ĐẠI HỌC KINH TẾ THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

ĐỒ ÁN MÔN HỌC

ĐỀ TÀI:

GIẢI BÀI TOÁN NETWORK FLOW PROBLEM DỰA TRÊN GIẢI THUẬT GREEDY BEST FIRST SEARCH

Học phần: Trí Tuệ Nhân Tạo

Nhóm Sinh Viên:

1. NGUYỄN TRẦN THẾ ANH

2. PHẠM BẰNG

3. NGUYỄN BẢO CÁT MINH

4. NGUYỄN THÀNH VINH

Chuyên Ngành: KHOA HỌC DỮ LIỆU

Khóa: K48

Giảng Viên: TS. Đặng Ngọc Hoàng Thành

TP. Hồ Chí Minh, ngày 8 tháng 6 năm 2024.

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN

1.1. Bài toán tối ưu Mạng/Luồng giao thông (Network flow problem)

1.1.1. Lịch sử hình thành

Bài toán luồng trên mạng (network flow) là một trong những bài toán quan trọng trong lĩnh vực toán ứng dụng và khoa học máy tính. Bài toán được phát triển đầu tiên như một phần của nghiên cứu về các vấn đề vận tải và phân phối năm 1940.

Lý thuyết về luồng và mạng được phát triển từ các nghiên cứu về các vấn đề giao thông và phân phối hàng hóa. Đầu tiên, bài toán được quan tâm đến là bài toán vận tải, trong đó mục tiêu là tối ưu hóa việc vận chuyển hàng hóa từ nguồn cung đến điểm tiêu thụ thông qua mạng lưới giao thông.

Cuối những năm 1940, những nhà khoa học đã phát triển lý thuyết và phương pháp giải bài toán, trong đó quan trọng nhất là mô hình hóa bài toán dưới dạng đồ thị có hướng với các đỉnh biểu diễn các điểm trong mạng lưới và các cạnh biểu diễn các tuyến đường vận chuyển. Nó đã được áp dụng rộng rãi trong các lĩnh vực như mạng lưới viễn thông, quản lý tài nguyên, và lập kế hoạch sản xuất.

1.1.2. Lý thuyết

Bài toán Network flow được mô hình hóa dưới dạng đồ thị có hướng như sau:

Cho một đồ thị có hướng G = (V, E) với một nguồn s và đỉnh đích t, mỗi cạnh (u, v) trong E có một dung lượng dương c(u, v) và một luồng f(u, v) không âm trên mỗi cạnh sao cho các điều kiện sau được thỏa mãn:

- Điều kiện 1 (Capacity Constraint): Luồng trên mỗi cung e ∈ E không vượt quá khả năng thông qua của nó:

$$0 \leq f(e) \leq c(e)$$

- Điều kiện 2 (Flow conservation): Tổng luồng trên các cung đi vào đỉnh v bằng tổng luồng trên các cung đi ra khỏi đỉnh v (nếu v khác s và t).

$$\sum_u f(u, v) = \sum_u f(v, u)$$ (với mọi u thuộc V, trừ s, t)

Trong bài toán, chúng ta thường cố gắng tối ưu hóa một giá trị, ví dụ như tổng luồng vào đích t (đối với bài toán luồng cực đại), hoặc tổng chi phí của luồng (đối với bài toán tối ưu luồng tối ưu). Mục tiêu của bài toán Network Flow là

tìm một luồng hợp lệ sao cho giá trị tối ưu của mục tiêu được đạt được và các điều kiện ràng buộc được thỏa mãn.

1.1.3. Các biến thể bài toán Network Flow

Bài toán luồng cực đại (Maximum Flow Problem - MFP): MFP là dạng cơ bản nhất của NFP, với mục tiêu tìm luồng lớn nhất có thể di chuyển từ nguồn cung cấp đến điểm thu trên mạng.

- Bài toán luồng tối thiểu chi phí (Minimum Cost Flow Problem - MCF): MCF tương tự như MFP, nhưng thay vì tối đa hóa luồng, MCF tập trung vào việc tìm luồng thỏa mãn nhu cầu với chi phí vận chuyển thấp nhất.
- Bài toán luồng đa nguồn đa cầu (Multi-Source Multi-Sink Flow Problem - MMSF): MMSF mở rộng MFP bằng cách cho phép nhiều nguồn cung cấp và nhiều điểm thu.
- Bài toán luồng trong mạng lưới (Network Flow Problem in a Grid): NFP trong mạng lưới áp dụng cho các bài toán NFP trên mạng lưới hình vuông hoặc hình chữ nhật, với các ràng buộc bổ sung về cấu trúc mạng.
- Bài toán luồng phân đoạn (Split Flow Problem): Bài toán luồng phân đoạn chia luồng thành nhiều luồng nhỏ hơn, mỗi luồng đi qua một đường dẫn khác nhau trên mạng.

Ngoài ra, còn có nhiều dạng NFP khác, bao gồm bài toán luồng với dung lượng cạnh thay đổi theo thời gian, bài toán luồng với ràng buộc về thời gian, và bài toán luồng với các nút trung gian.

Lựa chọn phương pháp giải phù hợp cho NFP phụ thuộc vào nhiều yếu tố, bao gồm kích thước mạng, cấu trúc mạng, và các ràng buộc của bài toán.

1.1.4. Một số ứng dụng thực tế của NFP

- Mạng máy tính: Định tuyến lưu lượng mạng, phân bổ băng thông, và kiểm soát tắc nghẽn mạng.
- Giao thông vận tải: Lập kế hoạch tuyến đường, tối ưu hóa lưu lượng giao thông, và quản lý chuỗi cung ứng.
- Tài chính: Lập kế hoạch đầu tư, quản lý rủi ro, và định giá tài sản.
- Sản xuất: Lập kế hoạch sản xuất, quản lý hàng tồn kho, và tối ưu hóa chuỗi cung ứng.
- Xã hội: Phân bổ tài nguyên, lập kế hoạch dự án, và quản lý mạng lưới xã hội.

NFP là một công cụ mạnh mẽ để giải quyết các vấn đề tối ưu hóa phức tạp trong nhiều lĩnh vực khác nhau. Việc hiểu rõ các dạng và phương pháp giải NFP sẽ giúp bạn lựa chọn phương pháp phù hợp để giải quyết các bài toán thực tế.

1.2. Bài toán luồng chi phí tối thiểu (Minimum Cost Flow Problem - MCF)

Phát biểu bài toán:

Bài toán Minimum cost flow (Luồng chi phí tối thiểu) là một biến thể của bài toán Network Flow, trong đó, mỗi cạnh của đồ thị được gán một chi phí. Mục tiêu của bài toán là tìm ra phương án có chi phí tổng nhỏ nhất từ nguồn đến đích để vận chuyển một lượng hàng cho trước, trong khi vẫn thỏa mãn các ràng buộc về dung lượng cạnh (Điều kiện 1) và bảo toàn luồng (Điều kiện 2).

Tóm tắt yêu cầu bài toán:

- Cho đồ thị có hướng G = (V, E), có **vertex** đỉnh và **edges** cạnh, với đỉnh **source** là đỉnh nguồn và đỉnh **sink** là đỉnh đích.
- Mỗi cạnh (u,v) $\in$ E có:
    - Sức chứa (Capacity) **c(u,v)**: Giới hạn tối đa lượng tài nguyên có thể di chuyển qua cạnh đó.
    - Chi phí (Cost) **cost(u,v)**: Chi phí di chuyển một đơn vị tài nguyên qua cạnh đó.
- Mục tiêu của bài toán là vận chuyển một lượng **flow** từ đỉnh **source** đến **sink** sao cho tổng chi phí là bé nhất mà vẫn thỏa 2 điều kiện đã nêu. Với công thức tính cost như sau:

$$\text{cost}(f) = \sum_{e \in E} \text{cost}(e) \cdot f(e)$$

1.3. Một số hướng tiếp cận giải quyết bài toán

Vấn đề về bài toán luồng tối thiểu chi phí có thể được giải quyết bằng phương pháp linear programming. Ngoài ra, còn có rất nhiều các thuật toán tổ hợp khác. Một vài trong số chúng là sự khái quát của thuật toán luồng cực đại, một số khác thì sử dụng các phương thức khác hoàn toàn.

Các thuật toán cơ bản để giải quyết bài toán:

Giải thuật Out-of-kilter: Ý tưởng chủ yếu của thuật toán dựa trên định nghĩa của "out-of-kilter" (OOK) cho mỗi cạnh trong mạng lưới. Thuật toán sẽ tìm cạnh có chỉ số OOK lớn nhất và tìm đường đi từ nguồn đến đích có sử dụng cạnh đó. Tăng luồng cho đường đi này lên mức tối đa và cập nhật lại chỉ số OOK cho các cạnh trên

đường đi. Quá trình này sẽ tiếp tục cho đến khi tất cả các cạnh đều có chỉ số out-of-kilter bằng 0, nghĩa là ta đã tìm được giải pháp tối ưu.

Giải thuật Cycle Canceling: Thuật toán với ý tưởng là tìm một chu trình có chi phí âm trong mạng lưới. Điều này có thể được thực hiện bằng cách sử dụng các thuật toán tìm đường đi ngắn nhất như Dijkstra hoặc Bellman-Ford. Tăng lưu lượng chảy qua chu trình có chi phí âm này lên đến khi một cạnh nào đó trong chu trình trở nên bão hòa. Cập nhật luồng trên các cạnh trong chu trình để loại bỏ chu trình có chi phí âm. Quá trình này lập đi lập lại cho đến khi không tìm được thêm chu trình có chi phí âm nào. Kết quả ta sẽ thu được luồng tối ưu với chi phí tối thiểu.

Giải thuật Successive Shortest Path: Thuật toán này tìm đường đi ngắn nhất từ nguồn đến đích, sau đó gửi lưu lượng tối đa có thể qua đường này. Tăng lưu lượng trên đường đi ngắn nhất này lên đến khi một cạnh nào đó trở nên bão hòa và cập nhật luồng trên các cạnh trong đường đi. Quá trình này lặp lại cho đến khi không còn đường đi nào ta sẽ thu được một luồng tối ưu.

Ngoài ra, còn có một số thuật toán khác cũng được dùng để tiếp cận giải quyết bài toán như: giải thuật Minimum Mean Cycle Canceling, giải thuật Primal-Dual, giải thuật Greedy Best First Search … Mỗi thuật toán điều có những ưu, nhược điểm khác nhau nên việc lựa chọn giải thuật phù hợp phụ thuộc vào quy mô và yêu cầu cụ thể của bài toán.

CHƯƠNG 2. GIẢI THUẬT GREEDY BEST FIRST SEARCH

2.1. Giới thiệu thuật toán Greedy Best First Search

2.1.1. Thuật toán Greedy Best First Search

Giải thuật Greedy Best First Search là một giải thuật tìm kiếm được sử dụng để tìm kiếm lời giải trong các bài toán tối ưu. Giải thuật này không thuộc nhóm các giải thuật tìm kiếm theo chiều cụ thể như Breadth-First Search (BFS) hay Depth-First Search (DFS). Thay vì tập trung vào việc mở rộng theo chiều ngang hoặc chiều dọc của cây tìm kiếm, GBFS sử dụng một hàm heuristic để tập trung vào việc lựa chọn hướng đi có giá trị hàm đánh giá tốt nhất tiếp theo.

Giải thuật Greedy Best First Search là giải thuật tìm kiếm đánh giá hoàn toàn bằng hàm heuristic. Bỏ qua các trọng số cạnh trong biểu đồ và chỉ xem xét giá trị heuristic. Để tìm kiếm nút mục tiêu, thuật toán mở rộng nút được xác định gần mục tiêu bởi hàm heuristic. Cách tiếp cận này giúp nó có khả năng đưa ra giải pháp nhanh chóng. Tuy nhiên, giải pháp từ Greedy Best First Search có thể không tối ưu vì có thể tồn tại đường đi ngắn hơn. Nó được gọi là "Tham lam" vì ở mỗi bước nó đều cố gắng tiến gần đến mục tiêu nhất.

Ý tưởng của giải thuật Greedy Best First Search là tìm kiếm trạng thái tiếp theo mà có chi phí ước tính tốt nhất (best-first) dựa trên hàm heuristic. Hàm heuristic là hàm đánh giá chi phí từ trạng thái hiện tại đến trạng thái đích mà không cần biết đầy đủ thông tin về tất cả các trạng thái. Trong quá trình tìm kiếm, giải thuật Greedy Best First Search chọn trạng thái tiếp theo có giá trị heuristic thấp nhất và tiếp tục tìm kiếm từ trạng thái đó. Quá trình này được tiếp tục cho đến khi tìm ra lời giải tối ưu hoặc không còn trạng thái nào để xét.

2.1.2. Đánh giá giải thuật

Độ phức tạp của Giải thuật Greedy Best First Search phụ thuộc vào cách thiết kế và cài đặt của hàm heuristic, số lượng trạng thái trong không gian tìm kiếm và cấu trúc của cây tìm kiếm được tạo ra bởi giải thuật.

Trong trường hợp tốt nhất, giải thuật Greedy Best First Search có độ phức tạp là O(bm), với b là số lượng trạng thái con của trạng thái hiện tại, m là số lượng trạng thái trong không gian tìm kiếm. Tuy nhiên, trong trường hợp xấu nhất, độ phức tạp có thể lên tới $O(b^m)$, khi giải thuật phải xét tất cả các trạng thái có thể có trong không gian tìm kiếm.

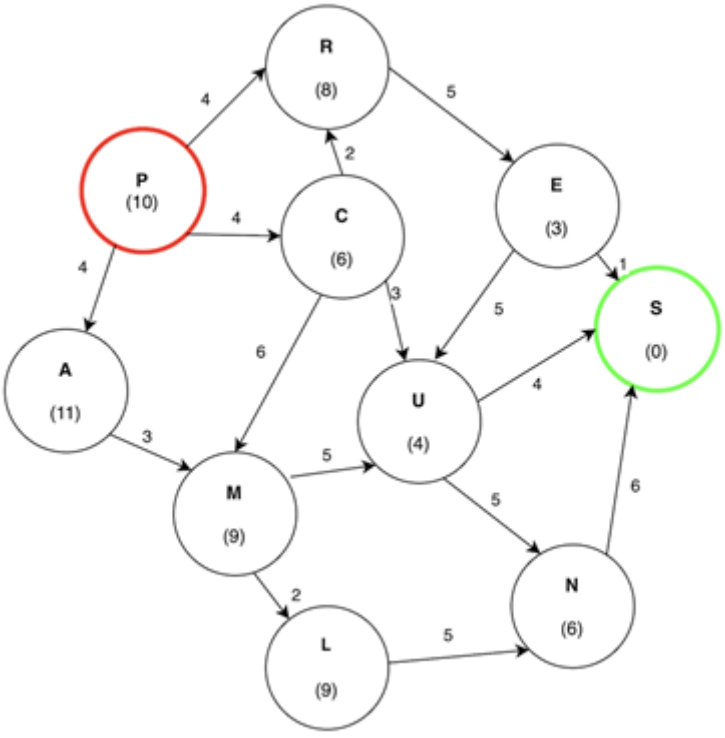
Ngoài ra, độ phức tạp của giải thuật còn phụ thuộc vào cách cài đặt của cây tìm kiếm và các phương pháp tối ưu hóa để loại bỏ các trạng thái trùng lặp trong quá trình tìm kiếm. Các phương pháp tối ưu hóa này có thể giúp giảm đáng kể độ phức tạp của giải thuật trong thực tế.

Giải thuật Greedy Best First Search có thể được sử dụng để giải quyết nhiều bài toán tối ưu như tìm đường đi ngắn nhất trên đồ thị, lập lịch sản xuất, tối ưu hóa đa mục tiêu và nhiều ứng dụng khác. Tuy nhiên, giải thuật này cũng có nhược điểm là có thể bị mắc kẹt vào các vòng lặp vô hạn nếu hàm heuristic không được thiết kế tốt hoặc không đủ thông tin để xác định lời giải tối ưu.

2.1.3. Ví dụ về giải thuật

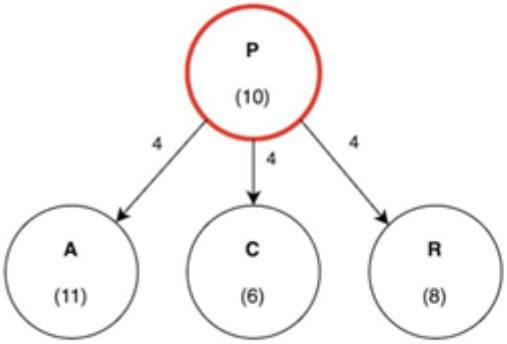
Sơ đồ cây của giải thuật Greedy Best First Search bao gồm các nút trạng thái, nút gốc của cây và các nút con mở rộng. Nút gốc của cây là trạng thái ban đầu của bài toán. Sau đó, giải thuật Greedy Best First Search duyệt nút gốc và tạo ra các nút con mở rộng. Thuật toán GBFS không mở rộng tất cả các nút như thuật toán tìm kiếm theo chiều rộng (Breadth First Search) và thuật toán tìm kiếm theo chiều sâu (Depth First Search) mà chỉ mở rộng các nút con được hàm heuristic đánh giá là tốt nhất. Tiếp tục quá trình này cho đến khi tìm thấy giải pháp hoặc không còn nút con để mở rộng.

VD: Tìm đường đi từ P đến S



Hình 2.1 Đồ thị tìm đường đi từ P đến S

Bước 1: Xét nút gốc và mở ra các nút con

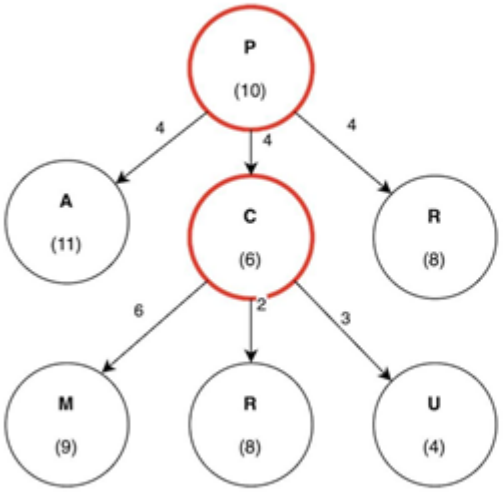


Node [cost]
C [6]
R [8]
A [11]

Closed List
P

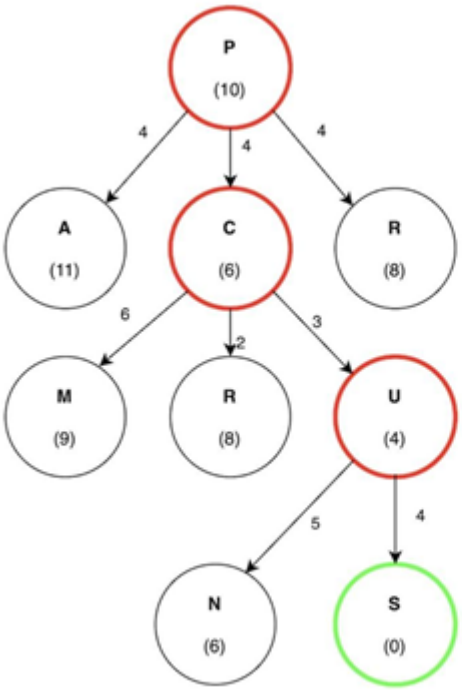
Hình 2.2 Minh họa bước 1 của thuật toán

Bước 2: Xét nút con mở rộng được đánh giá tốt nhất bởi hàm heuristic



Node [cost]
U [4]
R [8]
R [8]
M [9]
A [11]

Closed List
P
C



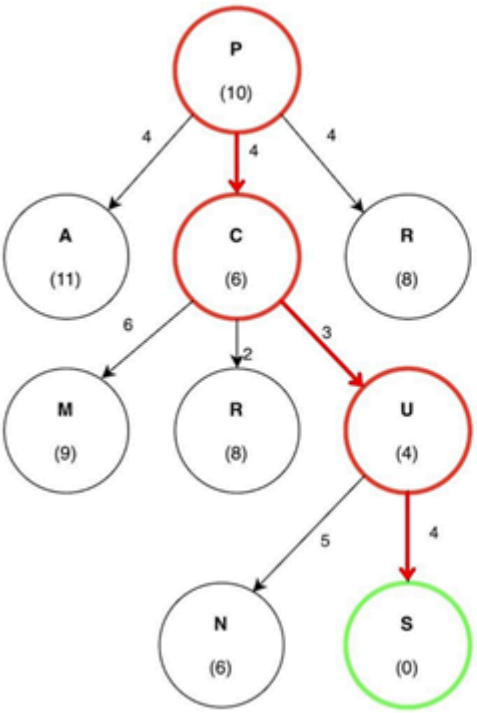
Node [cost]
S [0]
N [6]
R [8]
R [8]
M [9]
A [11]

Closed List
P
C
U

Hình 2.3 Minh họa bước 2 của thuật toán

Bước 3: Kết thúc thuật toán khi không còn nút con mở rộng để xét

Đây là kết quả tìm được của thuật toán Greedy Best First Search



Node [cost]
S [0]
N [6]
R [8]
R [8]
M [9]
A [11]

Closed List
P
C
U

Hình 2.4 Đường đi được thuật toán GBFS tìm ra

2.2. Áp dụng thuật toán GBFS vào bài toán Minimun Cost Flow

2.2.1. Input của bài toán

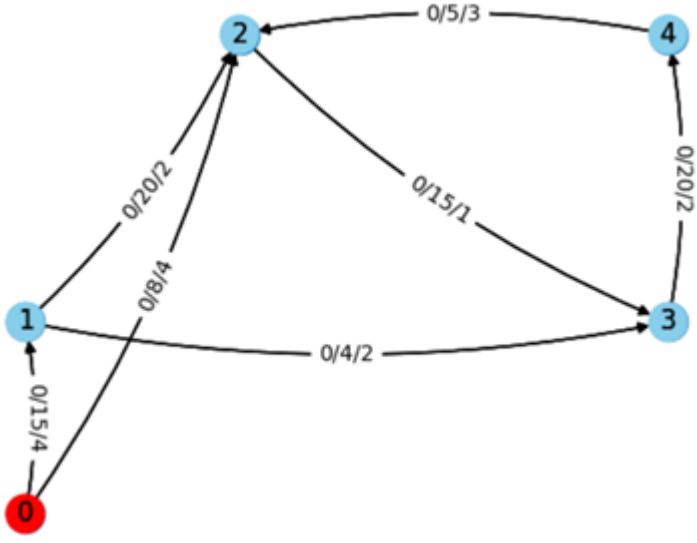
Đọc dữ liệu từ file "PY.txt". Trong file bao gồm:

- Dòng đầu tiên chứa 5 số nguyên không âm lần lượt là:
  - **vertex** – số đỉnh của đồ thị.
  - **edges** – số cạnh của đồ thị.
  - **flow** – lượng hàng cần vận chuyển.
  - **source** – đỉnh nguồn.
  - **sink** – đỉnh đích.
- **vertex** dòng tiếp theo, dòng thứ **i** chứa 2 số thực là tọa độ của đỉnh **i – 1.**
- **edges** dòng tiếp theo, mỗi dòng chứa 4 số nguyên, trong đó, 2 số nguyên đầu tạo thành một cạnh (u, v). Hai số nguyên tiếp theo lần lượt là **c(u, v)** và **cost(u, v)**.

<u>Ví dụ</u>:

Input:

5 7 12 0 4
0 0
0 2
2 5
6 2
6 5
0 1 15 4
0 2 8 4
1 2 20 2
1 3 4 2
2 3 15 1
3 4 20 2
4 2 5 3



Hình 2.5 Đồ thị được vẽ bởi dữ liệu ví dụ

2.2.2 Áp dụng thuật toán GBFS giải bài toán Minimum Cost Flow

Trong bài toán Minimum Cost Flow, ta cần tìm các đường đi trong mạng lưới từ nguồn (source) đến đích (sink) sao cho tổng chi phí vận chuyển là nhỏ nhất và đồng thời thỏa mãn các ràng buộc về khả năng chịu tải của các cạnh. Giải thuật Greedy Best First Search (GBFS) có thể được sử dụng để giải quyết bài toán theo các bước như sau:

Bước 1: Tìm đường đi có chi phí bé nhất từ đỉnh nguồn đến đỉnh đích trên đồ thị G bằng thuật toán GBFS:

1a. Tính heuristic tất cả các đỉnh trong đồ thị G. Hàm heuristic được tính bằng "khoảng cách đường chim bay" từ đỉnh đang xét đến đỉnh đích.

```
air = []

for i in range(vertex):

    kc = f'{sqrt((note_position[i][0] -
note_position[vertex-1][0])**2 +(note_position[i][1] -
note_position[vertex-1][1])**2):.2f}'

    air.append(float(kc))

air.append(10000)

air.append(0)
```

1b. Sử dụng một hàng đợi ưu tiên heapq. Xuất phát từ đỉnh nguồn, thêm đỉnh nguồn và "khoảng cách đường chim bay" từ đỉnh nguồn đến đỉnh đích vào heap.

1c. Lấy đỉnh có heuristic bé nhất ra khỏi heapq. Nếu đỉnh này đã được thăm, bỏ qua. Ngược lại, ta thăm đỉnh này và dò tất cả các đỉnh kề của nó, đỉnh kề nào chưa được thăm và có "giới hạn" của cạnh dương, ta thêm vào heapq đỉnh đó và heuristic của nó.

```
heap = [(air[source], source)]

visited =[0]*(vertex + 2)

while heap:

    d, node = heapq.heappop(heap)

    if visited[node] == 1:   continue

    visited[node] = 1

    for neighbor in graph[node]:

            if visited[neighbor] == 1: continue

                if graph[node][neighbor][0] > 0 :

                    prev[neighbor] = (node,
graph[node][neighbor][0], graph[node][neighbor][1])

                heapq.heappush(heap,(air[neighbor], neighbor))
```

1d. Lặp lại bước 1c cho đến khi hàng đợi heapq rỗng.

Bước 2: Nếu tìm thấy đường đi ngắn nhất có thể đi được từ nguồn đến đích, cập nhật luồng và dung lượng còn lại trên các cạnh trên đồ thị G theo luồng lớn nhất có thể được đưa qua đường đi đó. Tính tổng chi phí vận chuyển và tổng "lượng hàng" đã vận chuyển. Ngược lại thì kết thúc thuật toán.

```
if visited[sink] == 0:

    break

print('----------------')

node = sink

min_flow = float('inf')
```

```
path=[]
colored_edges_1 = []
while (node > 0):
    min_flow = min(min_flow, prev[node][1])
    node = prev[node][0]
    path.append(str(node))
    cost = min_flow * prev[node][2]
path.reverse()
print('-->'.join(path))
flow -= min_flow
total_flow+= min_flow
print('Max Flow:',min_flow)
node = sink
while (node != source):
    cost_path += min_flow * prev[node][2]
    total_cost += min_flow * prev[node][2]
    prev_node, _, _ = prev[node]
    temp = graph[prev_node][node]
    graph[prev_node][node] = (temp[0] - min_flow, temp[1])
    node = prev_node
```

Bước 3: Lặp lại 2 bước trên cho đến khi vận chuyển hết "lượng hàng" hoặc không còn đường nào để đi.

CHƯƠNG 3: THIẾT KẾ GIAO DIỆN

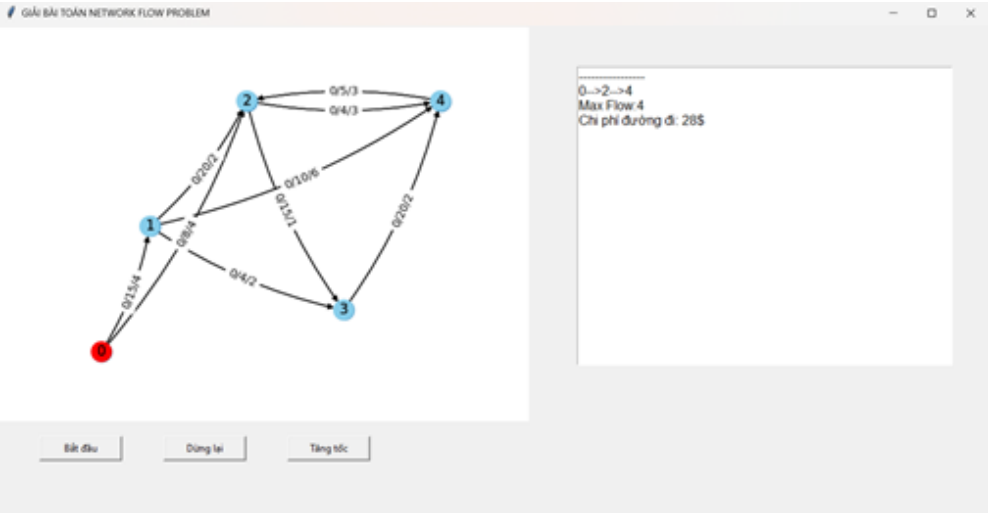
3.1. Màn hình chính

Khi khởi chạy chương trình, cửa sổ chính GIẢI BÀI TOÁN NETWORK FLOW PROBLEM xuất hiện với 3 button Bắt đầu, Dừng lại, Tăng tốc và ô cửa sổ nhỏ ở góc phải để thể hiện kết quả cuối cùng của bài toán.

Hình 3.1 Màn hình chính khi chạy chương trình

3.2. Giải thích các yếu tố xuất hiện trên đồ thị

- Các chỉ số a/b/c trên mỗi cạnh lần lượt là lưu lượng hiện tại, giới hạn của đường đi và chi phí vận chuyển khi đi qua đường đó.
- Các đỉnh chuyển từ xanh sang đỏ thể hiện trạng thái đang được "ghé thăm".
- Các cạnh màu đỏ biểu diễn cho đường đi tối ưu thuật toán tìm ra.

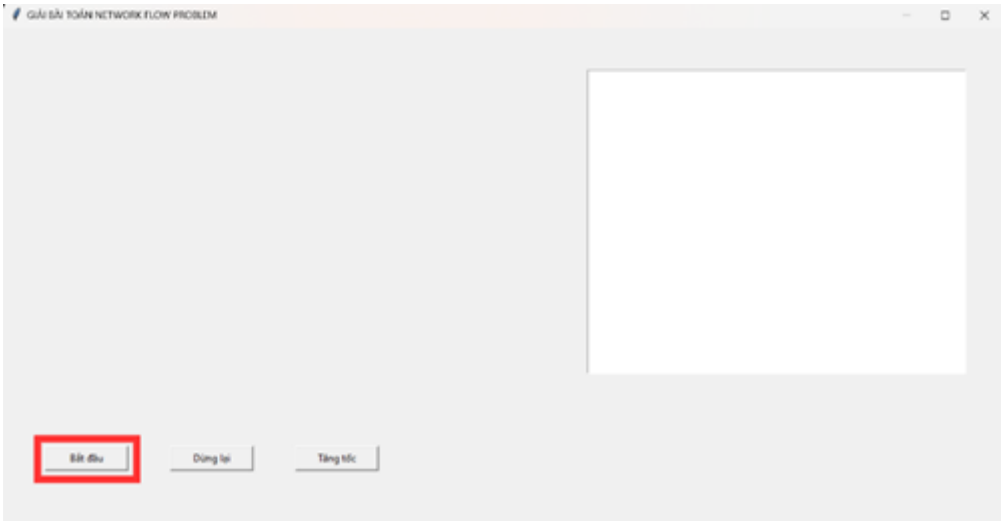


Hình 3.2 Minh họa các yếu tố xuất hiện trên đồ thị

3.3 Các bước tiến hành

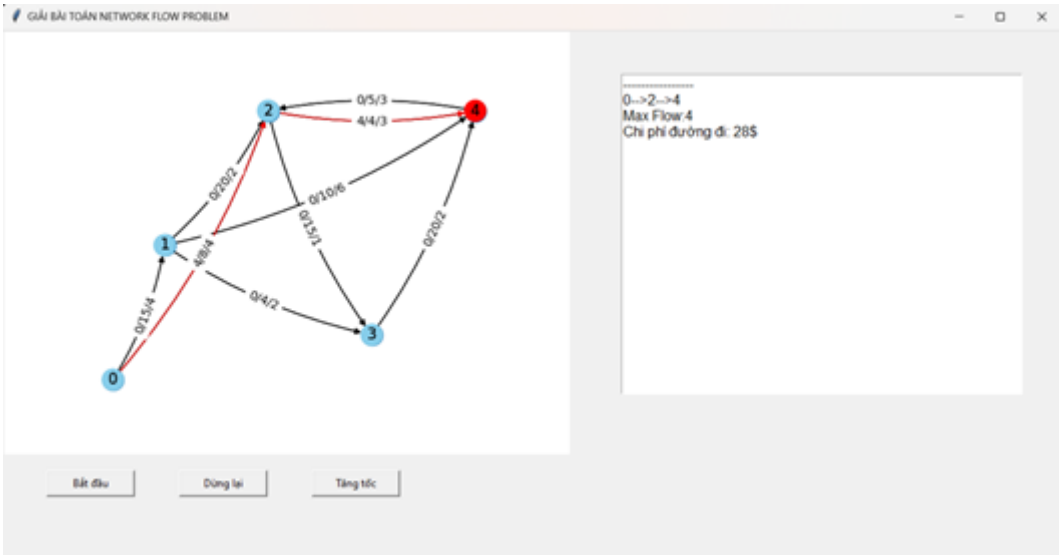
Ví dụ với một bài toán cụ thể:

Để hiển thị đồ thị và đường đi cùng kết quả của bài toán nhấn vào nút **Bắt đầu** ở cửa sổ.



Hình 3.3 Minh họa vị trí nút **Bắt đầu**

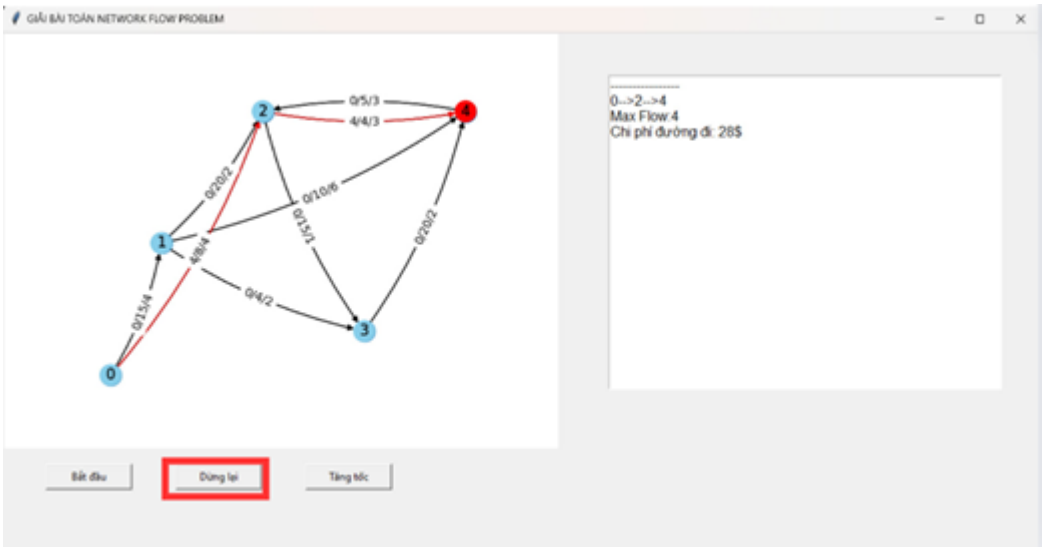
Sau khi nhấn vào nút bắt đầu, đường đi được tìm thấy sẽ hiện ở ô bên góc phải màn hình cùng kết quả Max Flow và chi phí của đường đi đó và đồ thị sẽ bắt đầu biểu diễn đường đi và cập nhật lưu lượng hiện tại cho mỗi đường được đi qua.



Hình 3.4 Minh họa màn hình chính sau khi nhấn nút **Bắt đầu**

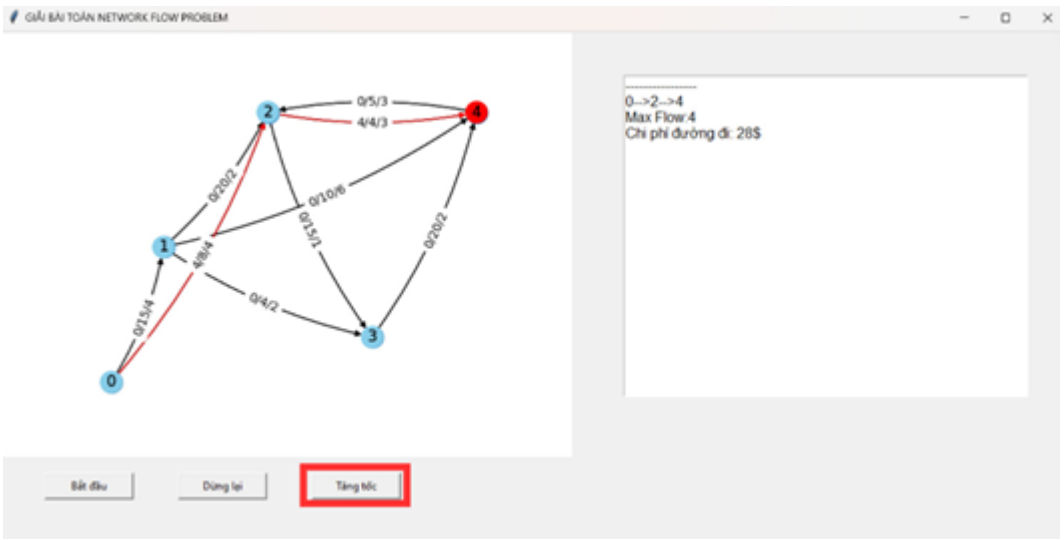
Ở bài toán này thì đồ thị có 5 đỉnh và các cạnh có trọng số như trong hình 3.4 (được giải thích ở phần 3.2). Các đường màu đỏ di chuyển từ đỉnh 0 đến đỉnh 4 thể hiện cho đường đi tối ưu được thuật toán tìm ra.

Đồ thị sẽ liên tục biểu diễn các đường đi tìm được cho đến khi bài toán kết thúc và đưa ra được kết quả cuối cùng. Vì vậy, để có thể quan sát dễ dàng hơn sự thay đổi của mỗi đường đi thì ta nhấn vào nút **Dừng lại** khi đó đồ thị sẽ ngừng biểu diễn. Và để tiếp tục biểu diễn đường đi tiếp theo ta nhấn vào nút **Bắt đầu**.



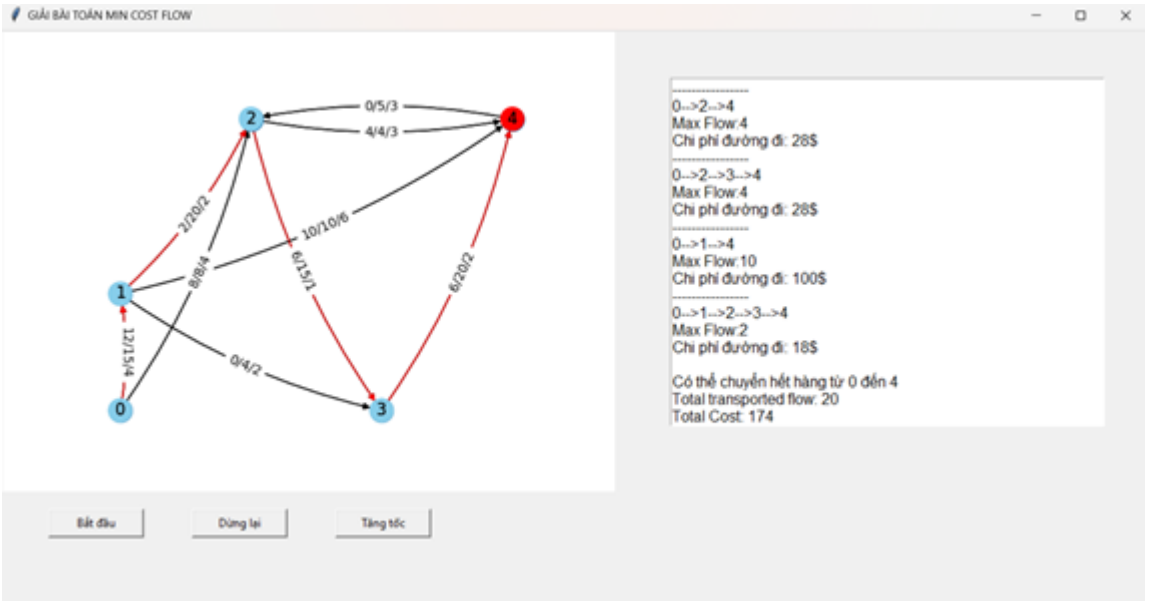
Hình 3.5 Minh họa vị trí nút **Dừng lại**

Các đường đi được biểu diễn theo một khoảng thời gian nhất định được cho trước, ta có thể tăng tốc độ biểu diễn của các đường đi bằng cách nhấn vào nút **Tăng tốc**



Hình 3.6 Minh họa vị trí nút **Tăng tốc**

Sau khi hoàn thành vận chuyển được hết lượng hàng từ đỉnh nguồn đến đỉnh đích hoặc không còn đường để đi thì bài toán sẽ kết thúc, đồ thị sẽ tự động ngừng chạy và cho kết quả như hình dưới. Khi muốn xem lại đồ thị, ta nhấn nút **Bắt đầu,** đồ thị sẽ biểu diễn lại các đường đi một lần nữa (nút **Bắt đầu** chỉ chạy lại đồ thị khi đã kết thúc bài toán).



Hình 3.7 Minh họa màn hình chính khi bài toán kết thúc

CHƯƠNG 4: THẢO LUẬN VÀ ĐÁNH GIÁ THUẬT TOÁN

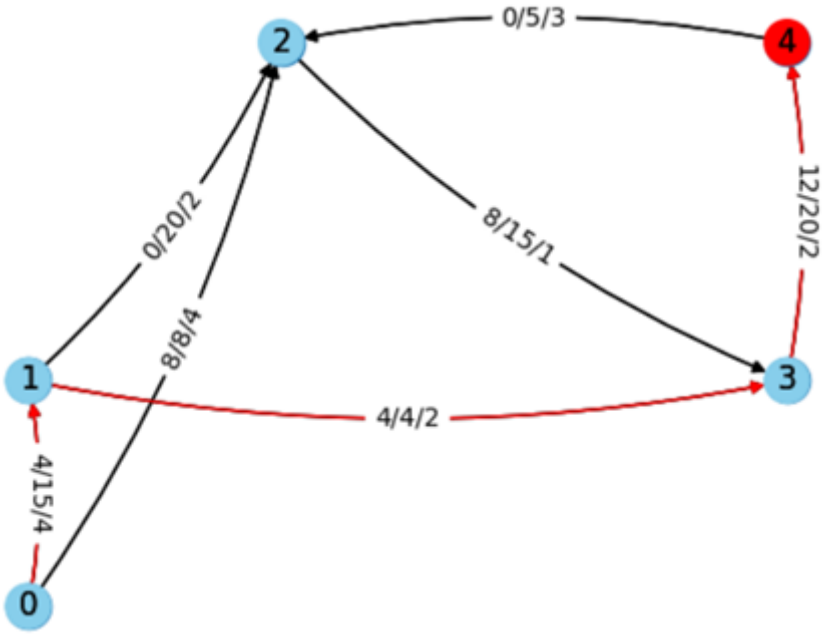
4.1. Đánh giá thuật toán.

Thuật toán Greedy Best First Search là một chiến lược tìm kiếm với tri thức bổ sung từ việc sử dụng các tri thức cụ thể của bài toán. Thuật toán sẽ sử dụng 1 hàm đánh giá là hàm heuristic h(n) . Hàm heuristic h(n) đánh giá chi phí để đi từ nút hiện tại n đến nút đích (mục tiêu), hàm heuristic đánh giá tốt sẽ giúp thuật toán chạy tối ưu hơn.

Trong bài toán của nhóm, h(n) được tính bằng "khoảng cách đường chim bay" từ node hiện tại đến node đích. Vì vậy, những node càng gần node đích thì chi phí để vận chuyển sẽ càng nhỏ.

4.1.1. Trường hợp 1: "Khoảng cách đường chim bay" phản ánh được đường đi từ đỉnh đó đến đỉnh đích.

Như hình 4.1, hàm heuristic đánh giá khá là tối ưu với trường hợp này. Với số đơn vị hàng cần vận chuyển là 12 thì GBFS cho kết quả tối ưu nhất (tức là chi phí thấp nhất khi chuyển 12 đơn vị hàng từ đỉnh 0 đến đỉnh 4). Đầu tiên, GBFS tìm được đường ngắn nhất là 0 -> 2 -> 3 -> 4 với lượng hàng lớn nhất có thể chuyển là 8, đường vận chuyển hàng ngắn nhất có thể tiếp theo là 0 -> 1 -> 3 ->4 chuyển được hết 4 đơn vị hàng còn lại với tổng chi phí vận chuyển là 88. Đây là chi phí thấp nhất (tối ưu nhất).

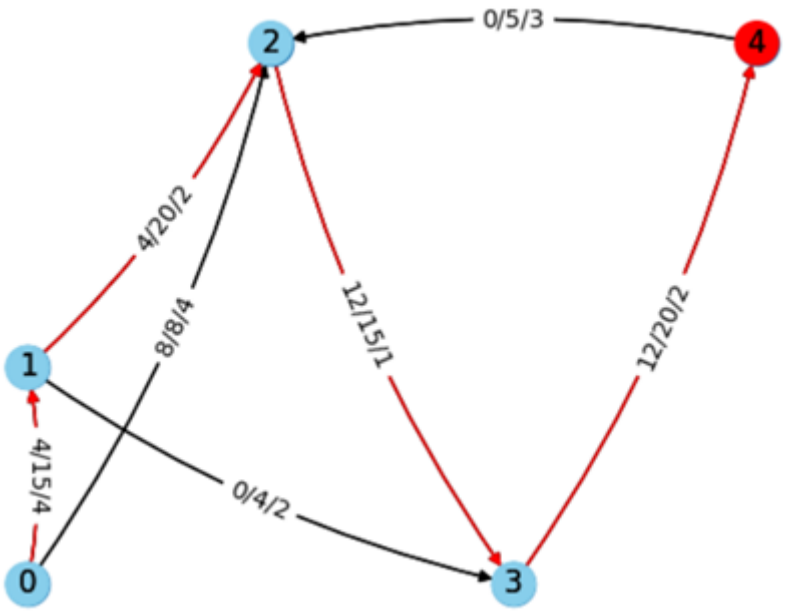


Hình 4.1 Đồ thị minh họa trường hợp 1

```
Kết quả bài toán:
---------------
0-->2-->3-->4
Max Flow: 8
Chi phí đường đi: 56$
---------------
0-->1-->3-->4
Max Flow: 4
Chi phí đường đi: 32$
Có thể chuyển hết hàng từ 0 đến 4
Total Flow: 12
Total Cost: 88
```

4.1.2. Trường hợp 2: "Khoảng cách đường chim bay" không phản ánh được đường đi từ đỉnh đó đến đỉnh đích.

Như hình 4.2, khi đỉnh số 3 bị dịch chuyển ra xa đỉnh số 4 hơn đỉnh số 2 và giữ nguyên các chỉ số khác, vì hàm heuristic đánh giá chưa được tối ưu nên khi duyệt tới đỉnh số 1 thì đỉnh số 2 sẽ được ưu tiên hơn (dù từ đỉnh số 2 đến đỉnh số 4 không có đường đi nên muốn đi từ đỉnh số 2 sang đỉnh số 4 vẫn phải đi qua đỉnh số 3), dẫn đến việc GBFS cho kết quả không tối ưu do tốn thêm chi phí để chuyển hàng từ đỉnh số 2 về đỉnh số 3, mặc dù cũng vận chuyển 12 đơn vị hàng. Vì vậy tổng chi phí sẽ là 92.

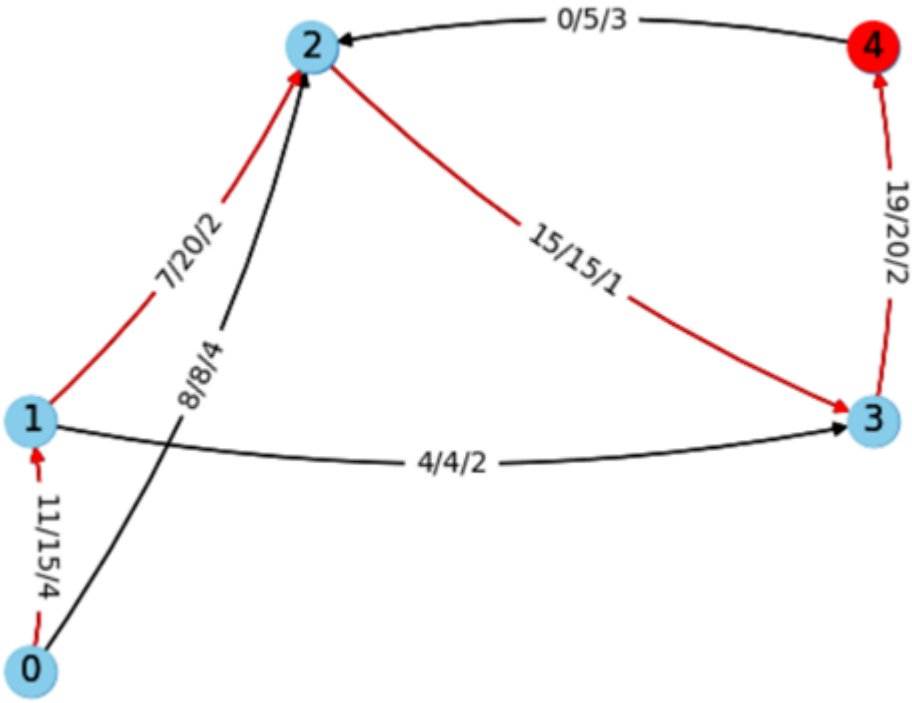


Hình 4.2 Đồ thị minh họa trường hợp 2

```
Kết quả bài toán:
----------------
0-->2-->3-->4
Max Flow: 8
Chi phí đường đi: 56$
----------------
0-->1-->2-->3-->4
Max Flow: 4
Chi phí đường đi: 36$
Có thể chuyển hết hàng từ 0 đến 4
Total Flow: 12
Total Cost: 92
```

4.1.3. Trường hợp 3: Chỉ chuyển được một phần hàng từ source đến sink.

Theo đồ thị ở hình 4.3, mặc dù từ đỉnh nguồn 0 có thể phát được 23 đơn vị hàng, nhưng chỉ có thể vận chuyển được tối đa 19 đơn vị hàng. Thuật toán GBFS tìm đường để vận chuyển nhiều lượng hàng cần chuyển nhất có thể với chi phí được coi là tối ưu.



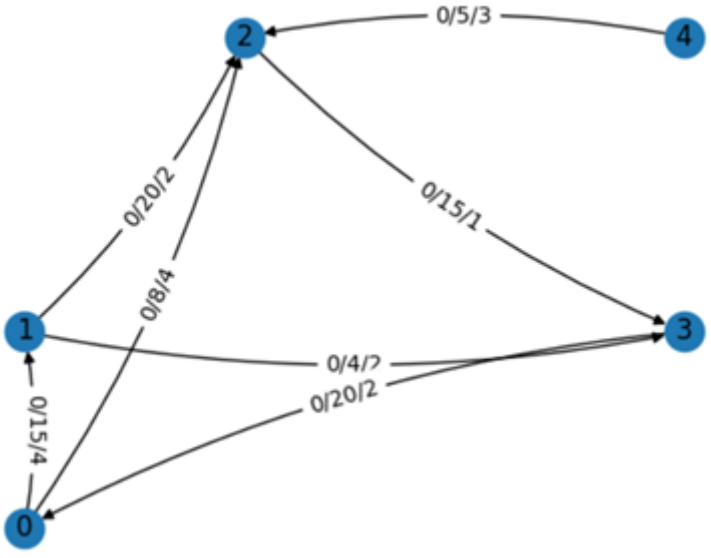
Hình 4.3 Đồ thị minh họa trường hợp 3

```
Kết quả bài toán:
----------------
0-->2-->3-->4
Max Flow: 8
Chi phí đường đi: 56$
----------------
```

```
    0-->1-->3-->4

    Max Flow: 4

    Chi phí đường đi: 32$

    ----------------

    0-->1-->2-->3-->4

    Max Flow: 7

    Chi phí đường đi: 63$

    Có thể chuyển 1 phần hàng từ 0 đến 4 và còn 6 đơn vị
hàng chưa chuyển

    Total Flow: 19

    Total Cost: 151
```

4.1.4. Trường hợp 4: Đồ thị không có đường đi từ source đến sink.

Có thể thấy ở hình 4.4, không có đường nào đi vào đỉnh đích (đỉnh số 4) nên thuật toán GBFS không thể tìm được đường từ đỉnh nguồn đến đỉnh đích để vận chuyển hàng.



Hình 4.4 Đồ thị minh họa trường hợp 4

```
Kết quả bài toán:
Không thể tìm đường để vận chuyển từ 0 đến 4
Total Flow: 0
Total Cost: 0
```

4.2. Đánh giá thuật toán GBFS khi giải bài toán Min Cost Flow.

4.2.1. Ưu điểm.

Greedy Best First Search tổ hợp các ưu điểm của phương pháp tìm kiếm theo chiều rộng và tìm kiếm theo chiều sâu và dùng tri thức của bài toán để dẫn dắt việc tìm kiếm nên tốc độ sẽ nhanh hơn so với các thuật toán tìm kiếm thông thường như Dijkstra hay A* vì nó chỉ cần xem xét các nút gần đích nhất mà không quan tâm đến chi phí thực sự của các nút đó.

Trong một số trường hợp thì GBFS có thể tìm ra giải pháp gần tối ưu với chi phí thấp hơn so với tìm kiếm đầy đủ.

Tính tham lam (Greedy): GBFS chọn nút tiếp theo dựa trên một hàm đánh giá cục bộ mà không cần phải xem xét toàn bộ không gian trạng thái, giúp giảm bớt độ phức tạp tính toán so với các phương pháp tìm kiếm toàn diện.

Tính hiệu quả: Do GBFS tập trung vào việc chọn lựa nút tiếp theo một cách tham lam, nó thường hoạt động hiệu quả trên các bài toán có không gian trạng thái lớn.

Dễ triển khai: Thuật toán GBFS có cấu trúc đơn giản, dễ triển khai và hiểu, giúp dễ dàng sử dụng và điều chỉnh cho các bài toán cụ thể.

4.2.2. Nhược điểm.

Thuật toán không có tính hoàn chỉnh vì có thể bị vướng vào các vòng lặp vô tận hoặc đi ra xa khỏi lời giải nếu không được cài đặt đúng cách hoặc không giới hạn về số lượng nút mở rộng.

Độ phức tạp về thời gian của thuật toán GBFS khi áp dụng giải bài toán MCF cao vì trong trường hợp tệ nhất, thuật toán sẽ duyệt một nhánh đến khi không còn đường đi tiếp. Lúc này, độ phức tạp về thời gian của GBFS giống như của thuật toán DFS.

Độ phức tạp về bộ nhớ của thuật toán GBFS cũng cao, vì trong khi tìm lời giải, thuật toán không được phép xóa các đường đã duyệt qua, lúc này độ phức tạp về bộ nhớ giống như thuật toán BFS.

Rủi ro rơi vào tối ưu cục bộ: Do GBFS chỉ tập trung vào việc chọn nút tiếp theo dựa trên hàm đánh giá cục bộ, nó có thể rơi vào tối ưu cục bộ và không thể đảm bảo tìm ra giải pháp tối ưu toàn cục.

Trong một số trường hợp, GBFS có thể phải "lạc hậu" (backtrack) để tìm kiếm một giải pháp tốt hơn, điều này có thể làm tăng độ phức tạp thời gian tính toán.

Không đảm bảo độ chính xác: GBFS không đảm bảo tìm ra giải pháp tối ưu toàn cục, đặc biệt là trên các đồ thị có cấu trúc phức tạp hoặc có nhiều hướng đi khả dĩ.

4.3. So sánh Greedy Best First Search với các giải thuật khác.

Thuật toán	Đảm bảo tối ưu	Khả năng xử lý cạnh âm	Linh hoạt với các bài toán khác
GBFS	Không	Không	Trung bình
Bellman-Ford	Có	Có	Trung bình
GA	Không	Không	Cao
Edmonds-Karp	Có	Không	Thấp
Cycle-Cancelling	Có	Có	Thấp
Primal-Dual	Có	Không	Cao

- Greedy Best-First Search (GBFS): Phù hợp cho các bài toán nhỏ và yêu cầu giải pháp nhanh chóng, nhưng không đảm bảo tối ưu.
- Bellman-Ford: Thích hợp cho các bài toán với cạnh có trọng số âm và yêu cầu giải pháp tối ưu, nhưng chậm với đồ thị lớn.
- Genetic Algorithm (GA): Linh hoạt và có khả năng tìm kiếm toàn cục, phù hợp với các bài toán phức tạp, nhưng không đảm bảo tối ưu và đòi hỏi nhiều tinh chỉnh.
- Edmonds-Karp: Đảm bảo tối ưu và tìm các đường tăng ngắn nhất, nhưng chậm với đồ thị có số lượng cạnh lớn.
- Cycle-Cancelling: Đảm bảo tối ưu và xử lý được cạnh có trọng số âm, nhưng tốn thời gian với đồ thị lớn.
- Primal-Dual: Đảm bảo tối ưu và hiệu quả cho nhiều loại bài toán, nhưng phức tạp trong triển khai và yêu cầu hiểu biết sâu về lý thuyết đối ngẫu.

CHƯƠNG 5: KẾT LUẬN

5.1. Những hạn chế và hướng phát triển

5.1.1. Những hạn chế

- **Không đảm bảo tìm ra giải pháp tối ưu:** GBFS không đảm bảo tìm ra giải pháp tối ưu vì nó chỉ tập trung vào việc di chuyển đến các nút gần mục tiêu nhất mà không xem xét các yếu tố khác như chi phí tổng cộng của đường đi.
- **Không hiệu quả cho các bài toán lớn:** Trong các bài toán có không gian trạng thái lớn, GBFS có thể không hiệu quả do phải duyệt qua nhiều nút mà không có sự cải thiện rõ rệt về chi phí.
- **Không xử lý được các hạn chế động:** GBFS không xử lý được các hạn chế động trong mạng, nơi các yếu tố như chi phí hoặc dung lượng có thể thay đổi theo thời gian.
- **Không phù hợp cho các bài toán đa mục tiêu:** Trong các bài toán có nhiều mục tiêu, GBFS có thể không hiệu quả vì không xem xét đến các mục tiêu khác nhau một cách cân đối.
- **Không tận dụng tốt thông tin từ các luồng hàng hóa khác nhau**: Trong MCF, việc xử lý đồng thời nhiều loại hàng hóa có thể cung cấp thông tin bổ sung giúp tối ưu hóa tổng thể. GBFS, với tính chất tham lam của mình, có thể bỏ qua hoặc không tận dụng tốt các thông tin này, dẫn đến các giải pháp không hiệu quả.

5.1.2. Hướng phát triển

- **Hàm đánh giá (heuristic) cải tiến:** Phát triển các phương pháp để đánh giá mục tiêu (heuristic) một cách hiệu quả và chính xác hơn. Điều này có thể bao gồm việc sử dụng hàm heuristic phức tạp hơn hoặc sử dụng các kỹ thuật học máy để học các heuristic từ dữ liệu.
- **Kết hợp với các phương pháp tìm kiếm khác:** Kết hợp GBFS với các phương pháp tìm kiếm khác như A*, để tận dụng lợi ích của cả hai phương pháp. Ví dụ, A* sử dụng thông tin về chi phí thực tế từ điểm bắt đầu đến điểm cuối, trong khi GBFS tập trung vào việc đưa ra ước lượng heuristics.
- **Đa mục tiêu và tìm kiếm đa dạng:** Mở rộng thuật toán để có thể xử lý các bài toán có nhiều mục tiêu hoặc tìm kiếm đa dạng. Điều này có thể đặt ra các thách thức về cách định nghĩa và ứng dụng các heuristic.
- **Tích hợp với học máy:** Sử dụng kỹ thuật học máy để cải thiện quá trình đưa ra quyết định trong GBFS, bằng cách học từ dữ liệu lịch sử hoặc thông tin môi trường.

- **Tối ưu hóa hiệu suất:** Tối ưu hóa thuật toán và cải thiện hiệu suất tính toán, đặc biệt là trong các bài toán có không gian trạng thái lớn.

5.2. Kết luận

Đề tài đã được nhóm trình bày chi tiết qua nội dung của 4 chương trên. Đầu tiên, nhóm trình bày về bài toán Network Flow cũng như các lý thuyết liên quan. Trong bài toán Network Flow có nhiều biến thể khác đã được nhóm liệt kê như: min cost flow, max flow, min cost max flow…Đối với đề tài này, nhóm chọn biến thể MCF để giải quyết. Ở phần tiếp theo, nhóm phát biểu bài toán MCF cũng như các thuật toán có thể dùng để tiếp cận bài toán. Chương 2, nhóm trình bày thuật toán GBFS về các chủ đề như: lý thuyết, bước thực hiện, đánh giá thuật toán và đưa ví dụ trực quan. Sau đó, nhóm áp dụng thuật toán GBFS vào để giải bài toán MCF và đưa ra các đánh giá, so sánh với các thuật toán khác.

Mong rằng bài nghiên cứu này có thể làm rõ hơn về bài toán NFP, thuật toán GBFS cũng như các biến thể của NFP và các thuật toán khác. Từ đó, giúp cho việc giải quyết các vấn đề đời sống, kinh tế và khoa học thêm đơn giản, hiệu quả hơn.

TÀI LIỆU THAM KHẢO

1. **Minimum-cost flow problem. (2024, May 20). Wikipedia.** https://en.wikipedia.org/wiki/Minimum-cost_flow_problem
2. **Spoj, T. (n.d.). Minimum-cost flow - Successive shortest path algorithm. Solution for SPOJ.**

   https://vnspoj.github.io/wiki/graph/min_cost_flow

3. ***AI | Search Algorithms | Greedy Best-First Search | Codecademy*. (2023, October 31). Codecademy.**

   https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search?fbclid=IwZXh0bgNhZW0CMTAAAR0Soxt58GpCZ_UDW7yPlYYIQDa0Qg_dgP-B9cIxLMSxX4aZRWJIvidKlag_aem_Afw9i0pHxjbe9rZAQoPON1zvFcnIWRvfX41mOm177DdOLgDkGNrHkNkZPfRp11O61DB2fI8sxopMwtrmN_6sMoYN

4. **H.M Channel. (2020, March 10). *Lý thuyết đồ thị - bài toán luồng trong mạng, luồng cực đại (bài 9)* [Video]. Youtube.**

   https://www.youtube.com/watch?v=3dOQ3IoVmvY

5. **15-451: Algorithm Design and Analysis. (2020, Fall). Carnegie Mellon University**

   https://www.cs.cmu.edu/~15451-f23/lectures/lecture12-flow-i.pdf

6. **Network Flows: Theory, Algorithms, and Applications by Ravindra K. Ahuja (1993) Thomas L. Magnanti, and James B. Orlin**

   https://www.dl.behinehyab.com/Ebooks/NETWORK/NET005_354338_www.behinehyab.com.pdf?fbclid=IwZXh0bgNhZW0CMTAAAR1QaUH3OAmo9OYKdM5BTxLSW-7UFfNJBReFXpOytuQia-Sar9t1NHlmbgE_aem_AapMXQE6_1eGrwBs5_JypN1_7tMBk2CVUP7p3NFsDIpv75_2rIgpCwjn-DtCILUFxFCKwuWq1tNPfiYo7p4943RD

7. **GeeksforGeeks. (2024b, April 30). *Greedy Algorithm Tutorial Examples, Application and Practice Problem*. GeeksforGeeks.** https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

PHỤ LỤC

Phụ lục 1. Hướng dẫn chạy dự án

- Source code toàn bộ dự án: **Link GitHub**
- Thư viện cần cài đặt: thư viện **tkinter, networkx, matplotlib, math.**
- Download flie GBFS_MCF.ipynb và file PY.txt (file input)
- Có thể thay đổi đồ thị bài toán muốn giải bằng cách đổi input theo cấu trúc như mục 2.2.1 đã trình bày hoặc các input mẫu của các trường hợp ở phần phụ lục 3.
- Đầu tiên, dẫn nguồn file PY.txt được tải về để đọc vào lưu vào biến f.

```
from math import sqrt
f = open("D:\\PY.txt","r")
read = f.readline()
read=read.split()
vertex = int(read[0])
edges = int(read[1])
flow = int(read[2])
source = int(read[3])
sink = int(read[4])
```

Hình 5: Minh họa vị trí của biến f

- Tiếp theo chạy tất cả các ô code (hoặc run all) để chạy code và xuất hiện giao diện.

- Nhấn "Bắt đầu" để giải bài toán.
- "Dừng lại" để dừng lại tại vị trí mong muốn và "Bắt đầu" để tiếp tục giải bài toán.
- Nhấn "Tăng tốc" để tăng tốc giải bài toán.
  - Ô bên phải là nơi hiển thị những đường đi tìm đc để vận chuyển và kết quả thu được.

Phụ lục 2. Phân công nhiệm vụ.

Tên thành viên	**Nội dung thực hiện**	**Hoàn thành**
Nguyễn Bảo Cát Minh	- Tìm hiểu, phát biểu bài toán và các dạng toán, thuật toán liên quan. - Tìm hiểu và xây dựng báo cáo. - Thiết kế giao diện.	100%
Phạm Bằng	- Thiết kế code giao diện. - Tìm hiểu các thuật toán liên quan. - Tìm hiểu và xây dựng báo cáo.	100%
Nguyễn Trần Thế Anh	- Tìm hiểu thuật toán, thiết kế code. - Tìm hiểu các trường hợp khi áp dụng thuật toán GBFS để giải MCF. - Tìm hiểu và xây dựng báo cáo.	100%

Nguyễn Thành Vinh	- Phân chia nhiệm vụ. - Tìm hiểu bài toán và các thuật toán liên quan. - Tìm hiểu thuật toán, thiết kế code.	100%

Phụ lục 3. Input của các trường hợp

INPUT Trường hợp 1: "Khoảng cách đường chim bay" phản ánh được đường đi từ đỉnh đó đến đỉnh đích.	5 7 12 0 4 0 0 0 2 2 5 6 2 6 5 0 1 15 4 0 2 8 4 1 2 20 2 1 3 4 2 2 3 15 1 3 4 20 2 4 2 5 3

INPUT Trường hợp 2: “Khoảng cách đường chim bay” không phản ánh được đường đi từ đỉnh đó đến đỉnh đích	5 7 12 0 4 0 0 0 2 2 5 4 0 6 5 0 1 15 4 0 2 8 4 1 2 20 2 1 3 4 2 2 3 15 1 3 4 20 2 4 2 5 3
INPUT Trường hợp 3: “Không thể chuyển hết hàng từ source đến sink”	5 7 25 0 4 0 0 0 2 2 5 6 2 6 5 0 1 15 4 0 2 8 4 1 2 20 2 1 3 4 2 2 3 15 1 3 4 20 2 4 2 5 3

INPUT Trường hợp 4: “Không tồn tại đường đi từ source đến sink”	5 7 12 0 4 0 0 0 2 2 5 6 2 6 5 0 1 15 4 0 2 8 4 1 2 20 2 1 3 4 2 2 3 15 1 3 0 20 2 4 2 5 3