Đường đi trên đồ thị (Version 0.2)

Trần Vĩnh Đức

HUST

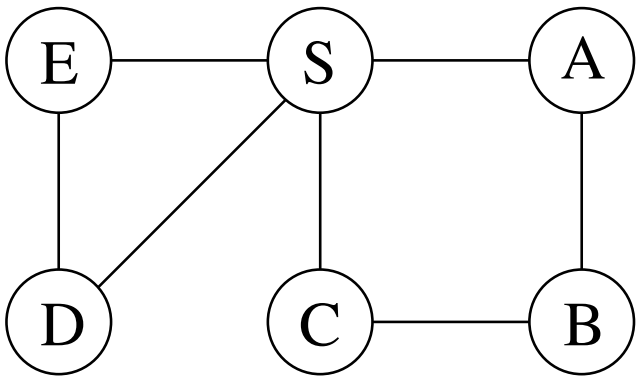
Ngày 24 tháng 7 năm 2018

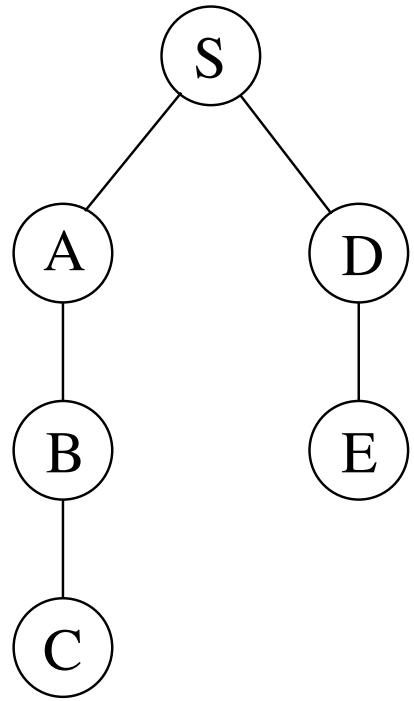
Tài liệu tham khảo

- S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*, July 18, 2016.
- Chú ý: Nhiều hình vẽ trong tài liệu được lấy tùy tiện mà chưa xin phép.

Nội dung

DFS và đường đi

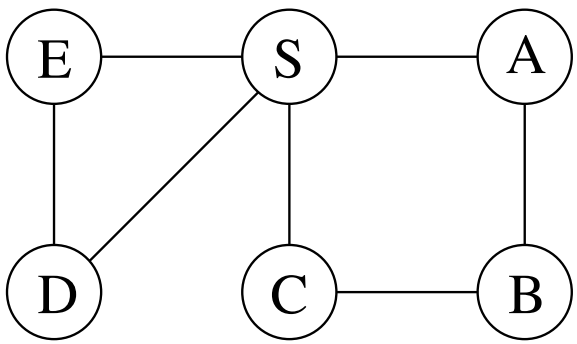




Đường đi trên cây DFS thường không phải là đường đi ngắn nhất.

Khoảng cách

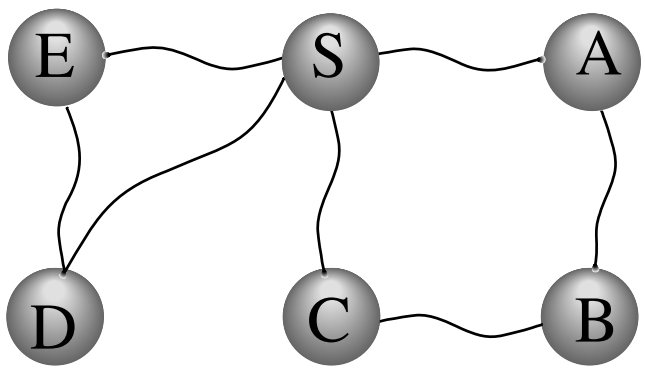
Khoảng cách giữa hai đỉnh là độ dài của đường đi ngắn nhất giữa chúng.

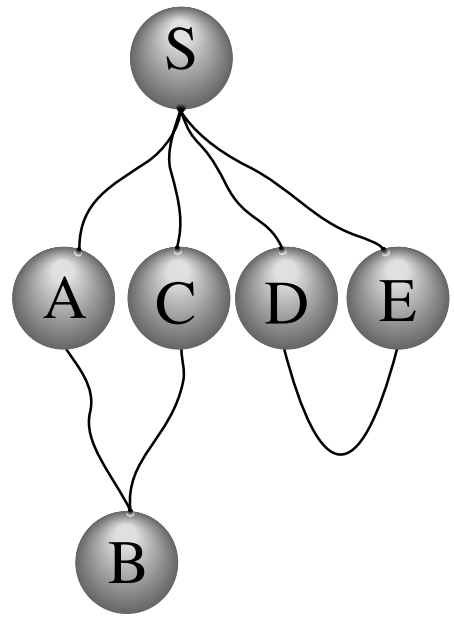


v	Khoảng cách $(S - v)$
A	
B	
C	
D	
E	

Mô hình vật lý của đồ thị

Giả sử rằng mọi cạnh có cùng độ dài. Ta nhấc đỉnh S lên:

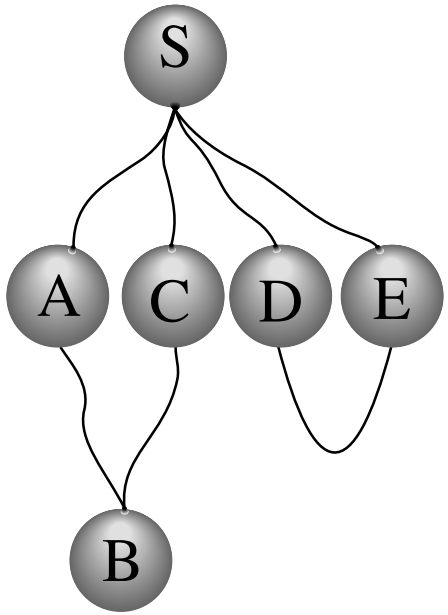




Tìm kiếm theo chiều rộng (Breadth-First Search)

Chia đồ thị thành các mức:

- S là mức có khoảng cách 0.
- Các đỉnh có khoảng cách tới S bằng 1.
- Các đỉnh có khoảng cách tới S bằng 2
- ...



Ý tưởng thuật toán: Khi mức d đã được xác định, mức $d+1$ có thể thăm bằng cách duyệt qua các hàng xóm của mức d.

Ý tưởng loang theo chiều rộng

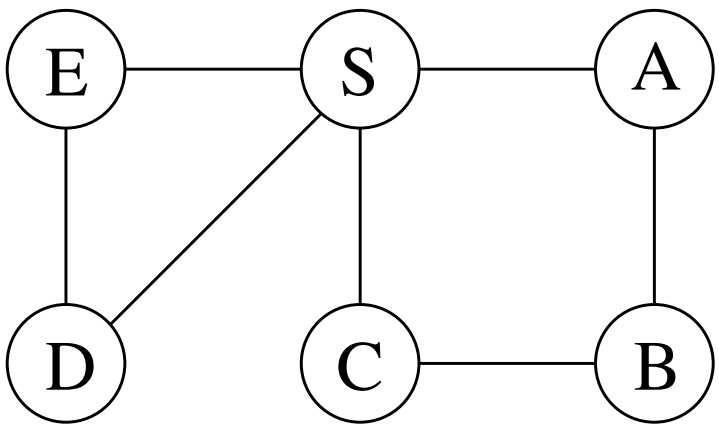
Khởi tạo: Hàng đợi Q chỉ chứa đỉnh s, là đỉnh duy nhất ở mức 0.

Với mỗi khoảng cách $d = 1, 2, 3, \ldots,$

- sẽ có thời điểm Q chỉ chứa các đỉnh có khoảng cách d và không có gì khác.
- Khi đó các đỉnh có khoảng cách d này sẽ được loại bỏ dần từ đầu hàng đợi,
- và các hàng xóm chưa được thăm sẽ được thêm vào cuối hàng đợi.

Bài tập

Chạy thuật toán BFS cho đồ thị dưới đây bắt đầu từ đỉnh S. Ghi ra hàng đợi Q sau mỗi lần thăm đỉnh.



Đỉnh thăm	Hàng đợi
S	$[S]$

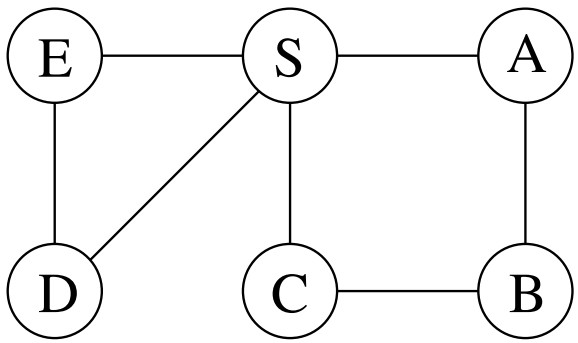
```
procedure bfs(G, s)
Input: đồ thị G = (V, E), có hướng hoặc vô hướng;
       một đỉnh s ∈ V
Output: Với mỗi đỉnh u đến được từ s,
       dist(u) = khoảng cách từ s tới u.

for all u ∈ V:
    dist(u) = ∞

dist(s) = 0
Q = [s]                                  (hàng đợi chỉ chứa s)
while Q khác rỗng:
    u = eject(Q)                         (loại bỏ u khỏi hàng đợi)
    for all edges (u, v) ∈ E:
        if dist(v) = ∞:
            inject(Q, v)                 (thêm v vào hàng đợi)
            dist(v) = dist(u) + 1
```

Bài tập

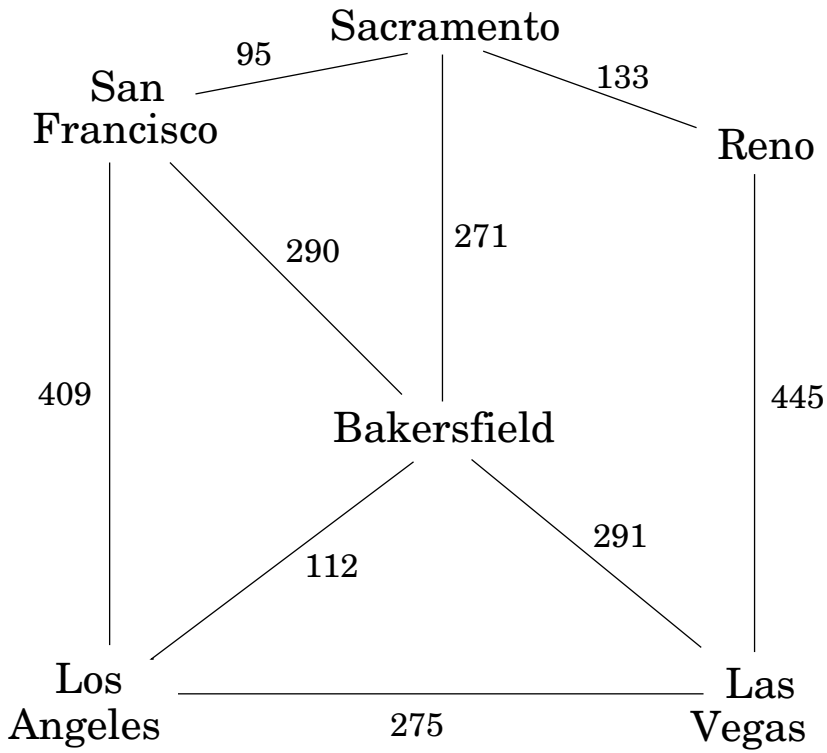
Hãy chạy thuật toán BFS cho đồ thị dưới đây và ghi ra nội dung của hàng đợi Q sau mỗi bước:



Thứ tự thăm đỉnh	Hàng đợi
S	$[S]$

Nội dung

Độ dài của cạnh

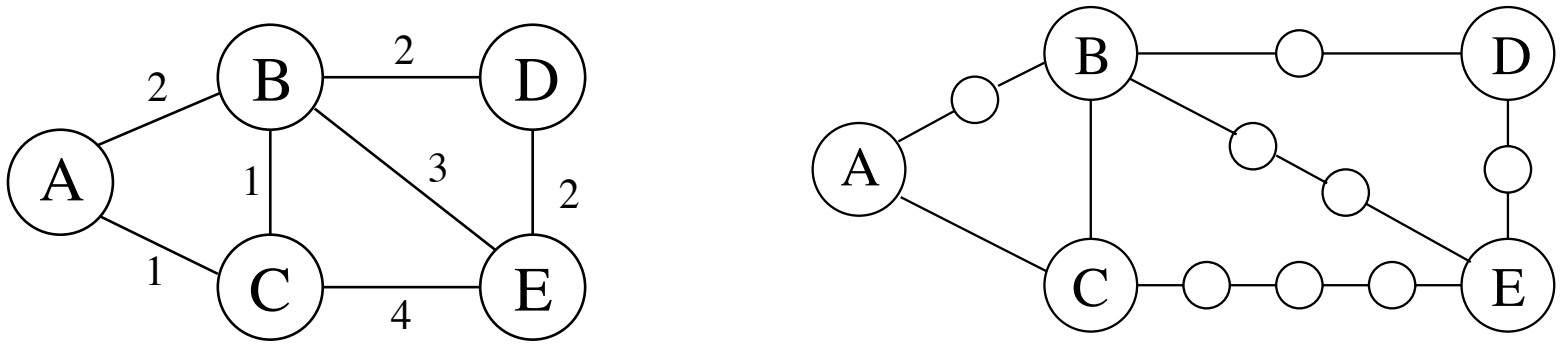


Trong các bài toán thực tế, mỗi cạnh e thường gắn với độ dài l_e.

Câu hỏi

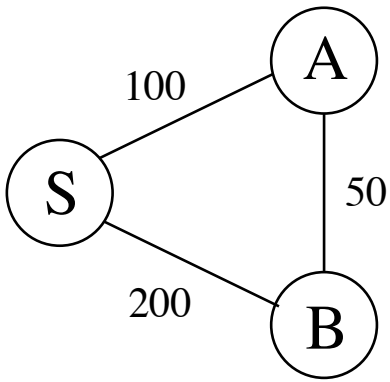
Liệu ta có thể sửa thuật toán BFS để nó chạy được trên đồ thị tổng quát $G = (V, E)$ trong đó mỗi cạnh có độ dài *nguyên dương* l_e?

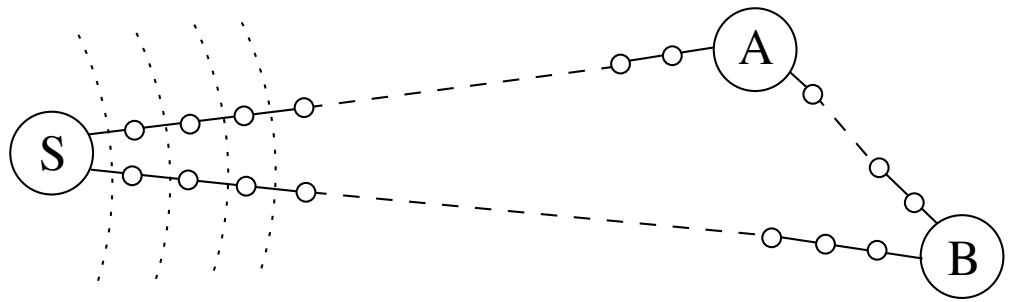
Tách cạnh thành các cạnh với độ dài đơn vị



Thay cạnh $e = (u, v)$ bởi l_e cạnh độ dài 1, bằng cách thêm $l_e - 1$ đỉnh tạm giữa u và v.

Vấn đề

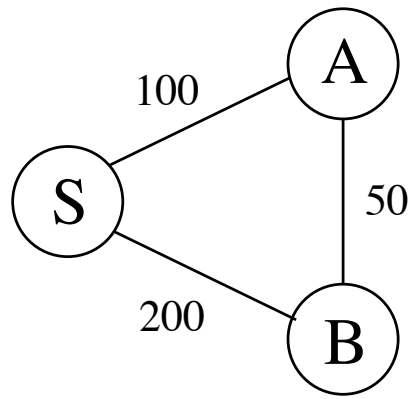




Cả 99 bước di chuyển đầu tiên đều xử lý $S-A$ và $S-B$ trên các đỉnh tạm.

Giải pháp: Đặt Alarm clock!

- Với đỉnh A, đặt hẹn $T = 100$
- Với đỉnh B, đặt hẹn $T = 200$
- Bị đánh thức khi A được thăm lúc $T = 100$
- Ước lượng lại thời gian đến của B là $T = 150$; và đặt lại Alarm cho B.



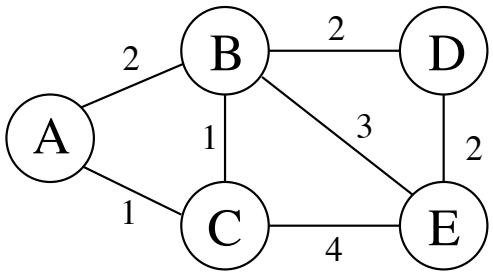
Thuật toán Alarm Clock

Đặt một alarm clock cho đỉnh s tại thời điểm $T = 0$

Lặp lại cho đến khi không còn alarm:
Giả sử alarm kêu tại thời điểm T cho đỉnh u. Vậy thì:

- Khoảng cách từ s tới u là T.
- Với mỗi hàng xóm v của u trong G:
  * Nếu vẫn chưa có alarm cho v,
    đặt alarm cho v tại thời điểm $T + l(u, v)$.
  * Nếu alarm của v đã đặt, nhưng lại muộn hơn so với $T + l(u, v)$,
    vậy thì đặt lại alarm cho v bằng $T + l(u, v)$.

Ví dụ



Thời gian	A	B	C	D	E
	0	–	–	–	–
0	(0)	2	1	–	–
1		2	(1)	–	5
2		(2)		4	5
4				(4)	5
5					(5)
	0	2	1	4	4

Hàng đợi ưu tiên

Tại sao cần hàng đợi ưu tiên? Để cài đặt hệ thống Alarm.

Hàng đợi ưu tiên là gì? Là một tập với mỗi phần tử được gắn với giá trị số (còn gọi là khóa) và có các phép toán sau:

Insert. Thêm một phần tử mới vào tập.

Decrease-key. Giảm giá trị khóa của một phần tử cụ thể.

Delete-min. Trả lại phần tử có khóa nhỏ nhất và xóa nó khỏi tập.

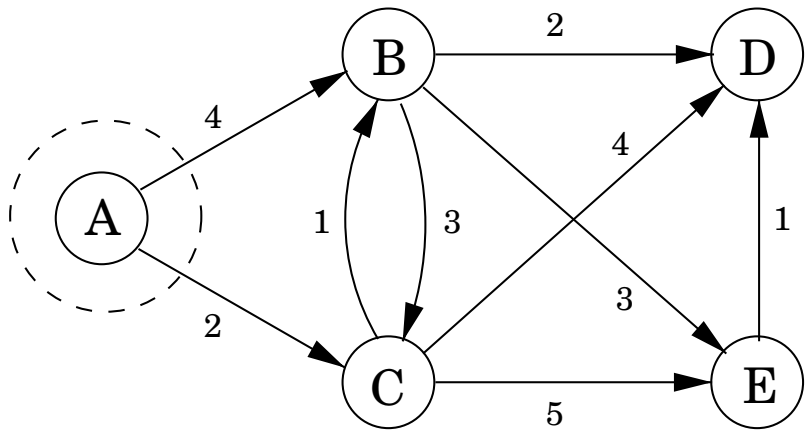
Make-queue. xây dựng hàng đợi ưu tiên cho tập phần tử và giá trị khóa cho trước.

Khóa của mỗi phần tử (đỉnh) ở đây chính là alarm của đỉnh đó. `Insert` và `Descrease-key` để đặt alarm; `Delete-min` để xác định thời điểm alarm tiếp theo kêu.

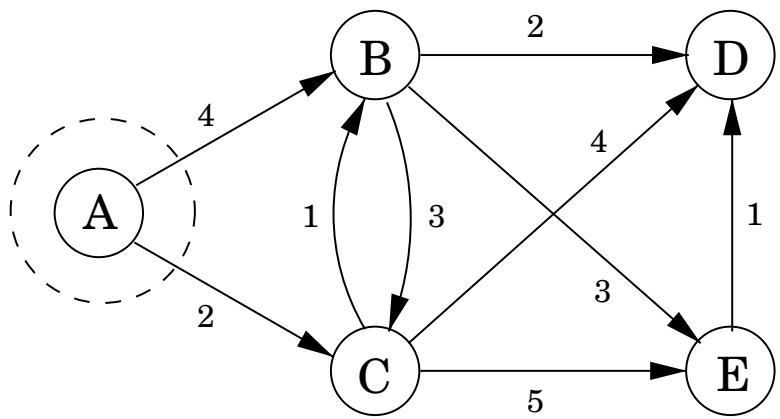
```
procedure dijkstra(G, l, s)
Input: đồ thị G = (V, E), có hướng hoặc vô hướng;
       độ dài các cạnh {l_e : e ∈ E}; đỉnh s ∈ V
Output: Với mỗi đỉnh u đến được từ s,
       dist(u) = khoảng cách từ s tới u.
for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil (đỉnh trước u trong đường đi ngắn nhất)
dist(s) = 0
H = makequeue(V)      (dùng các giá trị dist làm khóa)
while H khác rỗng:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if dist(v) > dist(u) + l(u, v):
            dist(v) = dist(u) + l(u, v)
            prev(v) = u                (đỉnh trước v là u)
            decreasekey(H, v)
```

Ví dụ

Hãy chạy thuật toán Dijkstra trên đồ thị sau. Sau mỗi bước, hãy chỉ ra mảng `prev`, phần tử và khóa của hàng đợi ưu tiên.

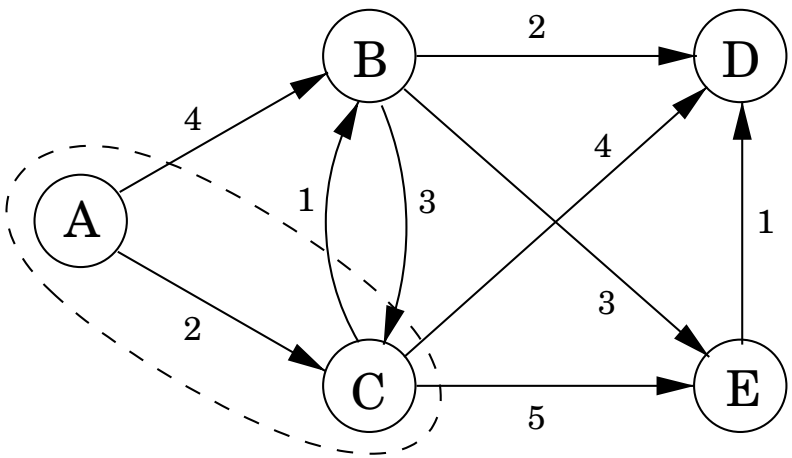


Ví dụ: Bước 1



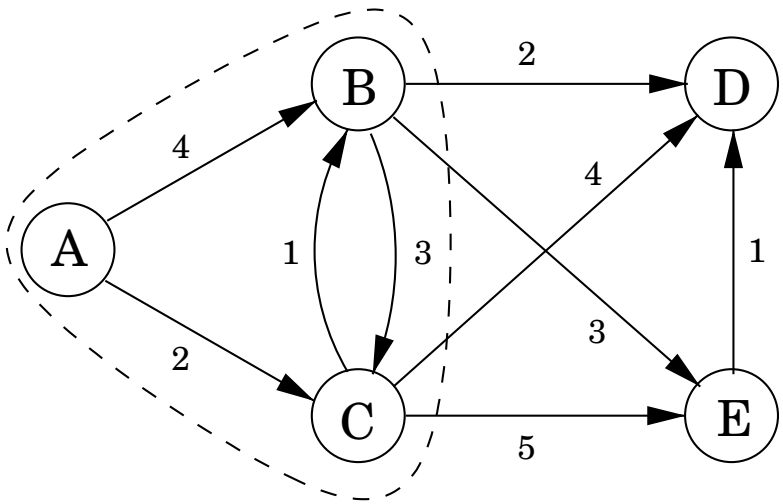
A: 0	D: ∞
B: 4	E: ∞
C: 2	

Ví dụ: Bước 2



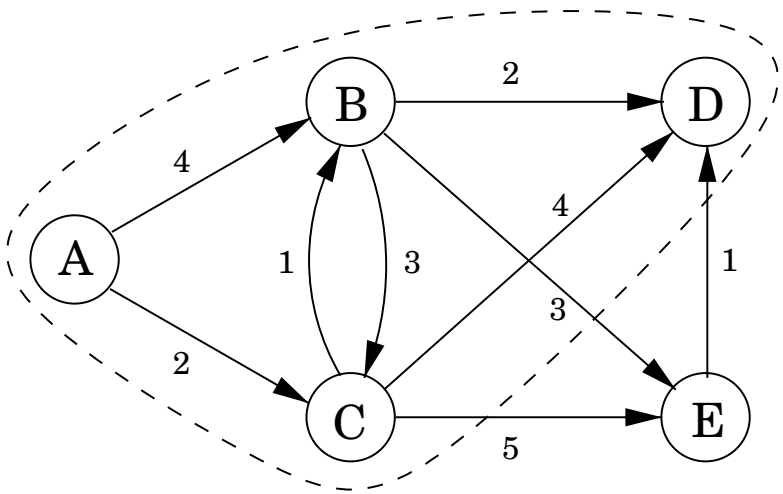
A: 0	D: 6
B: 3	E: 7
C: 2	

Ví dụ: Bước 3



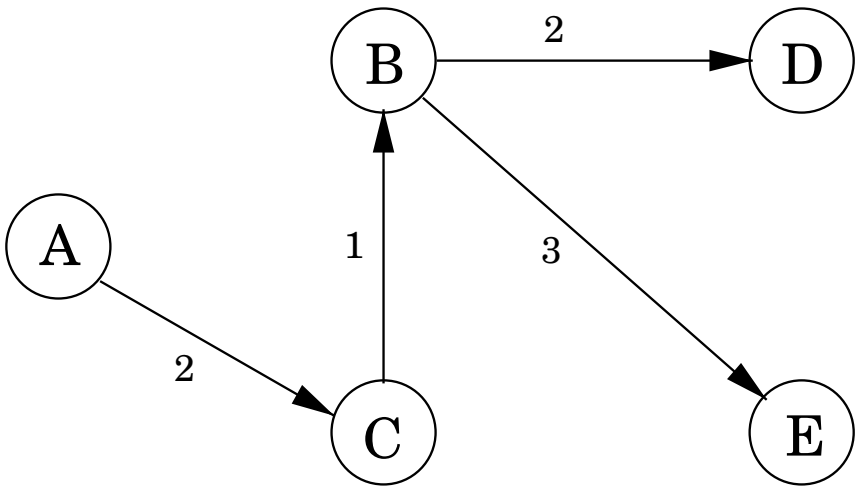
A: 0	D: 5
B: 3	E: 6
C: 2	

Ví dụ: Bước 4



A: 0	D: 5
B: 3	E: 6
C: 2	

Ví dụ: Mảng prev



u	prev[u]
A	nil
B	C
C	A
D	B
E	B

Nội dung

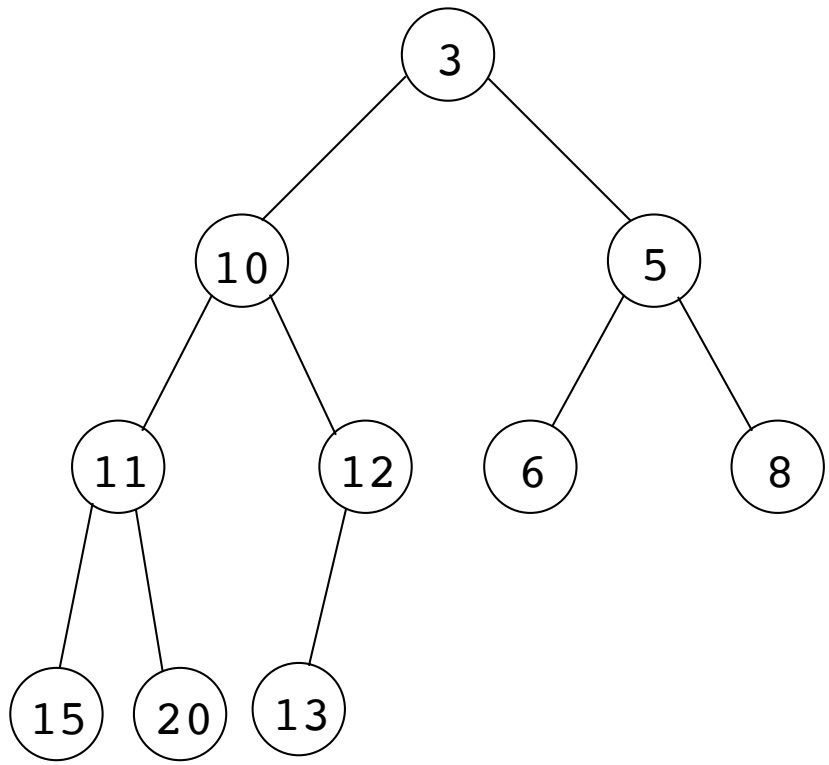
Cài đặt hàng đợi ưu tiên dùng mảng

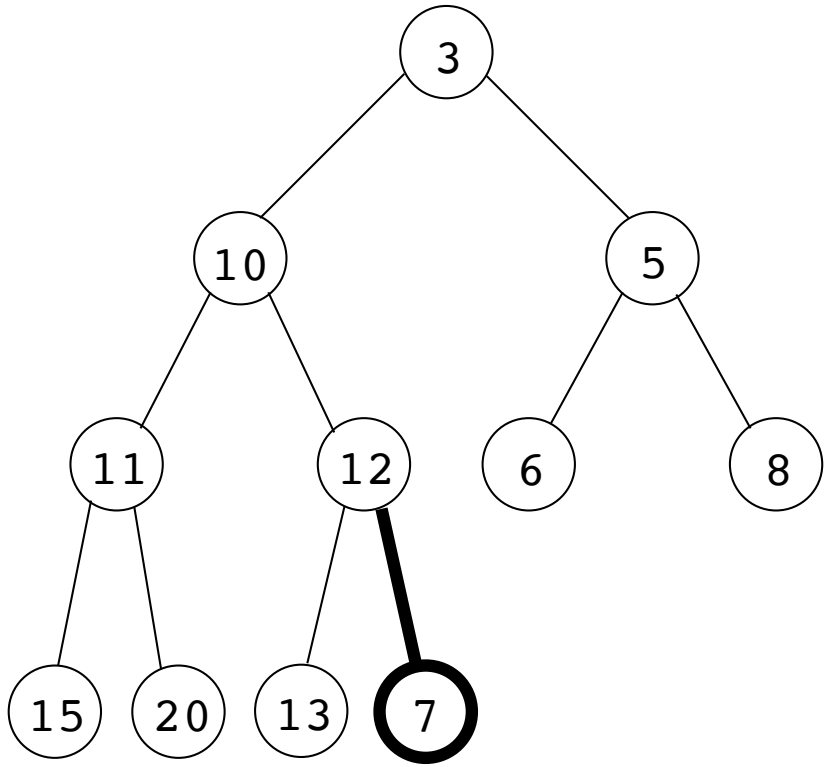
- Dùng mảng lưu trữ giá trị khóa cho mỗi phần tử (đỉnh của đồ thị).
- Phép toán `insert` và `decreasekey` chạy trong $O(1)$.
- nhưng `deletemin` chạy trong $O(|V|)$!
- Với cách cài đặt này, thuật toán Dijkstra chạy trong $O(|V|^2)$

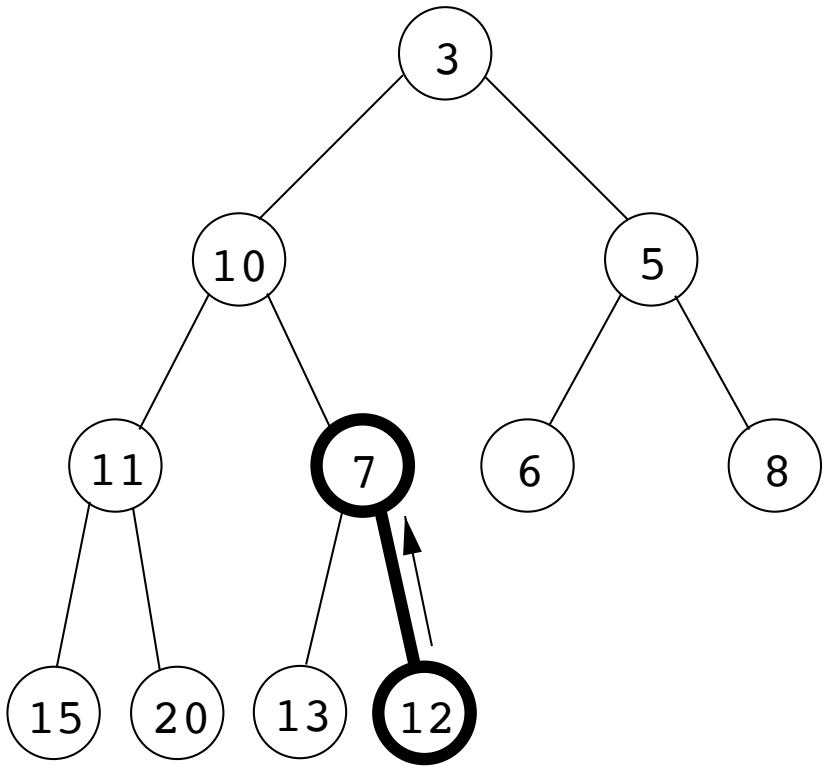
Dùng Binary Heap

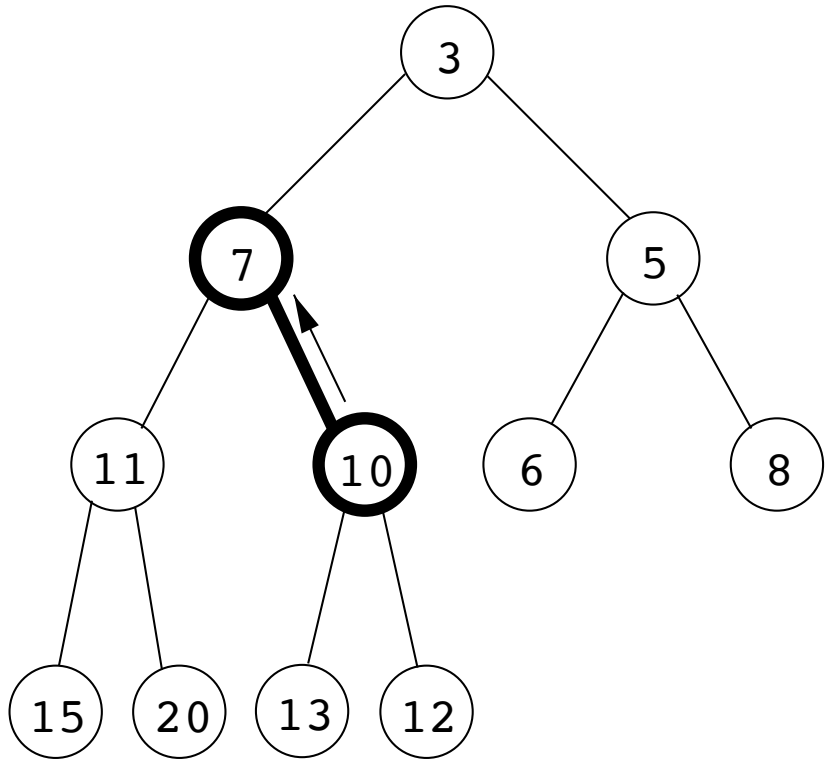
- Các phần tử được lưu trong cây nhị phân đầy đủ: Mỗi mức trên cây phải được điền đầy từ trái qua phải và phải đầy trước khi thêm mức tiếp theo.
- Ràng buộc: Giá trị khóa của nút trên cây phải nhỏ hơn hoặc bằng giá trị khóa của các con.
- Các phép toán `insert`, `decreasekey`, và `deletemin` chạy trong $O(\log |V|)$

Ví dụ: Thêm phần tử với khóa 7 vào binary heap

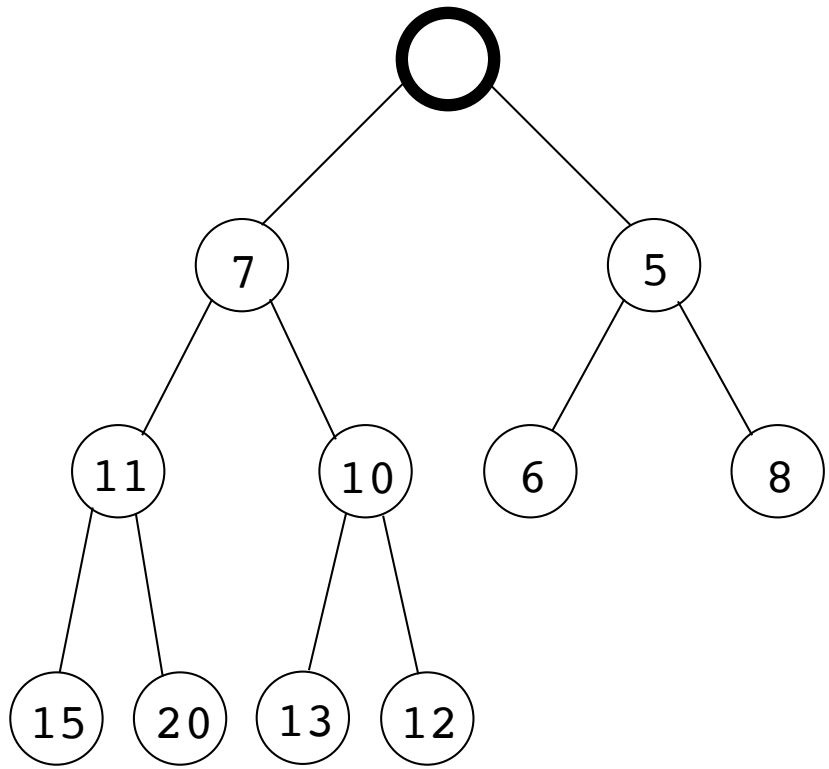


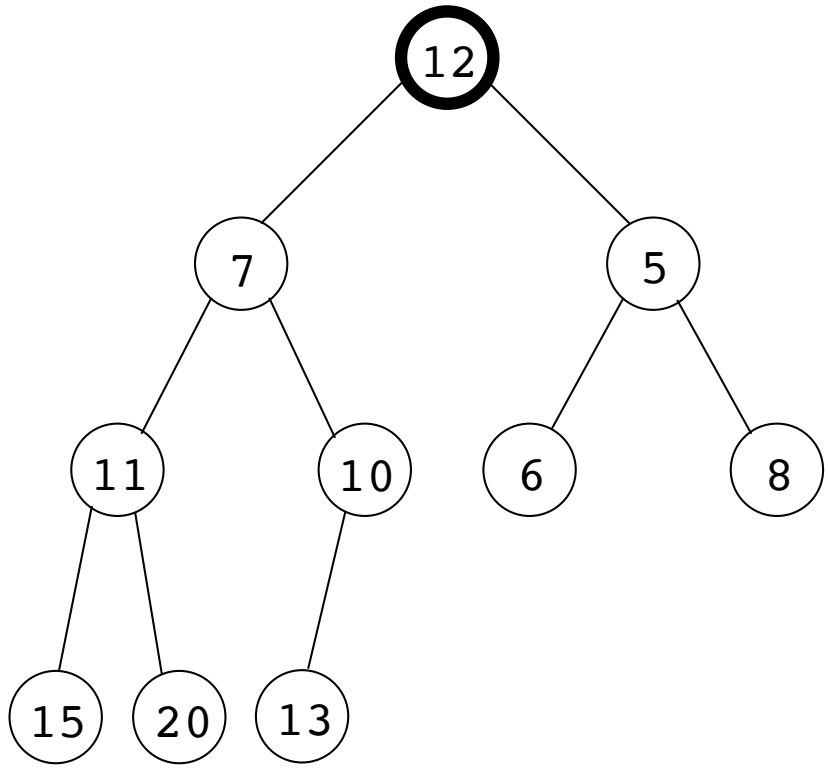

3
10
5
11
12
6
8
15
20
13
7

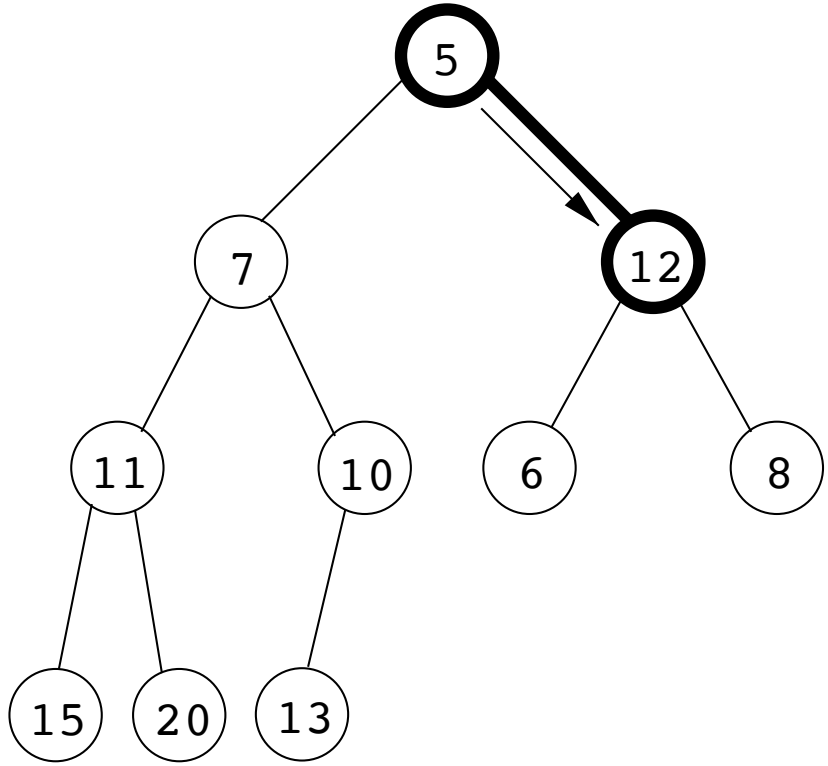

3
10
5
11
7
6
8
15
20
13
12

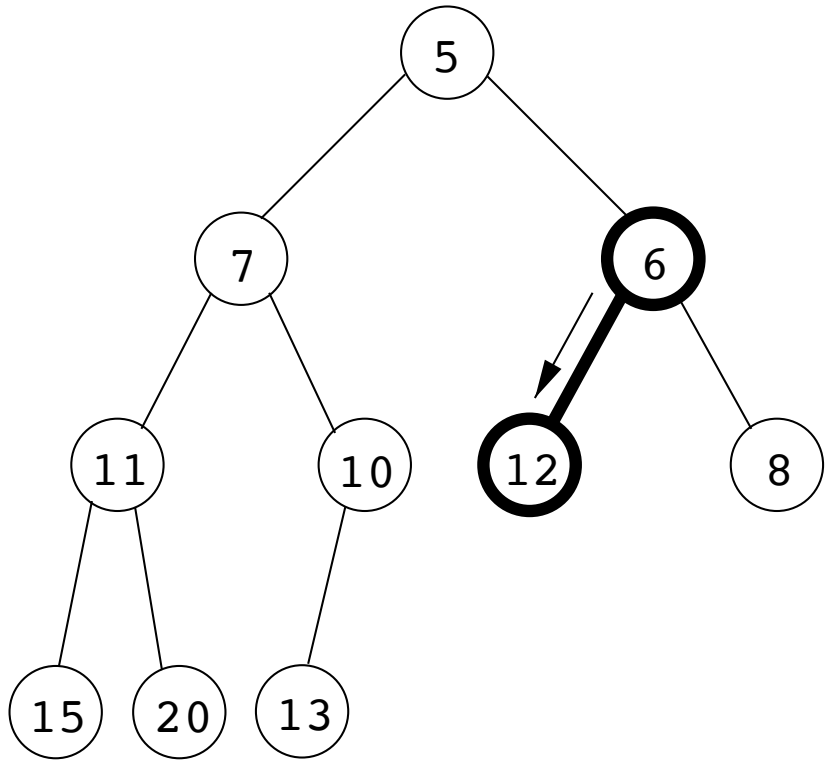

3
7
5
11
10
6
8
15
20
13
12

Ví dụ: Xóa phần tử ở gốc (`deletemin`)




12
7
5
11
10
6
8
15
20
13

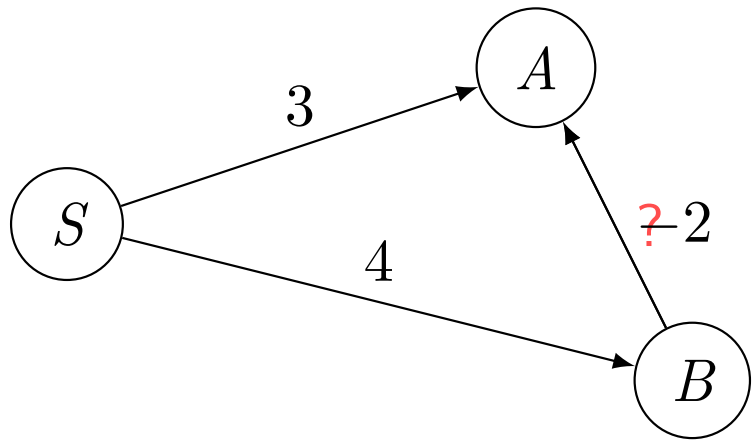

5
7
12
11
10
6
8
15
20
13


5
7
6
11
10
12
8
15
20
13

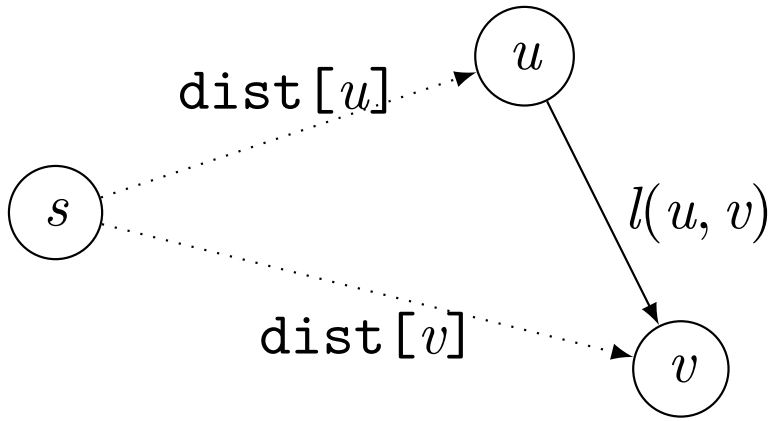
Nội dung

Câu hỏi

Liệu ta đã quyết định được đường đi ngắn nhất từ S đến A bằng bao nhiêu chưa?



Giải pháp



procedure update$((u, v) \in E)$

$\texttt{dist}(v) = \min\{\texttt{dist}(v), \texttt{dist}(u) + l(u, v)\}$

Ý tưởng: Khoảng cách từ s tới v không thể lớn hơn khoảng cách từ s tới u, cộng với $l(u, v)$.

`procedure update`$((u, v) \in E)$

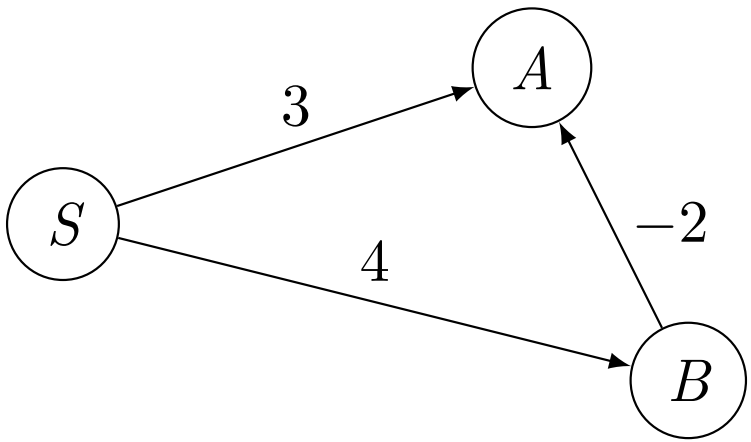
$\texttt{dist}(v) = \min\{\texttt{dist}(v), \texttt{dist}(u) + l(u, v)\}$

Câu hỏi

Giả sử

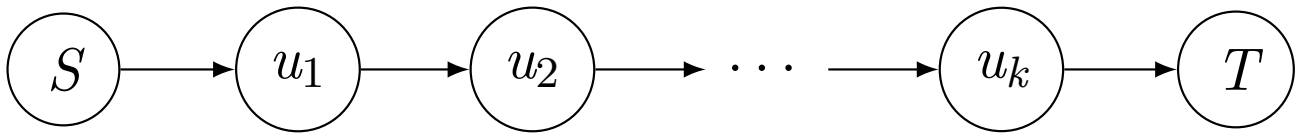
$$\texttt{dist}(S) = \texttt{dist}(A) = \texttt{dist}(B) = \infty.$$

Ta cần dùng hàm `update()` mấy lần thì tìm được khoảng cách ngắn nhất từ S đến A?



Dãy update để tìm đường đi ngắn nhất

- Lấy một đỉnh T bất kỳ và xét đường đi ngắn nhất từ S tới T:



- Đường đi này có nhiều nhất $|V| - 1$ đỉnh. Tại sao?

Quan sát

Mọi cách gọi update trên dãy cạnh có chứa dãy con

$$(S, u_1), (u_1, u_2), \ldots, (u_k, T)$$

đều tính đúng khoảng cách từ S tới T. Tại sao?

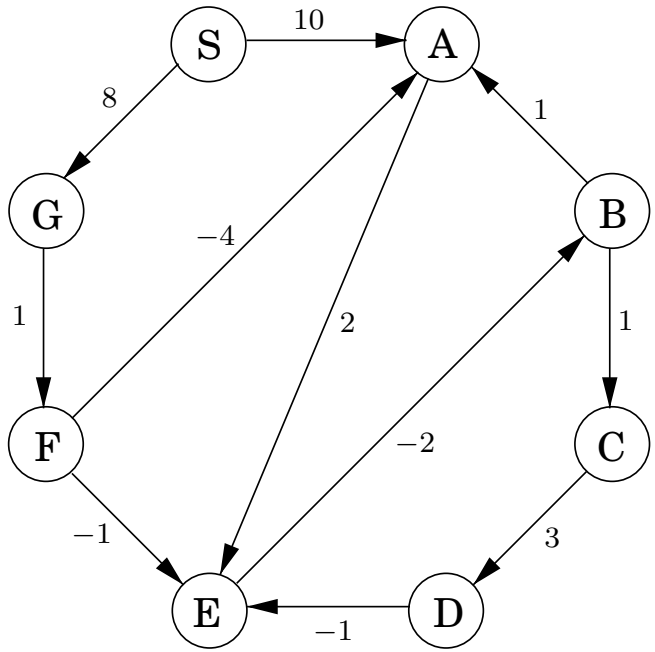
Thuật toán Bellman-Ford

```
procedure shortest-paths(G, l, s)
Input: đồ thị có hướng G = (V, E);
       độ dài các cạnh {l_e : e ∈ E}
       mà không có chu trình âm;
       đỉnh s ∈ V
Output: Với mỗi đỉnh u đến được từ s,
       dist(u) = khoảng cách từ s tới u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil                          (chưa sử dụng)

dist(s) = 0
repeat |V| − 1 times:
    for all edges e ∈ E:
        update(e)
```

Ví dụ



	Iteration							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Câu hỏi

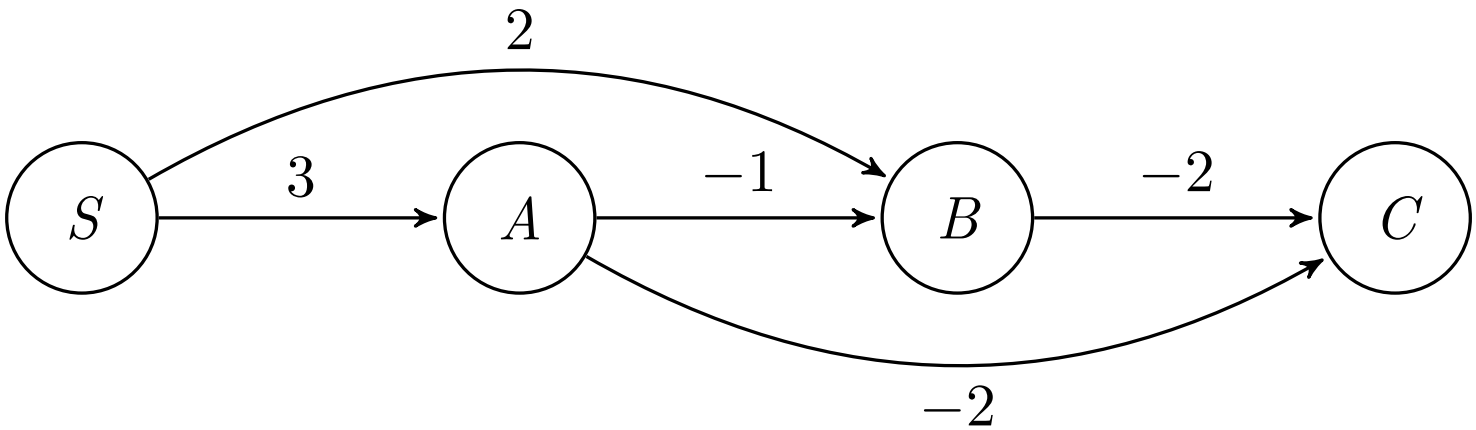
Làm thế nào để tìm được đường đi ngắn nhất theo thuật toán Bellman-Ford?

Kiểm tra sự tồn tại của chu trình độ dài âm?

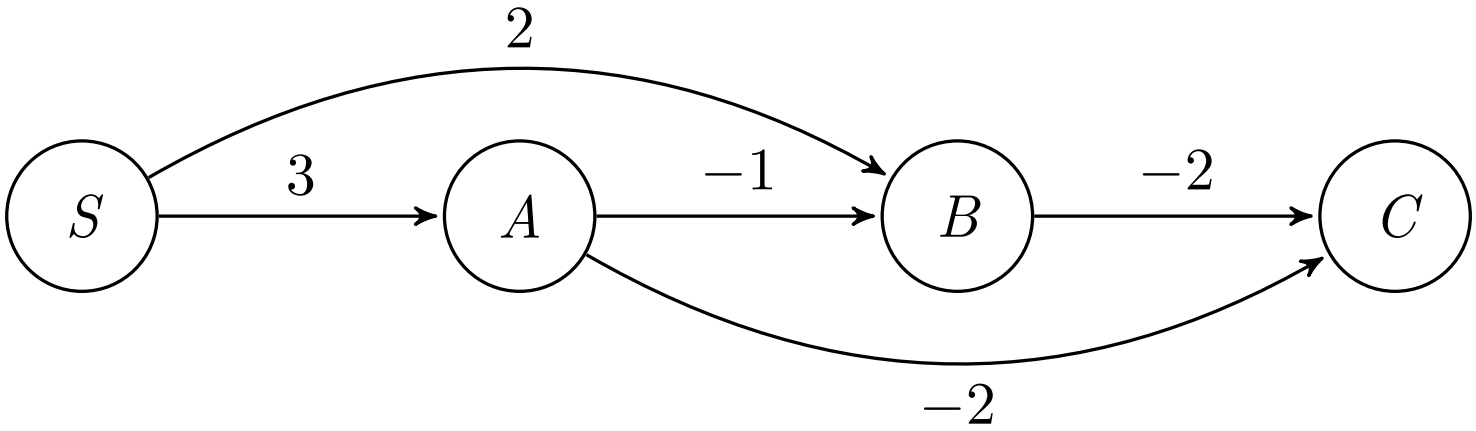
- Bài toán đường đi ngắn nhất từ s đến t sẽ không có ý nghĩa nếu từ s đến t có thể đi qua chu trình độ dài âm.
- Trong thuật toán Bellman-Ford, thay vì dừng sau $|V|-1$ vòng lặp, ta thực hiện thêm một lần nữa.
- Đồ thị có chu trình độ dài âm nếu và chỉ nếu tồn tại đỉnh v mà `dist[`v`]` vẫn bị giảm sau vòng cuối.

Nội dung

Thuật toán Bellman-Ford cho DAG



Trong DAG, cần `update` những cạnh nào?



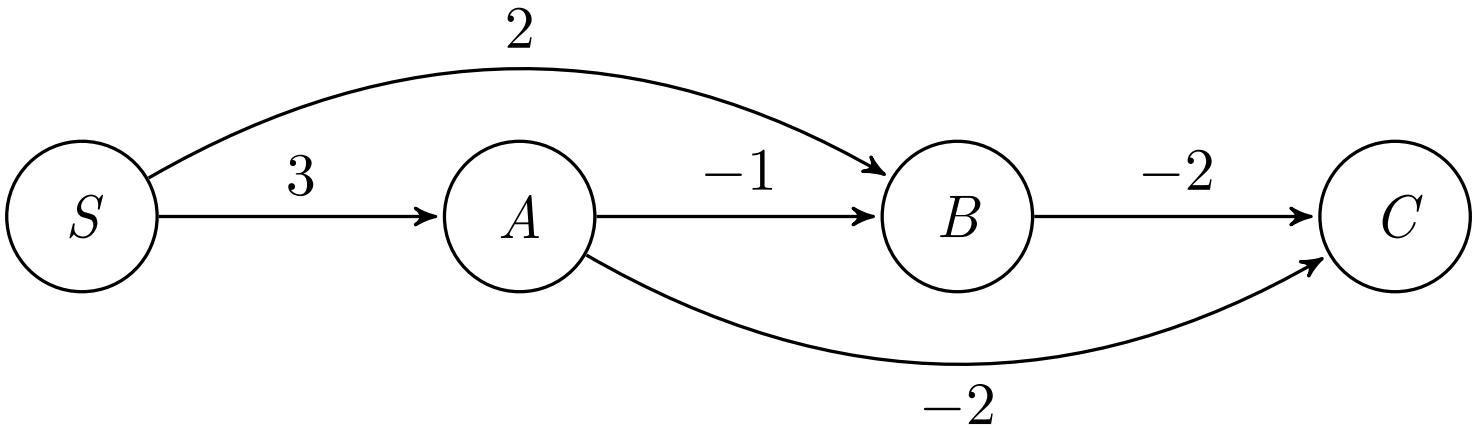
Tính chất

Với mọi đường đi của DAG, các đỉnh xuất hiện theo thứ tự topo.

```
procedure dag-shortest-paths(G, l, s)
Input: DAG G = (V, E);
       độ dài các cạnh {l_e : e ∈ E}; đỉnh s ∈ V
Output: Với mỗi đỉnh u đến được từ s, dist(u) được
đặt bằng
       khoảng cách từ s tới u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil

dist(s) = 0
Sắp topo các đỉnh của G
for each u ∈ V, theo thứ tự topo:
    for all edges (u, v) ∈ E:
        update(u, v)
```



Bài tập

Hãy liệt kê các lệnh `update` theo thuật toán Bellman-Ford cho DAG?