ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
Khoa Công nghệ Thông tin

BÁO CÁO ĐỒ ÁN CƠ SỞ TRÍ TUỆ NHÂN TẠO
LAB 01: SEARCH

Sinh viên thực hiện: Võ Văn Nam
Mã số sinh viên: 22120222
Lớp: CQ2022/22

MỤC LỤC

1. THÔNG TIN SINH VIÊN

- Họ và tên: Võ Văn Nam
- Mã số sinh viên: 22120222
- Lớp: CQ2022/22

2. MỨC ĐỘ HOÀN THÀNH

- Điểm tự đánh giá: 10/10

3. LÝ THUYẾT

a. Thuật toán BFS

- Khái niệm: BFS (Breadth First Search) là một thuật toán duyệt đồ thị và tìm kiếm đơn giản. Thuật toán bắt đầu từ một nút, sau đó duyệt tất cả các nút lân cận của nút đó trước. BFS có thể sử dụng để tìm đường đi ngắn nhất trong đồ thị không có trọng số.
- Độ phức tạp thời gian: $O(b^d)$, trong đó b là số nhánh tối đa của cây, d là chiều sâu thấp nhất của lời giải.
- Độ phức tạp không gian: $O(b^{d-1})$ cho tập mở và $O(b^d)$ cho biên.
- Tính chất:
    - BFS có thể tìm được đường đi tối ưu trong đồ thị không có trọng số.
    - BFS cần lưu trữ nhiều nút, độ phức tạp về không gian lớn nên sẽ kém hiệu quả trong trường hợp đồ thị có nhiều nút.

b. Thuật toán DFS

- Khái niệm: DFS (Depth First Search) là một thuật toán duyệt đồ thị và tìm kiếm đơn giản. Thuật toán bắt đầu tại một nút, sau đó duyệt các nút kế tiếp càng xa càng tốt. Sau khi duyệt hết một nhánh, ta quay lại và duyệt những nhánh còn lại.
- Độ phức tạp thời gian: $O(b^m)$, trong đó b là số nhánh tối đa của cây, m là chiều sâu tối đa của không gian tìm kiếm.
- Độ phức tạp không gian: $O(bm)$.
- Tính chất:
    - DFS có thể tìm được đường đi, nhưng có thể đường đi đó không phải là ngắn nhất.
    - DFS có thể sử dụng ít bộ nhớ hơn BFS trong trường hợp chiều sâu cây tìm kiếm nhỏ hơn chiều rộng.

c. Thuận toán UCS

- Khái niệm: UCS (Uniform-cost Search) tương tự với thuật toán Dijkstra, thuật toán sẽ mở nút n với chi phí đường đi $g(n)$ thấp nhất. UCS được sử dụng để tìm đường đi ngắn nhất trong đồ thị có trọng số không âm.
- Độ phức tạp thời gian: $O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$, với b là số nhánh tối đa của cây, ϵ là chi phí di chuyển thấp nhất, C^* là chi phí lời giải tối ưu.
- Độ phức tạp không gian: : $O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$.
- Tính chất:
    - UCS đảm bảo tìm kiếm được giải pháp tối ưu trong đồ thị có tất cả các cạnh có trọng số không âm.
    - Trong các đồ thị lớn có nhiều nút, UCS có thể hoạt động kém hiệu quả, độ phức tạp về không gian lớn do cần lưu trữ nhiều nút.

d. Thuật toán GBFS

- Khái niệm: GBFS (Greedy Best-First Search) là thuật toán tìm kiếm tham lam. GBFS ưu tiên mở các nút mà ước lượng khoảng cách từ nút đó đến đích là gần nhất trước.
- Độ phức tạp thời gian: $O(b^m)$, trong đó b là số nhánh tối đa của cây, m là chiều sâu tối đa của không gian tìm kiếm.
- Độ phức tạp không gian: : $O(b^m)$. GBFS giữ hết các nút trong bộ nhớ.
- Tính chất:
    - GBFS là một thuật toán Best-first Search, đánh giá trạng thái hiện tại chỉ dựa vào hàm heuristic $f(n) = h(n)$.
    - GBFS tìm kiếm nhanh nếu hàm heuristic đủ tốt, tuy nhiên không đảm bảo tìm được đường đi ngắn nhất.
    - Để đạt được hiệu quả tìm kiếm cao, GBFS cần có hàm đánh giá $h(n)$ chính xác.

e. Thuật toán A* Search

- Khái niệm: A* (A Star Search) là thuật toán tìm kiếm Best-first Search phổ biến nhất. A* sử dụng hàm đánh giá $f(n) = g(n) + h(n)$, trong đó
    - $g(n)$ là chi phí đường đi từ nút ban đầu đến nút n.
    - $h(n)$ là ước tính khoảng cách từ nút n đến đích.
    - $f(n)$ là ước tính chi phí để đến đích.

- Độ phức tạp thời gian: phụ thuộc vào hàm heuristic $h(n)$. Trong trường hợp xấu, độ phức tạp thời gian của A* có thể là hàm mũ.
- Độ phức tạp không gian: tương tự như độ phức tạp thời gian.
- Tính chất:
    - A* tích hợp heuristic vào quá trình tìm kiếm, tránh các đường đi có chi phí lớn.
    - A* cần một hàm heuristic $h(n)$ hợp lý, không ước lượng quá cao chi phí đến đích. Gọi $h^*(n)$ là chi phí thấp nhất từ nút n đến đích, thì $h(n) \leq h^*(n)$ là một heuristic hợp lý.
    - Nếu hàm heuristic hợp lý, A* sẽ luôn tìm thấy đường đi ngắn nhất nếu đường đi đó có trong đồ thị.
    - UCS hay thuật toán Dijkstra là một trường hợp đặc biệt của thuật toán A* với hàm heuristic $h(n) = 0$.
    - Trong các bài toán thực tế, A* rất hiệu quả trong việc tìm kiếm đường đi ngắn nhất.

4. SO SÁNH THUẬT TOÁN UCS VÀ A*

Tiêu chí so sánh	Uniform-cost Search	A* Search
Phân loại	Tìm kiếm không định hướng	Tìm kiếm có định hướng
Nguyên tắc	Đánh giá dựa trên chi phí đường đi đã đi từ nút khởi đầu đến nút hiện tại.	Đánh giá dựa trên tổng chi phí đường đi từ nút khởi đầu đến nút hiện tại và chi phí ước lượng từ nút hiện tại đến đích.
Hàm đánh giá	$f(n) = g(n), h(n) = 0$	$f(n) = g(n) + h(n)$
Điều kiện	Nếu tất cả các cạnh của đồ thị đều có trọng số không âm, UCS sẽ tìm kiếm được đường đi ngắn nhất.	Nếu hàm heuristic $h(n)$ là hợp lý, A* sẽ tìm kiếm được đường đi ngắn nhất.
Hiệu suất	Thường tìm kiếm nhanh hơn UCS nếu có hàm heuristic hợp lý vì giảm được số nút cần mở rộng.	Có thể chậm hơn A* vì không sử dụng heuristic, số nút cần mở rộng nhiều hơn.
Ứng dụng	Thích hợp trong các bài toán tìm kiếm đường đi	Thích hợp trong các bài toán thực tế như tìm đường đi

	ngắn nhất trong các đồ thị có trọng số không âm.	ngắn nhất trong bản đồ, tìm lời giải trong mê cung.

5. CÁC TEST CASE SỬ DỤNG

- Mỗi test case bao gồm:
    - Input: dòng đầu tiên là 2 số nguyên không âm, số thứ nhất là nút bắt đầu, số thứ hai là nút đích. Các dòng tiếp theo là các dòng của ma trận kề, chứa khoảng cách giữa các đỉnh.
    - Output: Gồm có visited và path. Visited là một dictionary chứa các nút đã được duyệt: mỗi khóa là một nút đã được duyệt, mỗi giá trị là nút liền kề với khóa và đã được duyệt trước nút khóa. Path là một list, biểu diễn đường đi tìm được.

a. Test case 1

- Input:

```
1 6
0 0 9 3 0 0 9 0
0 0 8 5 5 7 6 7
6 9 0 4 2 5 6 8
5 1 2 0 2 4 7 9
0 3 6 8 0 6 6 5
0 6 3 9 9 0 4 0
5 1 8 1 5 3 0 5
0 3 9 8 3 0 7 0
```

- Output:
    - BFS: {1: None, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1}, [1, 6]
    - DFS: {1: None, 7: 1, 6: 7}, [1, 7, 6]
    - UCS: {1: None, 3: 1, 4: 1, 6: 1}, [1, 6]
    - GBFS: {1: None, 3: 1, 2: 3, 4: 3, 5: 3, 6: 5}, [1, 3, 5, 6]
    - A*: {1: None, 3: 1, 4: 1, 6: 1}, [1, 6]

b. Test case 2

- Input:

```
4 0
0 3 7 6 9 0
3 0 8 0 3 6
5 7 0 4 0 0
9 0 8 0 6 0
```

7 2 0 2 0 1
0 4 0 0 8 0

- Output:
  - BFS: {4: None, 0: 4}, [4, 0]
  - DFS: {4: None, 5: 4, 1: 5, 2: 1, 3: 2, 0: 3}, [4, 5, 1, 2, 3, 0]
  - UCS: {4: None, 5: 4, 1: 4, 3: 4, 0: 1}, [4, 1, 0]
  - GBFS: {4: None, 5: 4, 1: 4, 3: 4, 0: 1}, [4, 1, 0]
  - A*: {4: None, 3: 4, 5: 4, 1: 4, 0: 1}, [4, 1, 0]

c. Test case 3

- Input:

  0 5
  0 1 0 0 0 12
  0 0 3 1 0 0
  0 0 0 0 3 0
  0 0 0 0 1 2
  0 0 0 0 0 3
  0 0 0 0 0 0

- Output:
  - BFS: {0: None, 1: 0, 5: 0}, [0, 5]
  - DFS: {0: None, 5: 0}, [0, 5]
  - UCS: {0: None, 1: 0, 3: 1, 4: 3, 2: 1, 5: 3}, [0, 1, 3, 5]
  - GBFS: {0: None, 1: 0, 3: 1, 4: 3, 5: 3}, [0, 1, 3, 5]
  - A*: {0: None, 1: 0, 3: 1, 4: 3, 5: 3}, [0, 1, 3, 5]

d. Test case 4

- Input:

  0 6
  0 4 7 0 0 0 0 0 0 0
  4 0 3 0 6 0 0 0 0 3
  7 3 0 7 0 3 0 0 0 0
  0 0 7 0 10 8 0 0 10 0
  0 6 0 10 0 0 3 8 5 0
  0 0 3 8 0 0 2 0 0 0
  0 0 0 0 3 2 0 0 0 5
  0 0 0 0 8 0 0 0 9 0
  0 0 0 10 5 0 0 9 0 0

0 3 0 0 0 0 5 0 0 0

- Output:
  - BFS: {0: None, 1: 0, 2: 0, 4: 1, 9: 1, 3: 2, 5: 2, 6: 4}, [0, 1, 4, 6]
  - DFS: {0: None, 2: 0, 5: 2, 6: 5}, [0, 2, 5, 6]
  - UCS: {0: None, 1: 0, 2: 0, 9: 1, 4: 1, 5: 2, 6: 9}, [0, 1, 9, 6]
  - GBFS: {0: None, 1: 0, 2: 1, 9: 1, 5: 2, 6: 5}, [0, 1, 2, 5, 6]
  - A*: {0: None, 1: 0, 9: 1, 2: 0, 4: 1, 5: 2, 6: 9}, [0, 1, 9, 6]

e. Test case 5

- Input:

  4 3
  0 6 7 0 8 0 2 0
  6 0 0 4 3 4 9 1
  0 6 0 0 4 4 0 0
  9 9 0 0 7 0 0 1
  0 0 5 0 0 7 0 0
  0 8 5 0 0 0 0 0
  0 0 0 0 5 5 0 0
  7 0 0 0 6 9 8 0

- Output:
  - BFS: {4: None, 2: 4, 5: 4, 1: 2, 0: 1, 3: 1}, [4, 2, 1, 3]
  - DFS: {4: None, 5: 4, 2: 5, 1: 2, 7: 1, 6: 7, 0: 7, 3: 1}, [4, 5, 2, 1, 3]
  - UCS: {4: None, 2: 4, 5: 4, 1: 2, 7: 1, 3: 1}, [4, 2, 1, 3]
  - GBFS: {4: None, 2: 4, 5: 2, 1: 2, 7: 1, 3: 1}, [4, 2, 1, 3]
  - A*: {4: None, 2: 4, 5: 4, 1: 2, 7: 1, 3: 1}, [4, 2, 1, 3]

6. REFERENCES

- Slide bài giảng môn học Cơ sở trí tuệ nhân tạo, Thầy Nguyễn Ngọc Đức https://cloud.ducnn.com/index.php/s/cqCO2iiQt1Sm7Xv#pdfviewer
- Breadth First Search or BFS for a Graph - GeeksforGeeks https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
- Depth First Search or DFS for a Graph - GeeksforGeeks https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
- Uniform-Cost Search (Dijkstra for large Graphs) - GeeksforGeeks https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/

- Greedy Best first search algorithm - GeeksforGeeks
  https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/
- A* Search Algorithm - GeeksforGeeks
  https://www.geeksforgeeks.org/a-search-algorithm/