Longest Path - Lời Giải

Nhóm 3

November 2025

Mô tả bài toán

Bạn được cho một đồ thị vô hướng đơn có N đỉnh ($1 \leq N \leq 50$) và M cạnh ($1 \leq M \leq 100$). Cần tìm **đường đi đơn dài nhất** (không lặp đỉnh) bắt đầu ở đỉnh 1 và kết thúc ở đỉnh N. Độ dài tính bằng số cạnh.

- **Input**: Dòng đầu: N M. M dòng sau: cạnh u v.
- **Output**: Dòng 1: độ dài lớn nhất. Dòng 2: các đỉnh của đường đi đạt độ dài đó.

1 Mô tả thuật toán

Vì sao không duyệt toàn bộ đường đi ? Số đường đi đơn từ 1 tới N có thể nổ lên rất lớn khi đồ thị gần đầy, dẫn đến độ phức tạp lớn, vì thế các thuật toán ngây thơ như DFS hay BFS đơn thuần ko thể giải quyết được bài toán này. Ta sẽ sử dụng *Beam search*: giống BFS theo từng lớp nhưng **chỉ giữ lại tối đa K ứng viên tốt nhất ở mỗi lớp**.

Mỗi trạng thái là một đường đi, ta lưu trữ thông tin những đỉnh đã thăm bằng mask `visitedMask` (bitmask 64 bit đủ vì $N \leq 50$) và đỉnh cuối cùng của đường đi hiện tại u. Khi xét các đường đi tiềm năng tiếp theo, ta thử tất cả đỉnh kề v của u chưa xuất hiện trong mask, ta sẽ lưu các đường đi tiềm năng này để sau này chọn lọc lại. Sau khi sinh xong lớp bfs tiếp theo, ta:

- **Lọc trùng heuristic**: Ta xem hai trạng thái là tương đương về mặt heuristic nếu chúng có visitedMask giống nhau và cùng kết thúc tại một đỉnh. Chỉ giữ một đại diện (đại diện dài hơn ưu tiên).
- **Sắp xếp ưu tiên candidates**: Ta sắp xếp các đường đi tiềm năng theo: (1) độ dài đường đi **giảm dần**, (2) visitedMask, (3) đỉnh cuối của đường đi.

- **Giới hạn lượng candidates**: Ta chỉ giữ tối đa K đường đi, đảm bảo rằng số lượng trạng thái không bùng nổ

Ta sẽ tiếp tục Beam Search qua từng lớp cho đến khi ta không thể tiếp tục mở rộng đường đi được nữa, lúc này ta sẽ lấy đường đi có độ dài lớn nhất làm đáp án.

2 Phân tích tính chất thuật toán

- Bởi vì những đường đi có thể phức tạp nên ta phải lưu mask những đỉnh đã đi qua, nhằm đảm bảo yêu cầu đề bài
- nếu N nhỏ thì ta có thể sử dụng tằng dp bitmask luôn, nhưng vì $N \leq 50$ nên ta sẽ sử dụng thuật heuristic để tìm đường đi tốt nhất
- Lọc trùng giúp tăng độ đa dạng đường đi: 2 đường đi khác nhau nhưng cùng tập đỉnh đã đi và cùng kết thúc tại x thì các đường đi tiềm năng từ hai đường đi này sẽ tương đối giống nhau, ở đây ta hi sinh độ chính xác tuyệt đối để đổi lấy tiềm năng heuristic, tại vì có thể tồn tại những đường đi đa dạng hơn mà ta không hề xét nếu ta không lọc trùng
- Ta Ưu tiên đường đi dài hơn trước để những candidate tốt sẽ tiếp tục được duy trì lâu dài.

3 Độ phức thời gian và không gian

Giả sử mỗi beam giữ tối đa K trạng thái. Tổng số lượng đường đi tiềm năng là $K * \Delta$ (bậc trung bình), với $\Delta \leq N$. Một lớp beam search sẽ tốn: $O(K * N)$ sinh đường đi + sắp xếp $O((K * N)\log(K * N))$. Vì đường đi có độ dài tối đa là vN đỉnh nên thuật Beam sẽ chạy cùng lắm N lần, vậy độ phức tạp thời gian của thuật toán là $O(N * (K * N) * \log(K * N))$

Vì sau mỗi lần beam search ta xóa hết các đường đi tiềm năng và chỉ giữ lại K đường, độ phức tạp bộ nhớ của thuật toán là: $O(K * N)$

4 Cài đặt (Python)

trong cài đặt thuật toán, ta chọn $K = 10000$

```python
import sys
from collections import deque

def main():
```

```
    data = sys.stdin.read().strip().split()
    it = iter(data)
    N = int(next(it)); M = int(next(it))
    g = [[] for _ in range(N+1)]
    for _ in range(M):
        u = int(next(it)); v = int(next(it))
        g[u].append(v); g[v].append(u)

    if N == 1:
        print(0); print(1); return

    reach = [False]*(N+1); q = deque([1]); reach[1] = True
    while q:
        u = q.popleft()
        for v in g[u]:
            if not reach[v]: reach[v]=True; q.append(v)
    if not reach[N]:
        print(-1); return

    class State:
        __slots__ = ('path','mask')
        def __init__(self, p, m): self.path=p; self.mask=m

    K = 10000  # beam width
    beam = [State([1], 1 << 0)]
    bestLen = -1; bestPath = []

    while beam:
        nxt = []
        for st in beam:
            u = st.path[-1]
            if u == N:
                if len(st.path) > bestLen:
                    bestLen = len(st.path); bestPath = st.path[:]
                continue
            for v in g[u]:
                bit = 1 << (v-1)
                if st.mask & bit: continue
                nxt.append(State(st.path+[v], st.mask | bit))

        nxt.sort(key=lambda s: (-len(s.path), s.mask, s.path[-1]))

        dedup = []
        lastKey = None
        for s in nxt:
```

```
            key = (s.mask, s.path[-1])
            if key == lastKey: continue
            dedup.append(s); lastKey = key

        if len(dedup) > K: dedup = dedup[:K]
        beam = dedup

    if bestLen == -1:
        print(-1)
    else:
        print(bestLen - 1)
        print(*bestPath)

if __name__ == '__main__':
    main()
```