ĐỒ THỊ - THUẬT TOÁN DIJKSTRA (tt)

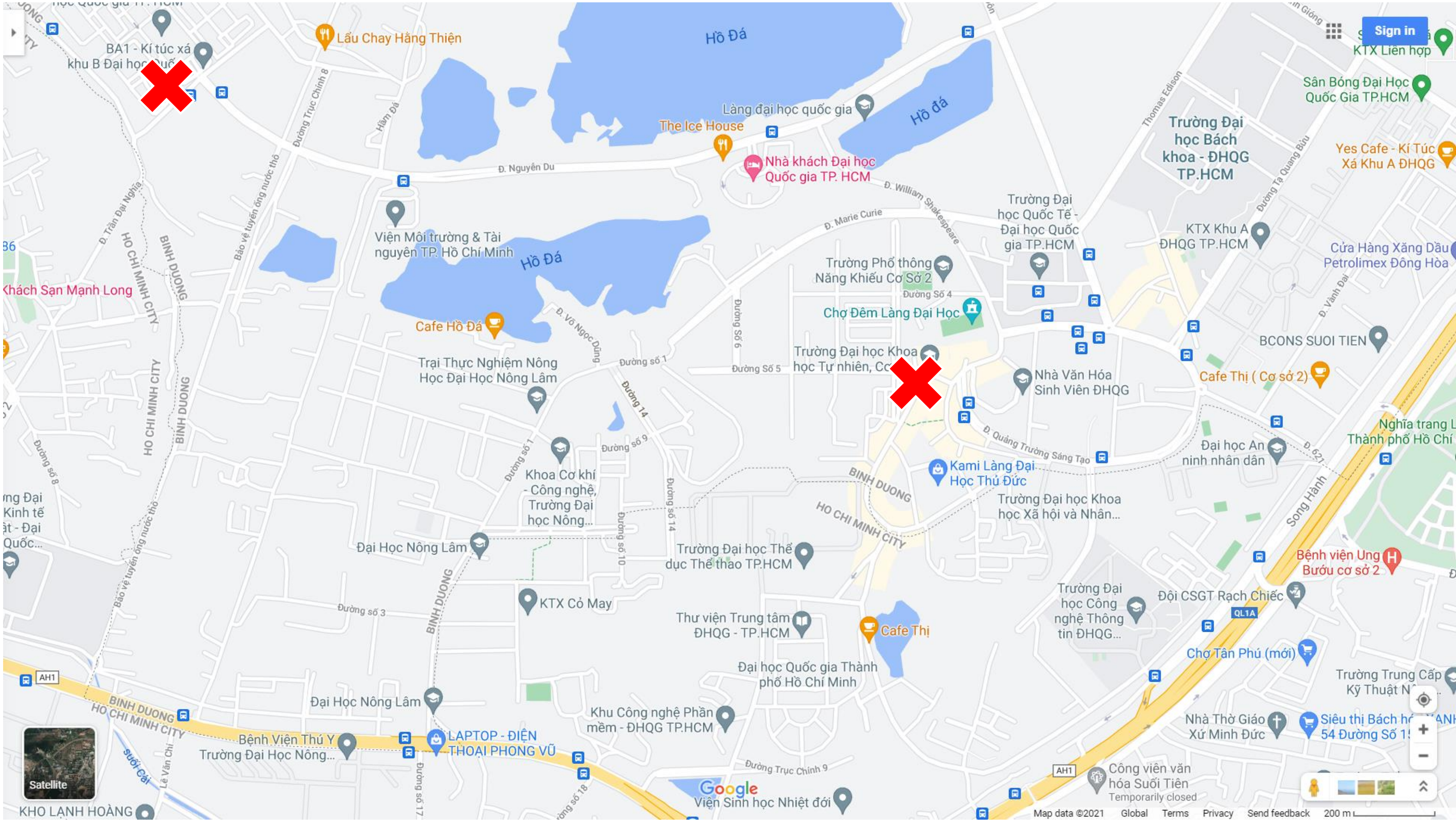
DATA STRUCTURES & ALGORITHMS

ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

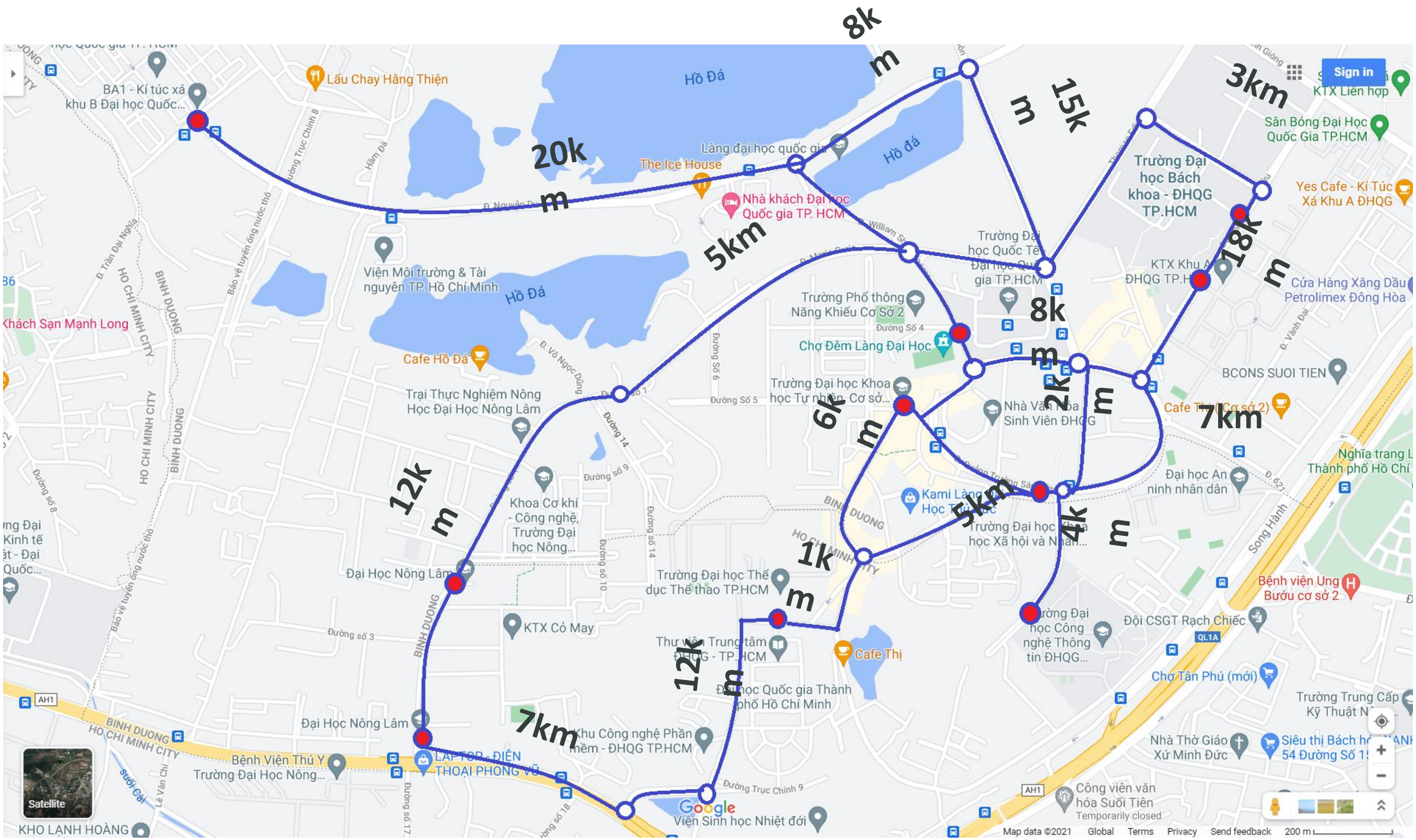
NỘI DUNG

- Các khái niệm trên đồ thị
  - Định nghĩa
  - Các loại đồ thị
  - Đường đi, chu trình, liên thông
- Biểu diễn đồ thị trên máy tính
- Các thuật toán duyệt đồ thị: BFS - DFS
- **Ứng dụng: Bài toán tìm đường đi ngắn nhất dùng thuật toán Dijkstra**

Ví dụ: Bài toán tìm đường đi ngắn nhất



Ví dụ: Bài toán tìm đường đi ngắn nhất



Các bài toán

1. Tìm đường đi ngắn nhất từ một đỉnh đến một đỉnh khác.
2. Tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại
3. Tìm đường đi ngắn nhất giữa các cặp đỉnh

Thuật toán Dijkstra

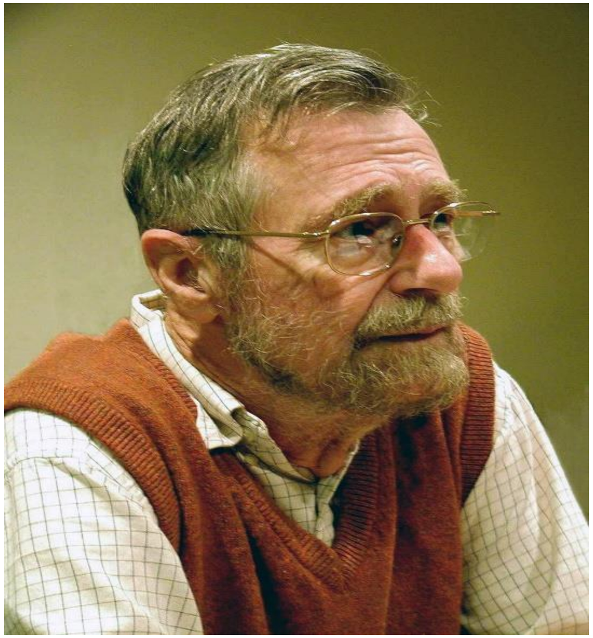
Giới thiệu:

- Thuật toán tìm đường đi ngắn nhất từ một đỉnh xuất phát đến tất cả các đỉnh còn lại trong đồ thị có hướng, có trọng số không âm
- Được công bố lần đầu vào năm 1959
- Được đặt tên theo tên của nhà toán học và nhà vật lý người Hà Lan Edsger W. Dijkstra

Giới thiệu

Edsger Wybe Dijkstra
(/ˈdaɪkstrə/ DYKE-strə)
(11/5/1930 – 6/8/2002)

- Nhà toán học, nhà vật lý, nhà khoa học máy tính người Hà Lan
- Làm việc tại Trung tâm toán học, Viện nghiên cứu quốc gia về toán học và khoa học máy tính tại Amsterdam
- Giữ chức vị giáo sư tại Đại học Kỹ thuật Eindhoven, Hà Lan

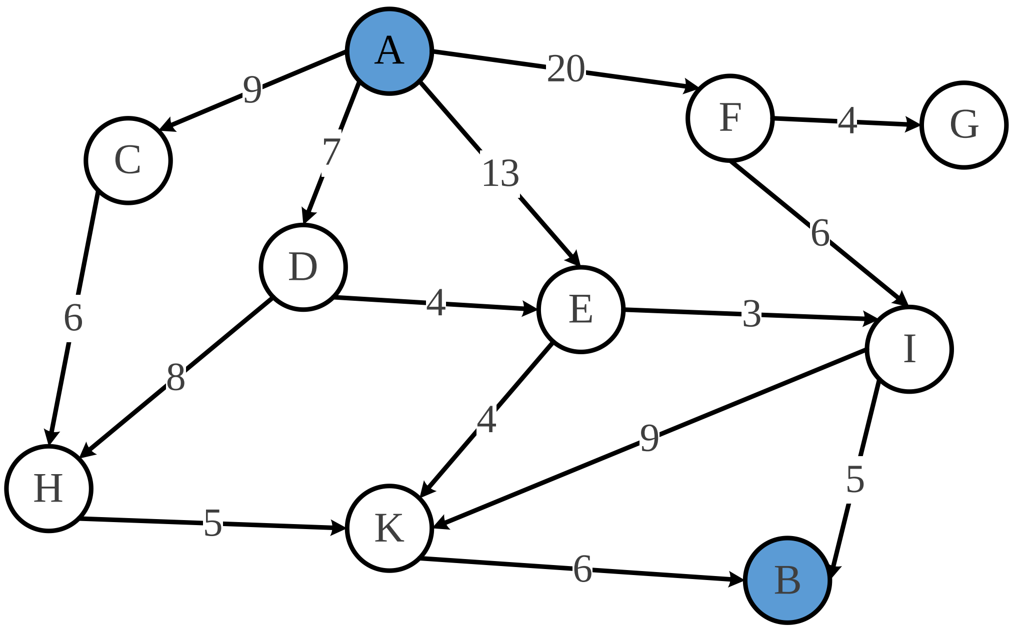


- Có các đóng góp mang tính chất nền tảng trong lĩnh vực ngôn ngữ lập trình
- Thuật toán Dijkstra, hệ điều hành THE và cấu trúc semaphore…

"Computer Science is no more about computers than astronomy is about telescopes"

Thuật toán Dijkstra

- **Input:** Đồ thị có hướng $G=(V, E, w)$, $w(i, j) \geq 0\ \forall\ (i, j) \in E$, đỉnh nguồn s, đỉnh đích g
- **Output:** đường đi ngắn nhất từ đỉnh s đến đỉnh g và độ dài của đường đi
- Ví dụ: Tìm độ dài đường đi ngắn nhất từ A đến B



Thuật toán Dijkstra

Ý tưởng chính:

- Hàm d(u) dùng để lưu trữ độ dài đường đi (khoảng cách) ngắn nhất từ đỉnh nguồn s đến đỉnh u

$$d(u) = \min\{d(v) + w(v,u), v \in X(u)\}$$

$X(u)$ là tập tất cả các đỉnh có cạnh đi tới đỉnh u

- Đặt khoảng cách từ đỉnh nguồn s đến chính nó là 0 và đến tất cả các đỉnh khác là vô cùng
- Tiến hành lặp cho đến khi tất cả các đỉnh đã được xác định khoảng cách ngắn nhất từ s hoặc không còn đỉnh nào có thể đạt tới từ s
- Mỗi lần lặp, chọn đỉnh p chưa đi qua có giá trị d(p) nhỏ nhất, cập nhật khoảng cách của các đỉnh kề thông qua đỉnh được chọn p

Ký hiệu

Gọi:

Open : tập các đỉnh có thể xét ở bước kế tiếp, các đỉnh có thể được xem xét lại, đỉnh chờ duyệt

Close : tập các đỉnh đã xét/đã duyệt, không xem xét lại

s : đỉnh xuất phát

g : đỉnh đích

p : đỉnh đang xét, đỉnh hiện hành

Γ(p) : tập các đỉnh kề của p (call gamma)

q $\in$ Γ(p) : một đỉnh k

Thuật toán Dijkstra

❖ Giải thuật tham khảo:

Mỗi đỉnh p tương ứng với 1 số d(p): khoảng cách đi từ đỉnh ban đầu tới p

Bước 1: Khởi tạo

Open := {s};

Close := {};

d(s) := 0;

Bước 2: While (Open ≠ {})

2.1 Chọn p thuộc Open có d(p) nhỏ nhất

2.2 Nếu p là trạng thái kết thúc thì xuất đường đi, thoát

2.3 Nếu p đã duyệt rồi thì bỏ qua, trở lại đầu vòng lặp

2.4 Chuyển p qua Close, và **mở các đỉnh kế tiếp q sau p (kề với p)**

d(q) = d(p) + cost(p,q) ;

Thêm q vào Open

Bước 3: Không có kết quả.

Thuật toán Dijkstra

<u>Bước 2</u>: While (**Open** $\neq$**{}**)

2.4 Chuyển p qua Close, và mở các đỉnh kế tiếp q sau p

2.3.1 Nếu q đã có trong Open

nếu d(q)> d(p)+cost(p,q)

d(q) = d(p) + cost(p,q); parent(q)=p ;

2.3.2 Nếu q chưa có trong Open và Close

d(q) = d(p) +cost(p,q);

parent(q)=p;

Thêm q vào Open

Câu hỏi thảo luận:

- Cách truy xuất đường đi?
- Nếu p hoặc q đã có trong Close thì có cần xử lý không?
- Độ phức tạp?

Using a priority queue

```
function Dijkstra(Graph, source):
    dist[source] ← 0                          // Initialization

    create vertex priority queue Q

    for each vertex v in Graph.Vertices:
        if v ≠ source
            dist[v] ← INFINITY                // Unknown distance from source to v
            prev[v] ← UNDEFINED               // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                     // The main loop
        u ← Q.extract_min()                   // Remove and return best vertex
        for each neighbor v of u:             // Go through all v neighbors of u
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

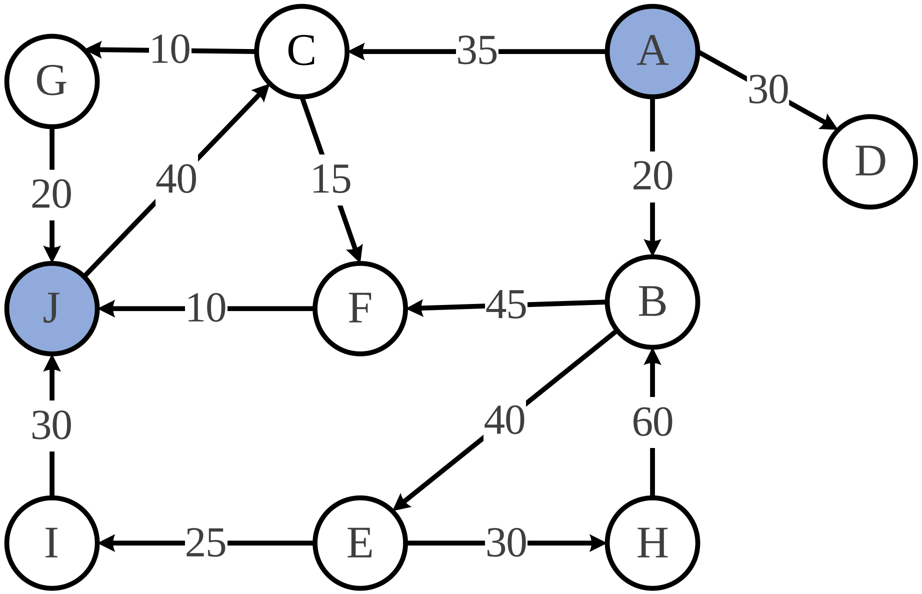
Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only *source*; then, inside the **if** `alt < dist`

- A min-priority queue is an abstract data type that provides 3 basic operations: `add_with_priority()`, `decrease_priority()` and `extract_min()`. As mentioned earlier, using such a data structure can lead to faster computing times than using a basic queue

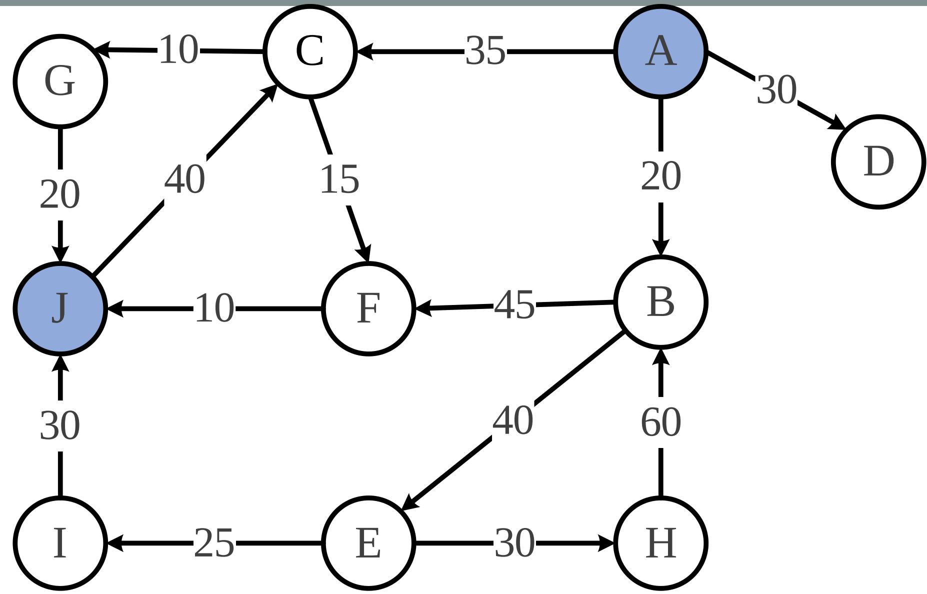
Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only source; then, inside the if alt < dist[v] block, the decrease_priority() becomes an add_with_priority() operation if the node is not already in the queue

Ví dụ: Tìm đường đi ngắn nhất từ A đến J

```
10 14
A B C D E F G H I J
A B 20
A C 35
A D 30
B E 40
B F 45
C F 15
C G 10
E H 30
E I 25
F J 10
G J 20
H B 60
I J 30
J C 40
```



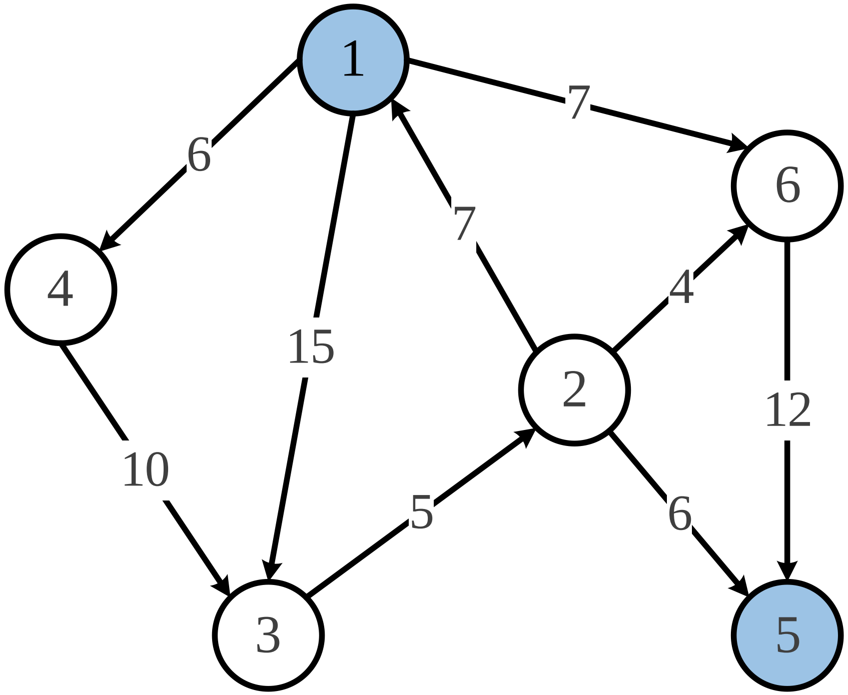
Ví dụ: Tìm đường đi ngắn nhất từ A đến J



p	Γ(p)	Open	Close
		{(A,0)}	{}
A	{B,C,D}	{(B,20), (C,35), (D,30)}	{A}
B	{E,F}	{(C,35), (D,30), (E,60), (F,65)}	{A,B}
D	{}	{(C,35), (E,60), (F,65)}	{A,B,D}
C	{F,G}	{(E,60), ~~(F,65)~~, (F,50), (G,45)}	{A,B,D,C}
G	{J}	{(E,60), (F,50), (J,65)}	{A,B,D,C,G}
F	{J}	{(E,60), ~~(J,65)~~, (J,60)}	{A,B,D,C,G,F}
J			

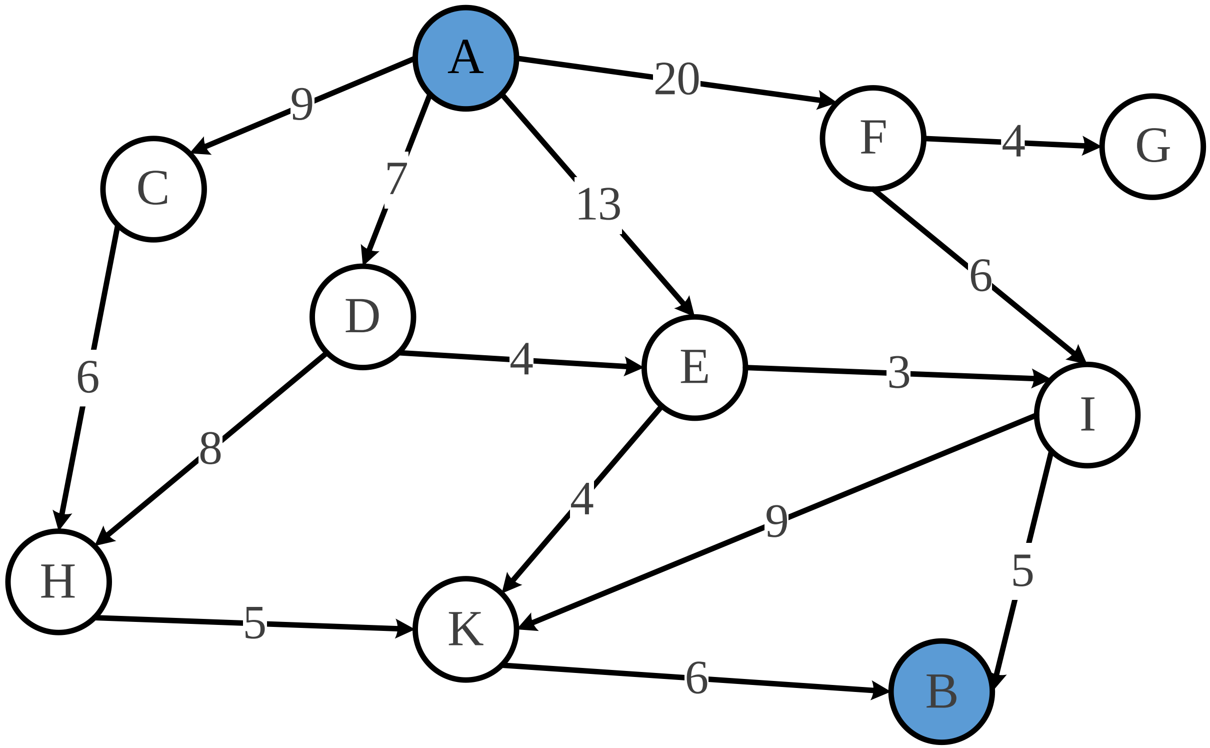
Bài tập

Tìm đường đi ngắn nhất từ 1 đến 5



Bài tập

- **Tìm đường đi ngắn nhất từ A đến B**



Tìm kiếm Dijkstra

❖Một số gợi ý về cài đặt:

- 1. Tổ chức cấu trúc dữ liệu
- vector<vector<int> > matrix; // ma trận trọng số của đồ thị
- vector<string> v_list; // lưu danh sách các tên đỉnh (chuỗi)
- map<string, int> v_index; // ánh xạ từ tên đỉnh sang index để có thể truy xuất thông tin trong ma trận kề
- priority_queue<….>open; // lưu các đỉnh chờ duyệt
- vector<bool> close(v, 0); // lưu thông tin đỉnh nào đã duyệt qua rồi
- map<string, string> parent; // lưu thông tin cha con, parent[u]=v nghĩa là cha của u và v
- vector<int> d(v,INF); // lưu khoảng cách (ngắn nhất) từ s đến các đỉnh khác

Priority Queue

- Khởi tạo min heap từ priority queue

```
priority_queue< pair<int,int>, vector<pair<int,int>>,greater<pair<int,int>>> open;
```

- Các hàm trên Priority Queue:
- size(): Trả về kích thước của hàng đợi
- empty(): kiểm tra có rỗng hay không
- top(): Lấy phần tử có độ ưu tiên cao nhất
- pop(): Xóa phần tử có độ ưu tiên cao nhất
- push(): Thêm 1 phần tử vào

Priority queue cũng giống như queue nhưng được thiết kế đặc biệt để phần tử ở đầu luôn luôn là phần tử có độ ưu tiên lớn nhất so với các phần tử khác

Tìm kiếm Dijkstra

❖ Một số gợi ý về cài đặt:

- Bước 1: open.push({0,s}); d[s] = 0;
- Bước 2: while (!open.empty())
-     pair<int, int> top = open.top(); open.pop();
-     int p=top.second, d=top.first;
-     if (p == g) {found = true; break; }
-     if(close[p]==1) continue;
-     close[p] = 1;
-     for (int i = 0; i < v; i++)
-         if (matrix[p][i]>0 && close[i]== 0)
-             if (d[i]>d[p]+ matrix[p][i])
-             d[i]=d[p]+ matrix[p][i] ; open.push({d[i],i}); parent[i] = p;
- Bước 3: Xử lý xuất đường đi hoặc kết luận không tìm thấy

Chúc các em học tốt!