Almost Shortest Path

Big-O Blue - Lecture 08: Dijkstra

Tóm tắt đề bài

Tóm tắt đề bài

Cho đồ thị một chiều **N** đỉnh, **M** cạnh. Tìm độ dài của đường đi gần ngắn nhất từ đỉnh **S** tới đỉnh **D**.

Tuy nhiên, tất cả các cạnh trên đường đi này đều không được thuộc bất kì đường đi ngắn nhất nào từ đỉnh **S** đến **D**.

Mô tả Input/Output

Input:

- Có nhiều bộ test, mỗi bộ test gồm:
- Dòng đầu gồm hai số nguyên **N** ($2 \le N \le 500$) và **M** ($1 \le M \le 10^4$). Các đỉnh đánh số từ 0 đến **N** – 1.
- Dòng tiếp theo gồm hai số nguyên **S** và **D**.
- **M** dòng kế tiếp gồm ba số **U, V, P** ($1 \le P \le 10^3$) cho biết có cạnh **U→V** với trọng số **P**. Có tối đa một cạnh **U→V**.
- Kết thúc bộ test là “0 0”.

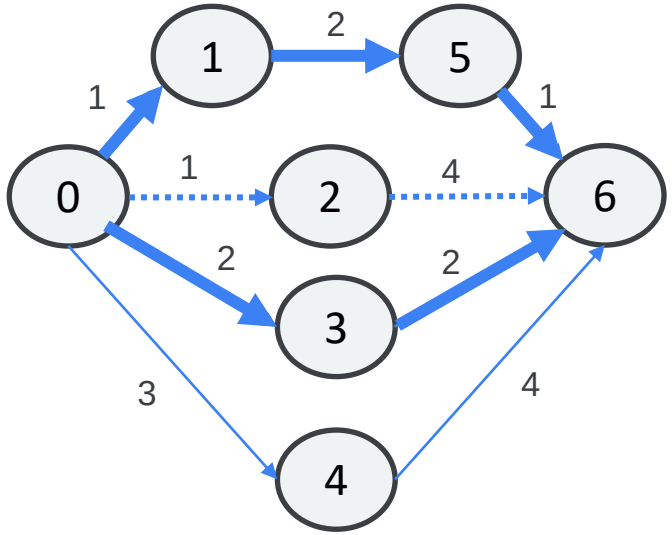
Output:

- Với mỗi bộ test, in ra độ dài đường đi gần ngắn nhất từ đỉnh xuất phát đến đỉnh cần đến. Nếu không tồn tại đường đi như vậy thì in -1.

Giải thích ví dụ

Ví dụ 1

Input:	Output:
7 9	5
0 6	
0 1 1	
0 2 1	
0 3 2	
0 4 3	
1 5 2	
2 6 4	
3 6 2	
4 6 4	
5 6 1	



Tìm đường đi gần ngắn nhất từ 0 → 6.

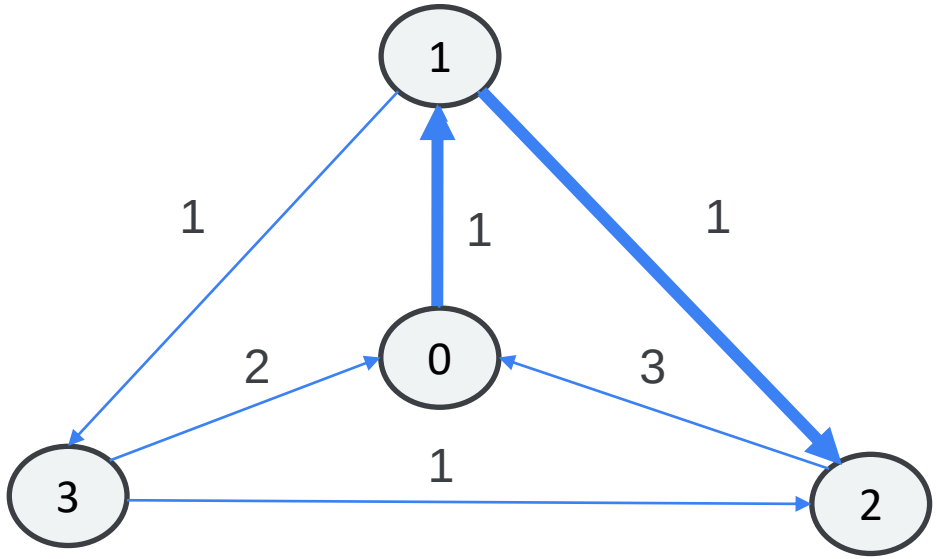
- Có 2 đường đi ngắn nhất (in đậm): 0 → 1 → 5 → 6 (độ dài 4) và 0 → 3 → 6 (độ dài 4).
- Đường đi gần ngắn nhất (đứt khúc) không chứa các đoạn đã được in đậm: 0 → 2 → 6 (độ dài 5).

Kết quả là 5.

Ví dụ 2

Input:
```
4 6
0 2
0 1 1
1 2 1
1 3 1
3 2 1
2 0 3
3 0 2
```

Output:
```
-1
```



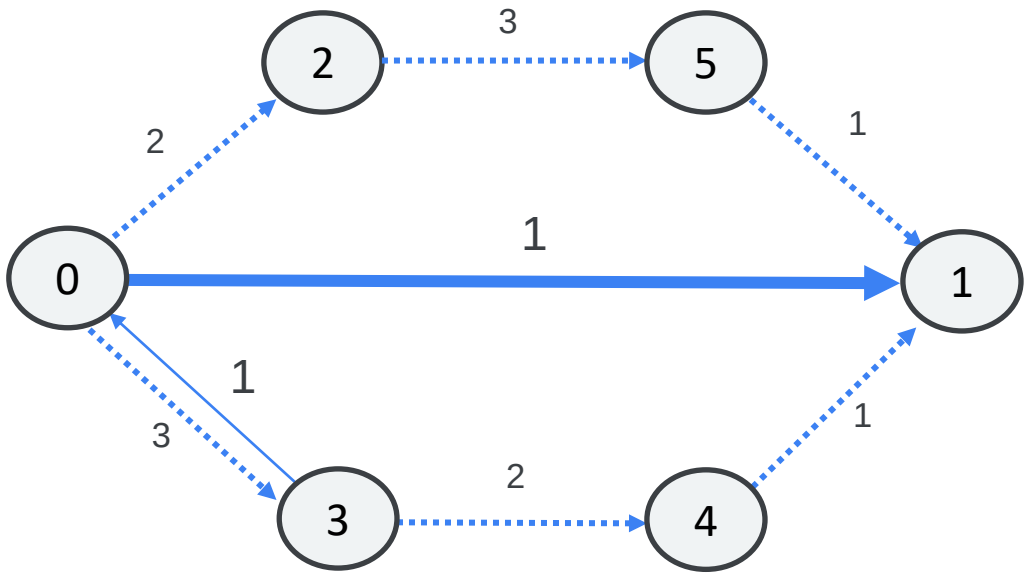
Tìm đường đi gần ngắn nhất từ 0 → 2.

- Có 1 đường đi ngắn nhất: 0 → 1 → 2 (độ dài 2).
- Không tồn tại đường đi nào từ 0 tới 2 mà không chứa cạnh (0, 1) và cạnh (1, 2)

Kết quả là -1.

Ví dụ 3

Input:	Output:
6 8	6
0 1	
0 1 1	
0 2 2	
0 3 3	
2 5 3	
3 4 2	
4 1 1	
5 1 1	
3 0 1	



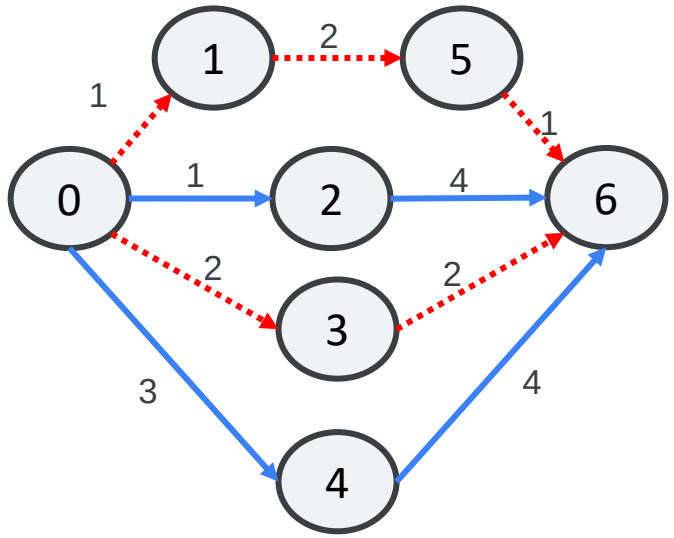
Tìm đường đi gần ngắn nhất từ 0 → 1.

- Đường đi ngắn nhất (in đâm): 0 → 1 (độ dài 1).
- Có 2 đường đi gần ngắn nhất (đứt khúc):
  0 → 2 → 5 → 1 và 0 → 3 → 4 → 1 (độ dài 6).

Kết quả là 6.

Hướng dẫn giải

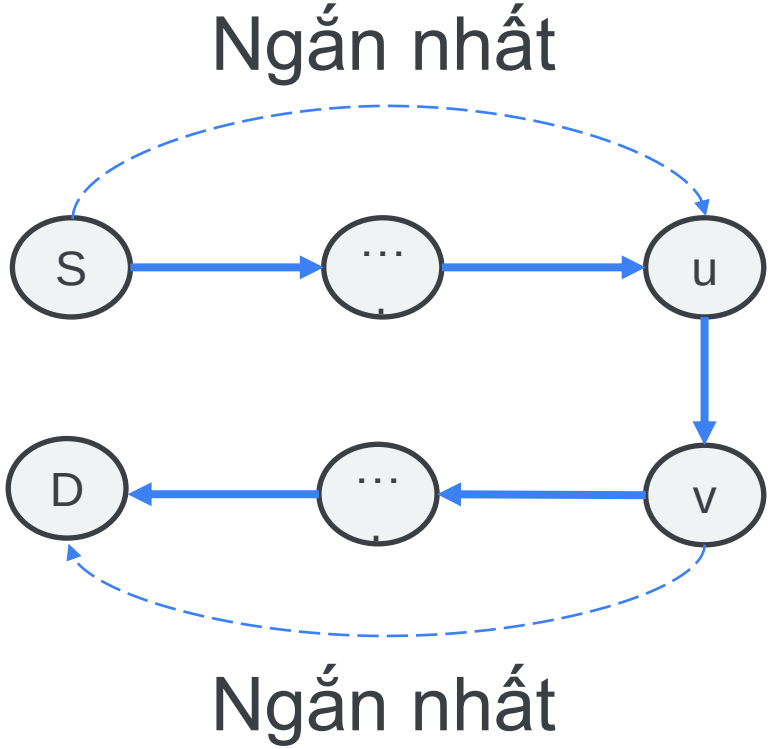
Nhận xét



- Vì đường đi gần ngắn nhất không chứa các cạnh của đường đi ngắn nhất
- Nên xóa tất cả các cạnh thuộc các đường đi ngắn nhất (cạnh đỏ).
- Sau đó tiến hành Dijkstra để tìm đường đi gần nhắn nhất

Làm thế nào để kiểm tra 1 cạnh có thuộc đường đi ngắn nhất nào không ?

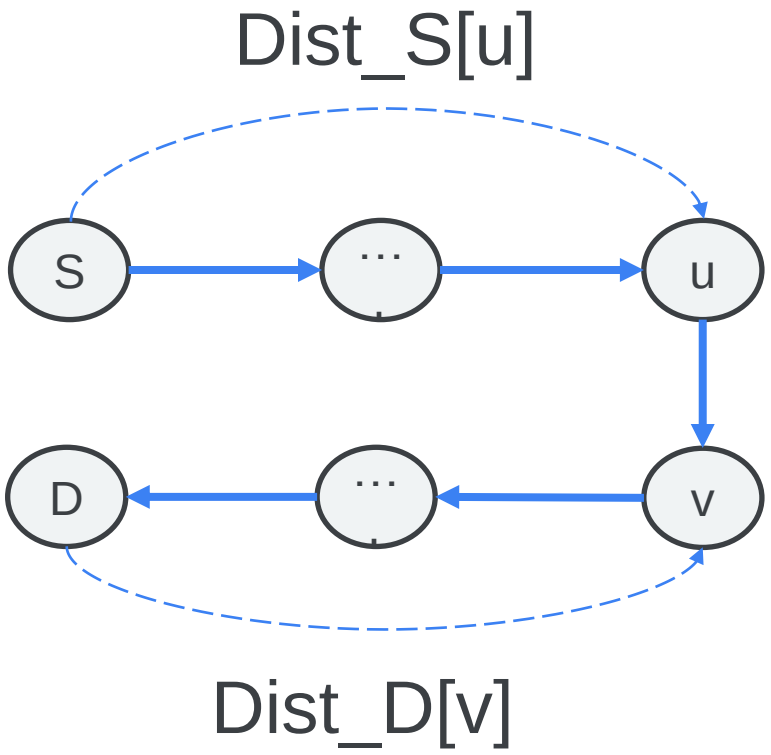
Nhận xét



Giả sử 1 cạnh **(u, v)** nào đó thuộc đường đi ngắn nhất từ **S** đến **D**.

- **S → … → u → v → … → D** là đường đi ngắn nhất.
- **S → … → u** là đường đi ngắn nhất từ **S** tới **u**.
- **v → …. → D** là đường đi ngắn nhất từ **v** tới **D**.

Nhận xét



- Gọi **dist_s[u]** là độ dài đường đi ngắn nhất từ **S** tới **u**.
- Gọi **dist_d[v]** là độ dài đường đi ngắn nhất từ **v** tới **D**.
- Đặt **|(u, v)|** là độ dài của cạnh 1 chiều u → v.

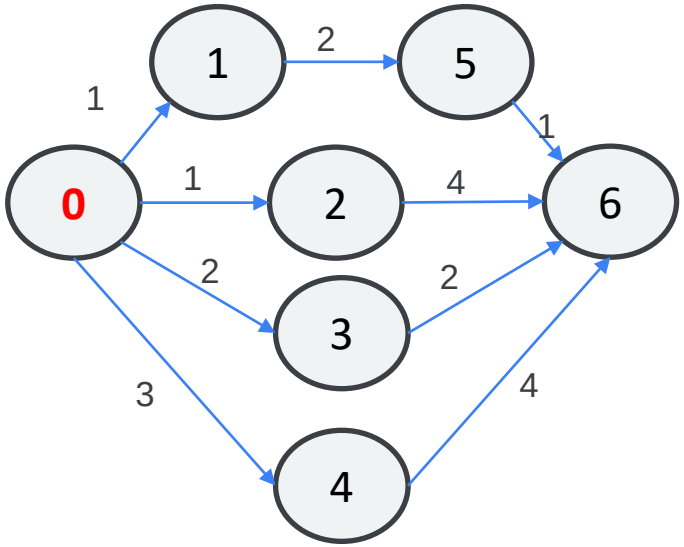
Vậy **(u,v)** thuộc đường đi ngắn nhất từ **S** tới **D** nếu:

dist_s[u] + |(u, v)| + dist_d[v] == dist_s[D]

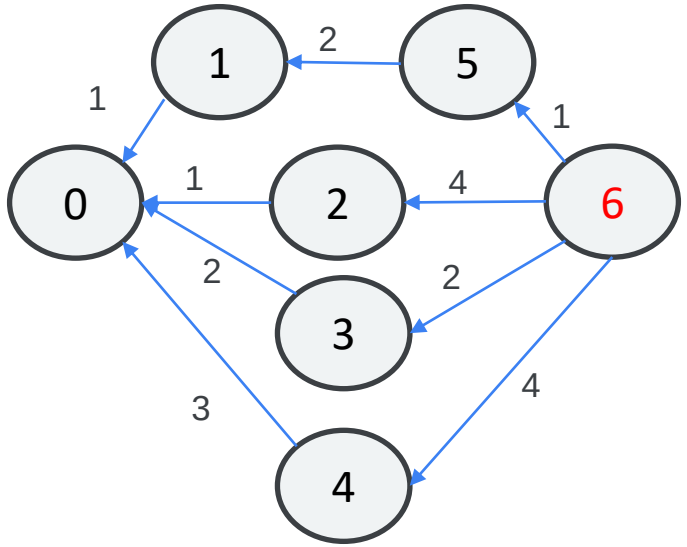
Nhận xét

Để tính **dist_s[u]** ta sẽ Dijsktra từ đỉnh **S**

Nhưng để tính **dist_d[v]** ta cần đổi chiều tất cả các cạnh trong đồ thị lại. Sau đó tiến hành Dijkstra từ đỉnh **D**.



Đồ thị ban đầu



Đồ thị sau khi đảo chiều

Các bước giải

B1: Tạo 3 mảng lưu đồ thị, đồ thị ban đầu, đồ thị đảo chiều, đồ thị mới sau khi xóa

B2: Tiến hành Dijkstra từ đỉnh **S** bằng đồ thị ban đầu, kết qua lưu vào mảng **dist_s**

B3: Tiến hành Dijkstra từ đỉnh **D** bằng đồ thị đảo chiều, kết quả lưu vào **dist_d**

B4: Xét các cạnh của đồ thị ban đầu, lưu cạnh **(u, v)** vào đồ thị mới nếu:

dist_s[u] + |(u,v)| + dist_d[v] != dist_s[D]

B5: Tiến hành Dijsktra trên đồ thị mới.

B6: Xuất đường đi ngắn nhất từ **S** tới **D** của đồ thị mới.

Độ phức tạp: **O(T * ElogV)** với **E** là số lượng cạnh của đồ thị, **V** là số lượng đỉnh của đồ thị và **T** là số lượng bộ test trong từng dữ liệu đầu vào.

Mã giả

Mã giả

```
while true:
    read(n, m)
    if n == 0 and m == 0:
        exit()
    read(S, D)
    graph_s , graph_d, graph_new = [[], [], ...]
    for i = 1 to m:
        read(u, v, p)
        graph_s[u].push((v, p))
        graph_d[v].push((u, p))
    dist_s, dist_d, dist_new = [Inf, Inf,...]
    Dijkstra(graph_s, dist_s, S)
    Dijkstra(graph_d, dist_d, D)
    for i = 0 to N - 1:
        for (v, w) in graph_s[i]:
            if dist_s[i]+w+dist_d[v] != dist_s[D]:
                graph_new[i].push((v, w))
```

```
Dijkstra(graph_new, dist_new, S)
    if dist_new[D] == inf:
        print(-1)
    else:
        print(dist_new[D])


function Dijkstra(graph, dist, S):
    priority_queue pq
    dist[S] = 0
    pq.push((S, 0))
    while pq not empty:
        u, du = pq.pop()
        if du >= dist[u]:
            continue
        for v, w in graph[u]:
            if w + du < dist[v]:
                dist[v] = w + du
                ps.push((v, dist[v]))
```