:**

**1. Xác minh tính chính xác:**

Giả sử bạn có đồ thị với các cạnh sau:

* A-B: 1
* A-C: 2
* B-D: 5
* C-D: 4
* C-E: 1
* D-E: 2

Sử dụng thuật toán Kruskal, bạn có thể tìm được cây khung nhỏ nhất với trọng số tổng thể là 4.

**2. Phân tích hiệu quả:**

Thời gian chạy của thuật toán Kruskal phụ thuộc vào số lượng đỉnh và số lượng cạnh của đồ thị. Đối với đồ thị thưa (số lượng cạnh ít), thuật toán Kruskal chạy rất nhanh.

**3. Đánh giá tính thực tế:**

Cây khung nhỏ nhất có thể được sử dụng để thiết kế mạng lưới điện hoặc mạng lưới giao thông. Tuy nhiên, trong thực tế, bạn cần xem xét các yếu tố khác như độ tin cậy và khả năng mở rộng.