ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

KHOA Toán - Tin Học

~~~~~~*~~~~~~

BÁO CÁO THỰC HÀNH NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Sinh viên	**:**	**VÒNG VĨNH PHÚ**
MSSV	**:**	**19110413**
Môn Học	**:**	**NHẬP MÔN TRÍ TUỆ NHÂN TẠO**
Trường	**:**	**ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 NĂM 2021
~~~~~~

Nội dung

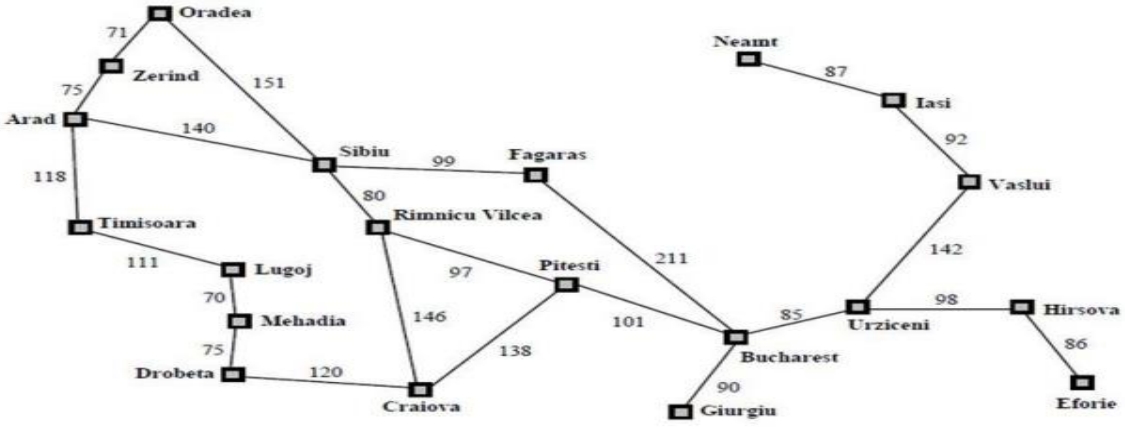
1. Ý TƯỞNG BÀI TOÁN

Đề bài :

- Cài đặt thuật toán Greedy – Best – First search để tìm đường đi từ Arad Hirsova như hình với h(n) được xác định như sau
- Cài đặt thuật toán A* để tìm đường đi ngắn nhất từ Arad tới Hirsova như hình với hàm h(n) được xác định như trong bài tập 1 và g(n) là khoảng cách giữa 2 thành phố

h(Arad) = 366	h(Hirsova) = 0	h(Rimnicu Vilcea) = 193
h(Bucharest) = 20	h(Iasi) = 226	h(Sibiu) = 253
h(Craiova)= 160	h(Lugoj) = 244	h(Timisoara)=329
h(Drobeta) = 242	h(Mehadia)=241	h(Urziceni) = 10
h(Eforie) = 161	h(Neamt) = 234	h(Vaslui) = 199
h(Fagaras) = 176	h(Oradea) = 380	h(Zerind) = 374
h(Giurgiu) = 77	h(Pitesti) = 100	

Bảng heuristic function



Đồ thị đường đi

- **Tìm đường đi dùng chi phí heuristic function**
  - Là kỹ thuật tìm kiếm dựa vào kinh nghiệm và sự hiểu biết của chúng ta về vấn đề cần giải quyết để xây dựng nên hàm đánh giá hướng dẫn sự tìm kiếm.

- **Giải pháp**
  - Ta sử dụng thuật toán Greedy Best First Search và tìm kiếm A* để giải quyết tìm kiếm đường đi theo chi phí và theo heuristic function

2. Greedy Best First Search

- **Ý tưởng thuật toán**
  - GBFS mở rộng nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến lời giải một cách nhanh nhất. Đánh giá chi phí của các nút chỉ dựa trên hàm heuristic function: F(n) = H(n).
  - Nhận xét : Về ý tưởng nó sẽ gần giống với tìm kiếm UCS nhưng ta sẽ phải tìm đánh giá tri phí dựa theo hàm H(n) thay vì total cost nhỏ nhất

- **Input/Output**
  - Input : (Graph, Heuristic Function, Start, Goal)
    - Graph : Đồ thị có dạng dictionary danh sách kề cho biết vị trí thành thành phố "A" có thể đi đến những thành phố nào với chi phí đường đi G(n) (GBFS ta sẽ không sử dụng đến chi phí này) theo dạng

      {"Thành phố đang xét": {"Điểm đến A": Chi phí },

      {"Điểm đến B: Chi phí},…}

    - Heuristic Function : Danh sách có dạng dictionary cho biết giá trị H(n) của các thành phố theo bảng Heuristic Function
    - Start : Vị trí điểm bắt đầu (Arad)

        - Goal : Vị trí kết thúc (Hirsova)
    - Output : Xuất ra thông tin đường đi từ nơi bắt đầu đến điểm kết thúc

- **Xây dựng Hàm và thuật toán**
    - Khởi tạo list queue (Open List)
    - Khởi tạo dictionary map (Close List) cho biết thành phố có hàm heuristic cost và khởi tạo thành phố đó đi bởi thành phố nào set khởi đầu các đi điểm có thể đi đến là none để ta có thể lưu vết các thông tin thành phố có thể đến thành phố đang xét city, map có dạng:
      {city:{'from': None, 'heuristic': Heuristic Function[city]}}
    - Tiến hành thực hiện vòng lặp để truy vết đến đích
        - Ta lấy thành phố có heuristic cost nhỏ nhất từ queue để xét (curCity) nếu thành phố đó bằng goal thì tìm ra đường đi nếu không ta tiếp tục xét
        - Tạo vòng lặp để explored các thông tin thành phố lấy từ graph bắt đầu từ curCity cho biết các thành phố curCity có thể đi đến
            - Nếu thành phố chưa được explored bằng các thành phố khác (map[city][‘from’] = none) ta sẽ thêm nó vào queue và thêm nó vào map bằng cách thêm curCity vào thông tin của ‘from’ trong map của thành phố explored (map[city][‘from’] = curCity) để có thể dễ dàng truy vết thông tin như đã đề cập trên
        - Nếu queue này rỗng có nghĩa là không có tìm thấy đường đi ta trả về kết quả không có kết quả
        - Sắp xếp queue này có phần tử heuristic cost nhỏ nhất trong hàng đợi để ta có thể lấy ra đầu tiên để xét sớm nhất có thể đến được goal bằng chi phí heuristic cost nhỏ nhất (Bước đầu vòng lặp)

- Nếu tìm thấy đường đi
  - Ta sẽ lưu curCity vào path làm phần từ đầu tiên
  - Và truy vấn ngược CurCity đến thông tin đường đi theo thông tin của ‘from’ đã lưu đến khi curCity = Start lúc này path là một list lưu các thông tin từ goal → start
  - Ta sẽ sử dụng hàm reserve của lớp list để path trả về start → goal

- Kết thúc thuật toán

3. A-Star (A*)

- **Ý tưởng thuật toán**
  - Thuật toán đánh giá 1 nút dựa trên chi phí đi từ nút gốc đến nút đó (G(n)) cộng với chi phí từ nút đó đến nút đích (H(n)).

$$F(n) = G(n) + H(n)$$

  - Hàm h(n) được gọi là chấp nhận được nếu với mọi trạng thái n, H(n) ≤ độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đi
  - Nhận xét : A* là thuật toán kết hợp giữa
- **Input/Output**
  - Input : (Graph, Heuristic Function, Start, Goal)
    - Graph : Đồ thị có dạng dictionary danh sách kề cho biết vị trí thành thành phố “A” có thể đi đến những thành phố nào với chi phí đường đi G(n) theo dạng

      {“Thành phố đang xét”: {“Điểm đến A”: Chi phí },

      {“Điểm đến B: Chi phí},…}

        - Heuristic Function : Danh sách có dạng dictionary cho biết giá trị H(n) của các thành phố theo bảng Heuristic Function
        - Start : Vị trí điểm bắt đầu (Arad)
        - Goal : Vị trí kết thúc (Hirsova)
    - Output : Xuất ra thông tin đường đi từ nơi bắt đầu đến điểm kết thúc, total cost

- **Xây dựng Hàm và thuật toán**
    - Khởi tạo list queue (Open List)
    - Khởi tạo dictionary map (Close List) cho biết thành phố có total cost (f(n)) (lúc đầu chỉ có heuristic cost (h(n)) do chưa biết các đoạn liên thông đường đi khi ta biết liên thông giữa 2 thành phố thì ta sẽ cộng thêm chi phí đường đi (g(n)) cho total cost) và khởi tạo thành phố đó đi bởi thành phố nào set khởi đầu các đi điểm có thể đi đến là none để ta có thể lưu vết các thông tin thành phố có thể đến thành phố đang xét city, map có dạng:

      {city:{'from': None(city),
            'total_cost': F(n) }}
    - Tiến hành thực hiện vòng lặp để truy vết đến đích
        - Ta lấy phần tử nhỏ nhất trong queue để tiến hành xét trạng thái của curCity nếu curCity bằng goal thì ta trả về đường đi còn không thì ta tiếp tục xét
        - Ta xét chi phí từ nơi bắt đầu đến start đến curCity bằng cách trừ chi phí H(curCity) để được chi phí từ start đến curCity (G(n) mới được cập nhật)
        - Tạo vòng lặp để explored các thông tin thành phố lấy từ graph bắt đầu từ curCity cho biết các thành phố curCity có thể đi đến

- Lấy thông tin total cost của thành phố explored đang xét hiện tại nếu chưa xét bao giờ thì nó là heuristic cost (h(n)) còn nếu đã xét thì nó là tổng cost của được tính theo công thức F(n) = G(n) + H(n)
- Totalcost là tổng chi phí từ nơi bắt đầu đến city explored được tính bằng cách tổng chi phí của curCity đến (city explored + Start đến curCity)(G(n)) + H(city)
- Nếu thành phố chưa được explored bao giờ hoặc chi phí của city explored dù đã được explored từ trước có Totalcost của thành phố explored đang xét lớn hơn total cost start đến city explored ta sẽ cập nhật lại vết đường đi ('from') và total cost (F(n)) cho map (lưu ý khi ta nói "cập nhật" lại map nghĩa là có thể đã tồn tại thành phố đó trong queue nên ta cần xét xem có tồn tại phần tử đó trong queue không để ta cần xoá bớt dữ liệu cũ và thêm vào queue phần tử mới để tránh trùng lặp vì bản chất queue là một tập hợp ta chỉ sử dụng class list để dễ dàng xử lý thay vì class set)

- Nếu queue này rỗng có nghĩa là không có tìm thấy đường đi ta trả về kết quả không có kết quả
- Sắp xếp queue này có phần tử Total cost nhỏ nhất trong hàng đợi để ta có thể lấy ra đầu tiên để xét sớm nhất có thể đến được goal bằng chi phí Total cost nhỏ nhất (Bước đầu vòng lặp)
- Nếu tìm thấy đường đi
  - Ta sẽ lưu curCity vào path làm phần từ đầu tiên và totalcost cho biến Cost để có thể trả về giá trị đường đi

- Truy vấn ngược CurCity đến thông tin đường đi theo thông tin của ‘from’ đã lưu đến khi curCity = Start lúc này path là một list lưu các thông tin từ goal → start
- Ta sẽ sử dụng hàm reserve của lớp list để path trả về start → goal

- Kết thúc thuật toán

4. Hàm hỗ trợ

- **Các hàm sử dụng trong bài**
  - Quicksort(array,Map, sortBy)
    - Đây là hàm thuật toán sort mảng QuickSort (có thể sử dụng các thuật toán khác) truyền vào tham số array, Map và sortBy để cho ta biết ta phải sort như thế nào vì list queue trong bài ta cần lấy phần tử nhỏ nhất ra trước nhưng trong list không tồn tại dữ liệu đường đi hay dữ liệu chi phí để sử dụng priorityqueue mà chỉ có tên các thành phố ta đã xét nên ta phải tự tạo một hàm sort riêng để xét các thông tin thành phố này bằng trong queue mà Map ta đã cập nhật đầy đủ thông tin như total cost (Heuristic cost) đã explored và cập nhật với sắp xếp theo thứ tự tăng dần theo sortBy