**DijkstraAlgorithm(***non-gegative weighted simple digraph, vertex first)*

*for all vertices v # first currDist(v) = ∞;*

*currDist(first) = 0;*

*toBeChecked = V (all vertices);*

*checked = empty;*

*while toBeChecked is not empty:*

*u = a* ***vertex*** *in toBeChecked with min.currDist(u);*

*remove u from toBeChecked and add to checked;*

***for all vertices v adjacent to u and in toBeChecked:***

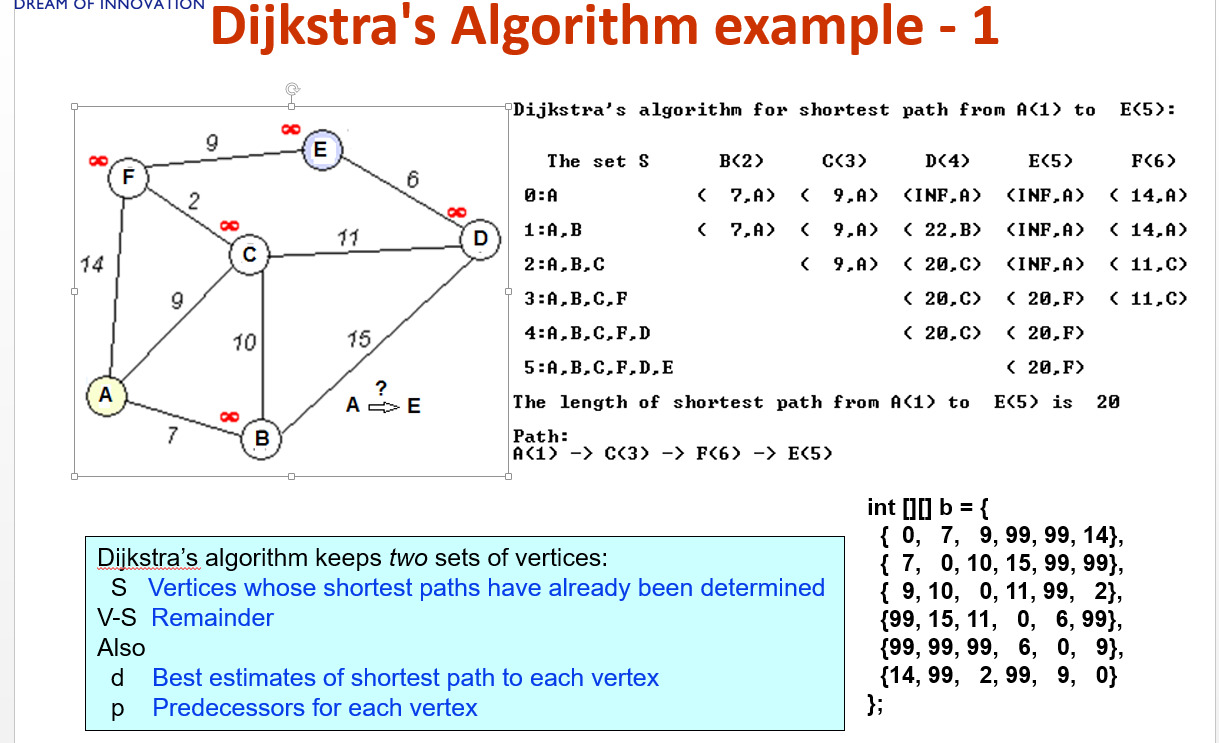
*if (currDist(v) > currDist(u) + weight(edge(uv)))*

*{*

*currDist(v) = currDist(u) + weight(edge(uv))*

*predeccessor(v) = u;*

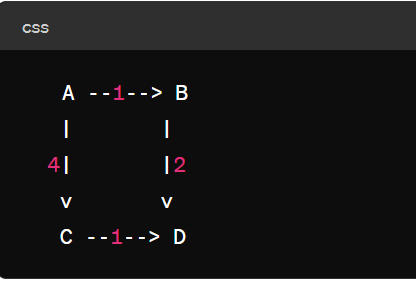
*}*



B0: Chọn u=A và cập nhật curDist(v) với v liền kề với u=A

B1: Chọn u=B và cập nhật curDist(v) với v liền kề với u=B

CHAT GPT:

  
Để minh họa thuật toán Dijkstra, hãy xem xét một đồ thị có hướng (directed graph) với trọng số không âm, và ta sẽ tìm đường đi ngắn nhất từ một đỉnh bắt đầu đến các đỉnh khác. Ví dụ sau đây cho thấy một đồ thị đơn giản và quá trình thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh khác.

**Đồ thị mẫu**

* Đồ thị có bốn đỉnh: A, B, C, D.
* Cạnh nối giữa các đỉnh có trọng số như sau:
  + A đến B: 1
  + A đến C: 4
  + B đến D: 2
  + C đến D: 1

**Dijkstra từ đỉnh A**

* Bắt đầu từ đỉnh A, thiết lập khoảng cách ban đầu từ A đến các đỉnh khác là vô cực (**∞**), trừ A có khoảng cách là 0. Tạo tập hợp **toBeChecked** gồm tất cả các đỉnh, và tập hợp **checked** rỗng.

1. **Chọn đỉnh có khoảng cách ngắn nhất**:
   * Trong **toBeChecked**, chọn đỉnh có khoảng cách nhỏ nhất. Đỉnh A có khoảng cách là 0, nên chọn A. Loại A khỏi **toBeChecked** và thêm vào **checked**.
   * **toBeChecked = {B, C, D}**
   * **checked = {A}**
2. **Cập nhật khoảng cách của các đỉnh liền kề**:
   * A có cạnh nối đến B và C.
   * Khoảng cách từ A đến B là 1, cập nhật **currDist(B)** = 1 và **predecessor(B)** = A.
   * Khoảng cách từ A đến C là 4, cập nhật **currDist(C)** = 4 và **predecessor(C)** = A.
3. **Chọn đỉnh tiếp theo**:
   * Trong **toBeChecked**, đỉnh B có khoảng cách nhỏ nhất là 1. Chọn B, loại khỏi **toBeChecked**, và thêm vào **checked**.
   * **toBeChecked = {C, D}**
   * **checked = {A, B}**
4. **Cập nhật khoảng cách của các đỉnh liền kề**:
   * B có cạnh nối đến D.
   * Khoảng cách từ B đến D là 2. Khoảng cách hiện tại từ A đến D là vô cực, nên cập nhật **currDist(D)** = 1 + 2 = 3 và **predecessor(D)** = B.
5. **Chọn đỉnh tiếp theo**:
   * Trong **toBeChecked**, đỉnh D có khoảng cách nhỏ nhất là 3. Chọn D, loại khỏi **toBeChecked**, và thêm vào **checked**.
   * **toBeChecked = {C}**
   * **checked = {A, B, D}**
6. **Cập nhật khoảng cách của các đỉnh liền kề**:
   * D không có cạnh nối đến các đỉnh trong **toBeChecked**, nên không có cập nhật.
   * **currDist(C)** = 4, không thay đổi.
7. **Chọn đỉnh cuối cùng**:
   * Trong **toBeChecked**, đỉnh C là đỉnh duy nhất còn lại. Chọn C, loại khỏi **toBeChecked**, và thêm vào **checked**.
   * **toBeChecked = {}**
   * **checked = {A, B, D, C}**

**Bản chất tư duy của thuật toán:**

- Đã chọn đỉnh nào vào tập S thì ta đã có độ dài shortest path từ đỉnh gốc đến đỉnh đó !

- Thuật toán Dijkstra thể hiện tính chất "greedy" (tham lam) ở mỗi bước chọn đỉnh có khoảng cách ngắn nhất từ tập hợp các đỉnh chưa được xử lý.