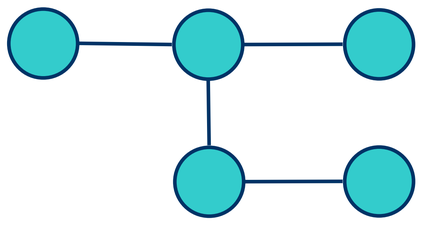
Cấu trúc dữ liệu và Giải thuật

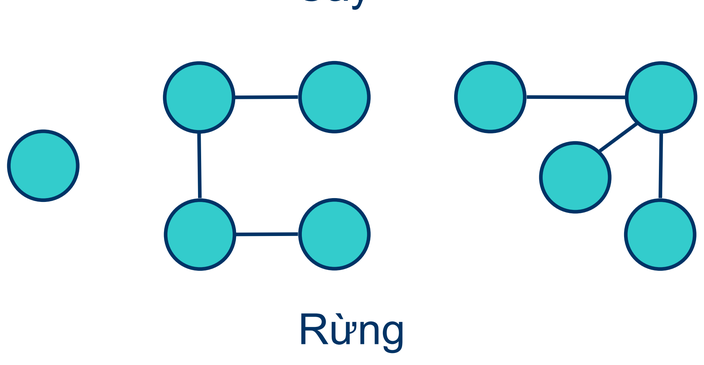
Chương V: Đồ thị (phần 2)

Cây và Rừng trong lý thuyết đồ thị

- Cây
  - Một đồ thị vô hướng liên thông
  - Không có chu trình
- Rừng
  - Một tập các cây phân biệt



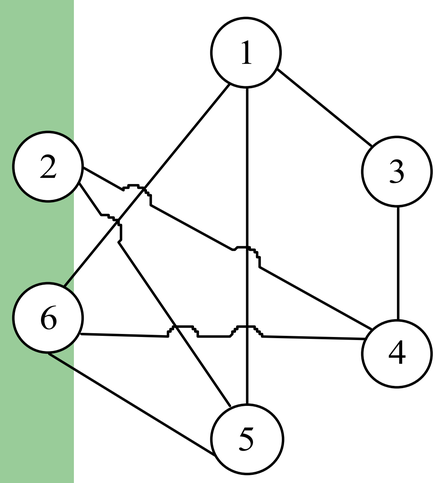
Cây



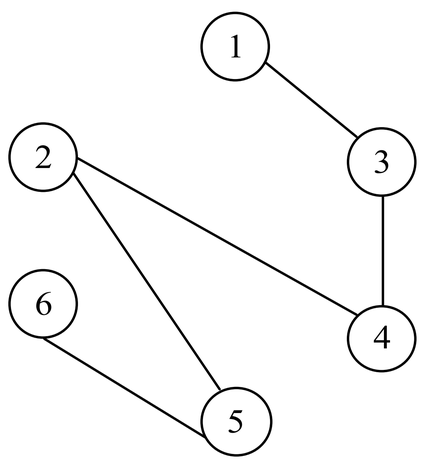
Rừng

Cây khung

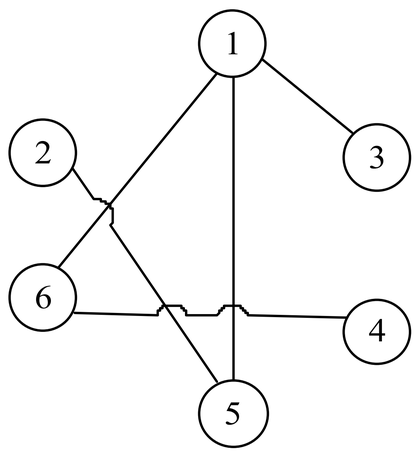
- Cho một đồ thị vô hướng, liên thông G
  - Cây khung trên G là cây có chứa tất cả các đỉnh trong G



Đồ thị



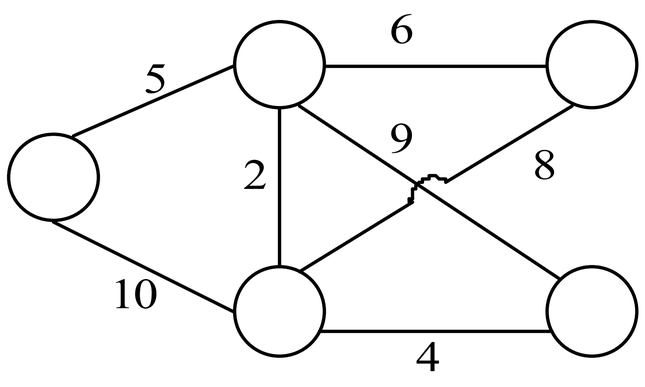
Cây khung



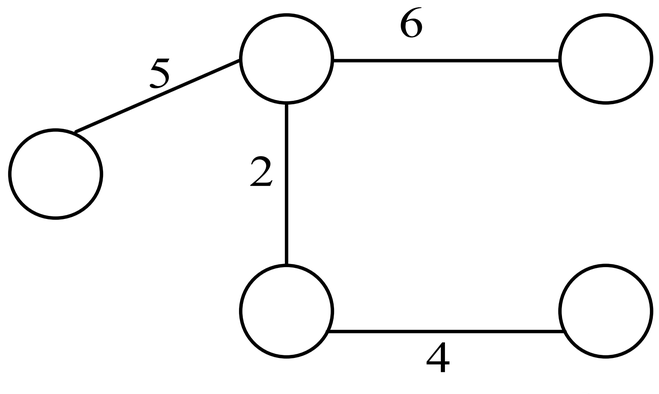
Cây khung

Bài toán tìm cây khung cực tiểu

- Cho một đồ thị vô hướng, liên thông có trọng số
- Giá trị của một cây khung là tổng trọng số của các cung trong cây
- Tìm một cây khung với giá trị nhỏ nhất trên đồ thị



Đồ thị đầu vào

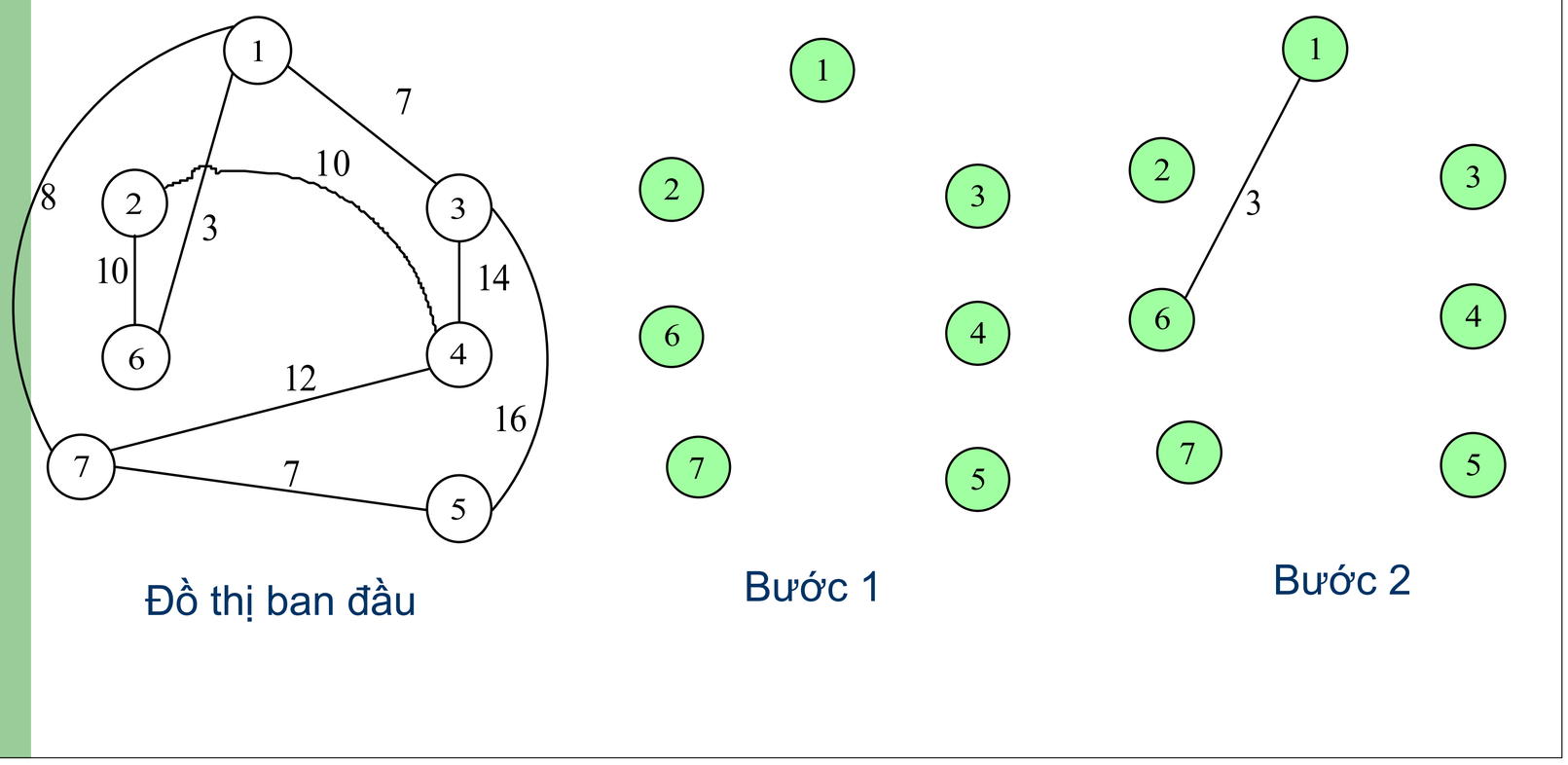


Cây khung cực tiểu

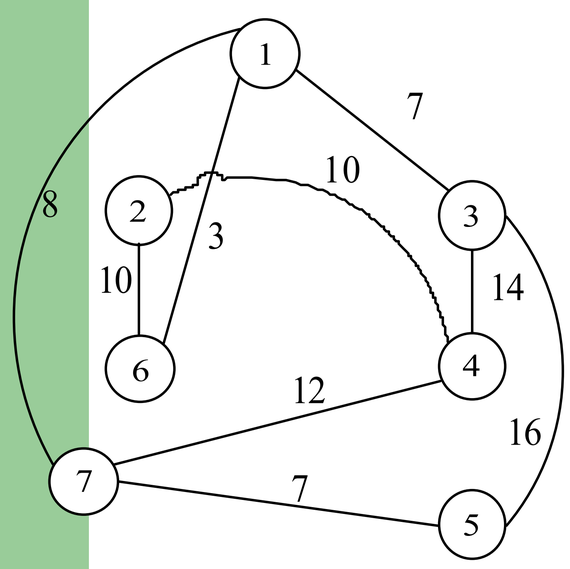
Giải thuật Kruskal - MST

- Ý tưởng
  - Lần lượt thêm vào cây khung cần tìm các cung có trọng số nhỏ nhất có được tại một thời điểm nếu cung đó không tạo thành chu trình trên phần cây khung đang tạm có

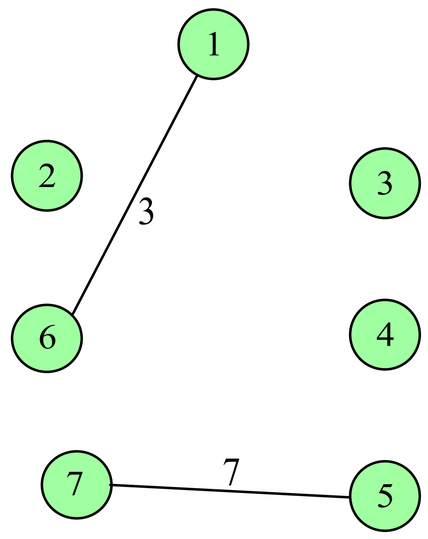
Giải thuật Kruskal-MST



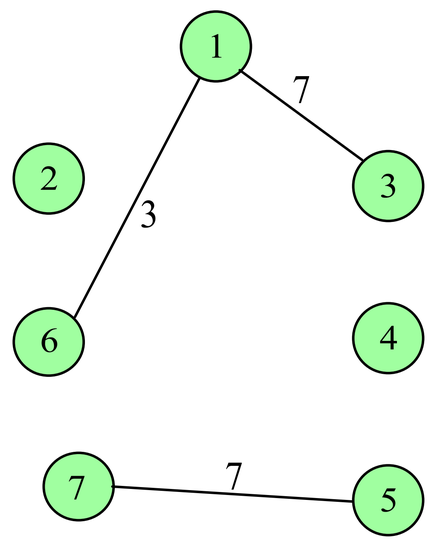
Giải thuật Kruskal – MST



Đồ thị ban đầu

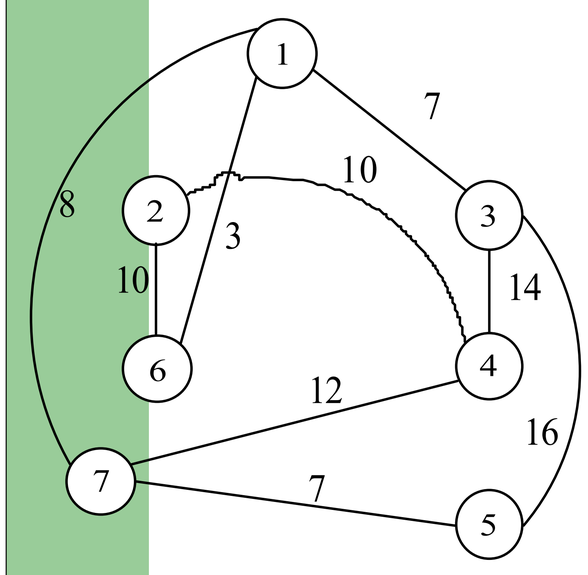


Bước 3

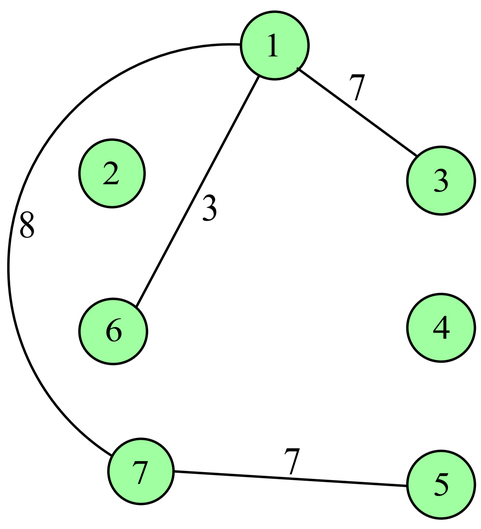


Bước 4

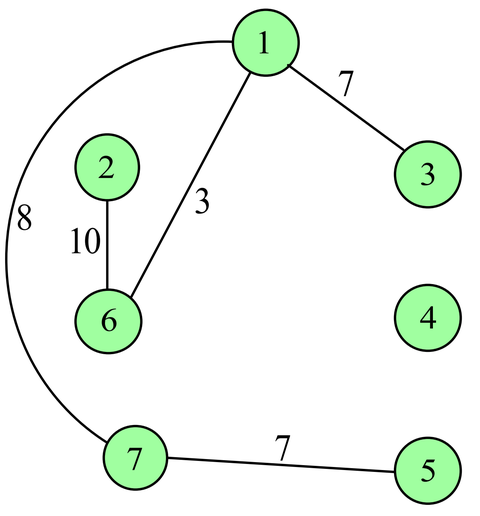
Giải thuật Kruskal - MST



Đồ thị ban đầu

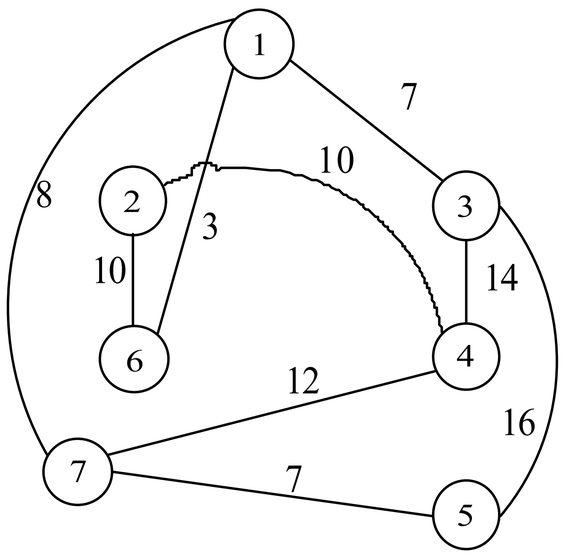


Bước 5

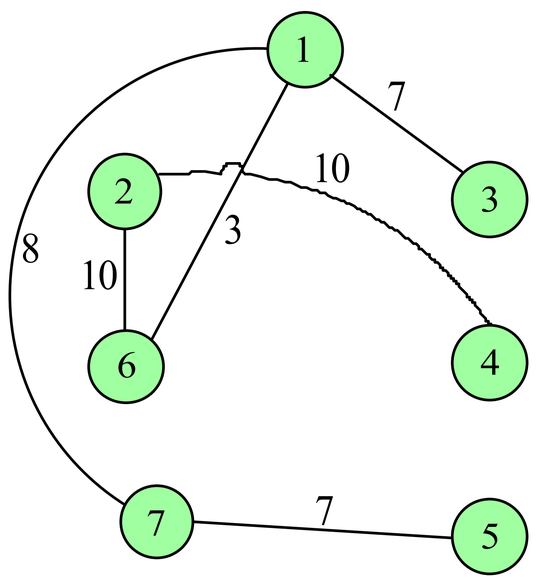


Bước 6

Giải thuật Kruskal - MST



Đồ thị ban đầu



Bước 7-
Cây khung cực tiểu

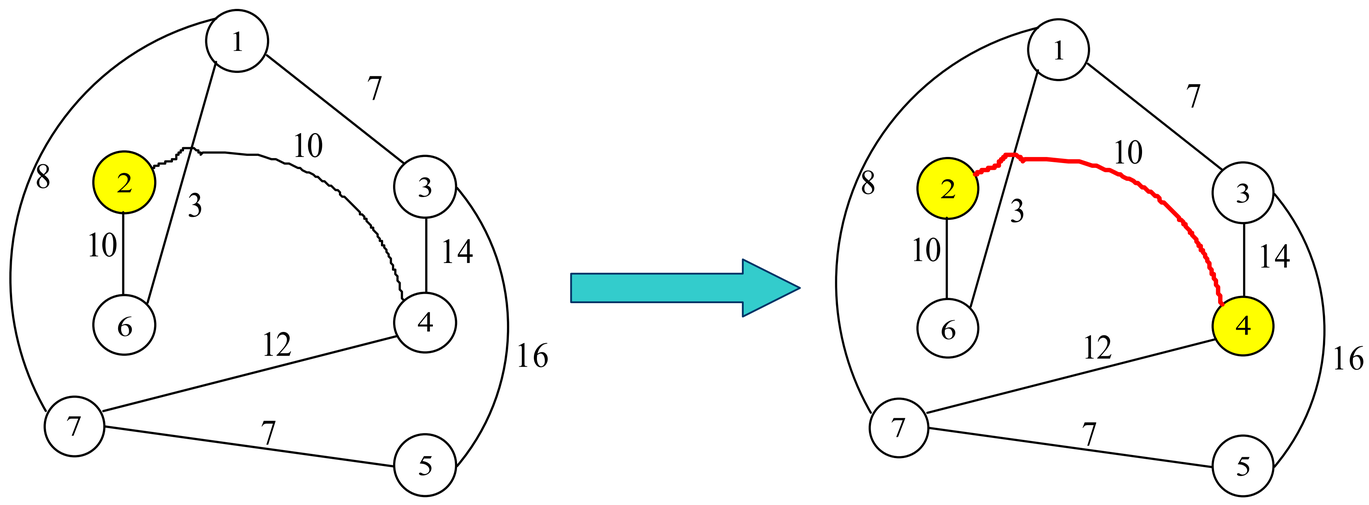
Giải thuật Kruskal-MST

Algorithm KRUSKAL(G) {đồ thị G có n đỉnh}
1. {Khởi tạo các cụm ban đầu, mỗi cụm chứa 1 đỉnh của đồ thị }
for each vertex v in G do C(v) ← {v}.
2. Khởi tạo một Queue Q chứa các cung trong G, sắp xếp theo chiều tăng dần của trọng số.
3. {Khởi tạo cây khung ban đầu rỗng} T ← ∅
4. {Lần lượt xét các cung đưa vào trong cây khung cần tìm}
while T chứa ít hơn n-1 cung **do begin**
Lấy ra từ Q cung (u,v) có trọng số nhỏ nhất
C(v) là cụm chứa v, C(u) là cụm chứa u.
if C(v) ≠ C(u) **then begin**
T = T U {(u,v)}
Nhập C(u) với C(v)
end
end
return T

Giải thuật Prim - MST

- Ý tưởng
  - Xây dựng một cây khung bắt đầu từ một đỉnh xuất phát
  - Thời điểm ban đầu, đỉnh xuất phát là đỉnh duy nhất trong một cụm C
  - Từng bước thêm vào cụm C một đỉnh w đang ở ngoài C mà w có nối với 1 đỉnh u trong C thông qua một cung (u,w) có giá trị nhỏ nhất tại thời điểm đó.

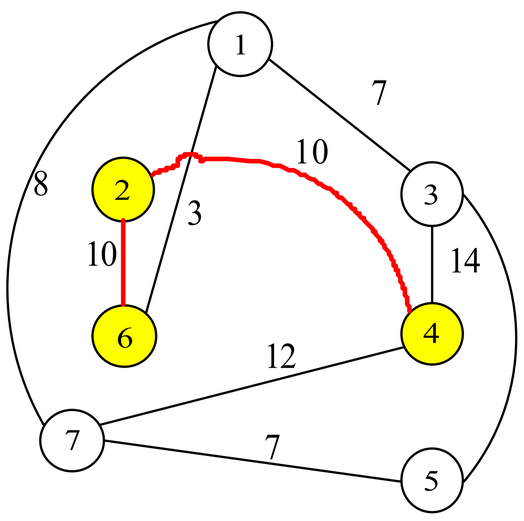
Giải thuật Prim - MST



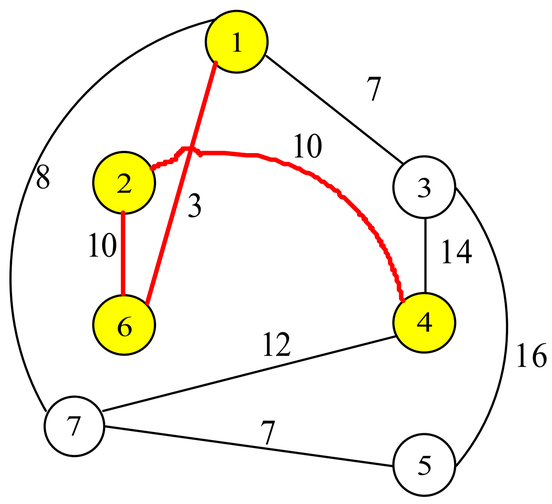
Đỉnh xuất phát được chọn
Là đỉnh số 2

Bước 1: Từ 2 có cung (2, 4) , (2,6) đều có trọng số 10.
Chọn (2,4) cho thêm vào cây khung

Giải thuật Prim - MST

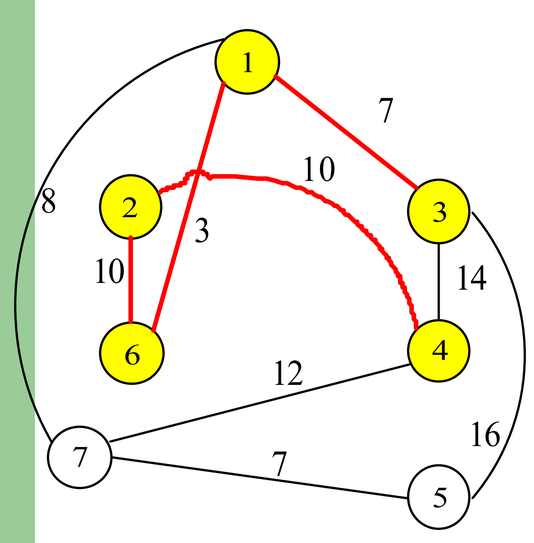


Bước 2:
Từ 2, 4 có các cung (2,6) , (4,7), (4,3)
Chọn (2,6) có đưa vào cây khung

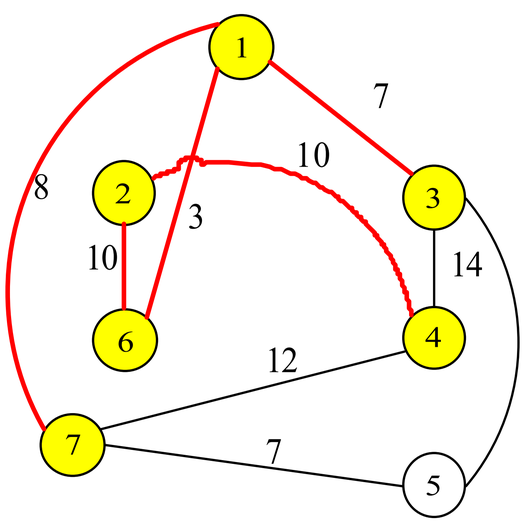


Bước 3: Chọn (6,1)

Giải thuật Prim - MST

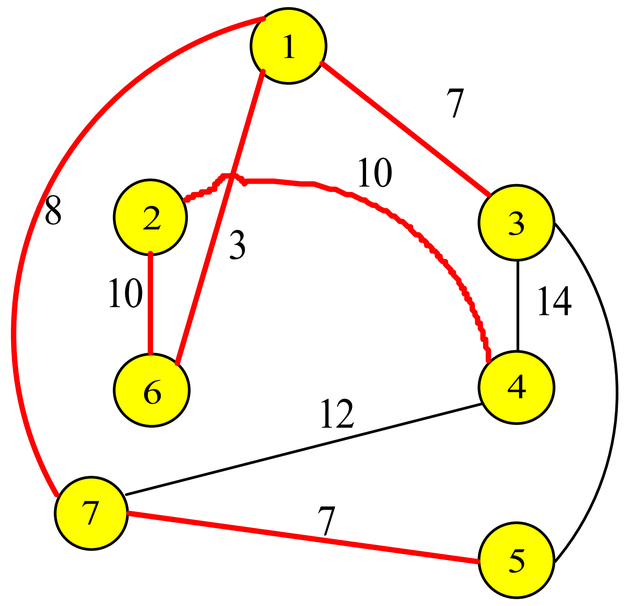


Bước 4: Chọn (1, 3)



Bước 5: Chọn (1, 7)

Giải thuật Prim - MST



Bước 6: Chọn (7,5). Tất cả các đỉnh trong đồ thị đều đã có trong cây khung

Giải thuật Prim - MST

Algorithm PRIM_MST(G, v)
1. {Khởi tạo cây khung ban đầu , chứa đỉnh v} T ← **{v}**
2. **Q = V – {v}** ; {Q là tập các đỉnh chưa ở trong cây khung}
3. { Thiết lập một mảng d chứa các giá trị trọng số của các cung để tiến hành chọn cung có giá trị nhỏ nhất nối một đỉnh trong cây với một đỉnh ngoài cây tại từng bước}
   d[v] = 0;
   for all w ∈ Q do begin
      if (tồn tại cung (v,w)) then d[w] = weight(v,w); else d[w] = ∞;
   end

Giải thuật Prim - MST

4. {Lần lượt lựa chọn đỉnh đưa vào trong cây khung}

While (Q ≠ rỗng) do begin

4.1 Xác định đỉnh u trong Q mà d[u] = min{d[w] | w ∈ Q} ;

4.2 Xác định cung (r,u) với r trong T và weight(r,u) = d[u];

4.3 T ← {(r,u)} ; Q = Q – {u};

{cập nhật lại các giá trị được lưu trong mảng d sau khi đã thêm u vào trong cây khung, mảng d mới sẽ tiếp tục sử dụng trong bước lựa chọn tiếp theo}

4.4 for all w ∈ Q do d[w] = min (d[w], weight(u,w));

End;

Bài toán tìm đường đi ngắn nhất

- Tìm đường đi ngắn nhất giữa 1 cặp đỉnh (i,j)
- Tìm đường đi ngắn nhất từ 1 đỉnh nguồn tới tất cả các đỉnh còn lại
- Tìm đường đi ngắn nhất giữa mọi cặp đỉnh

Giải thuật Dijkstra

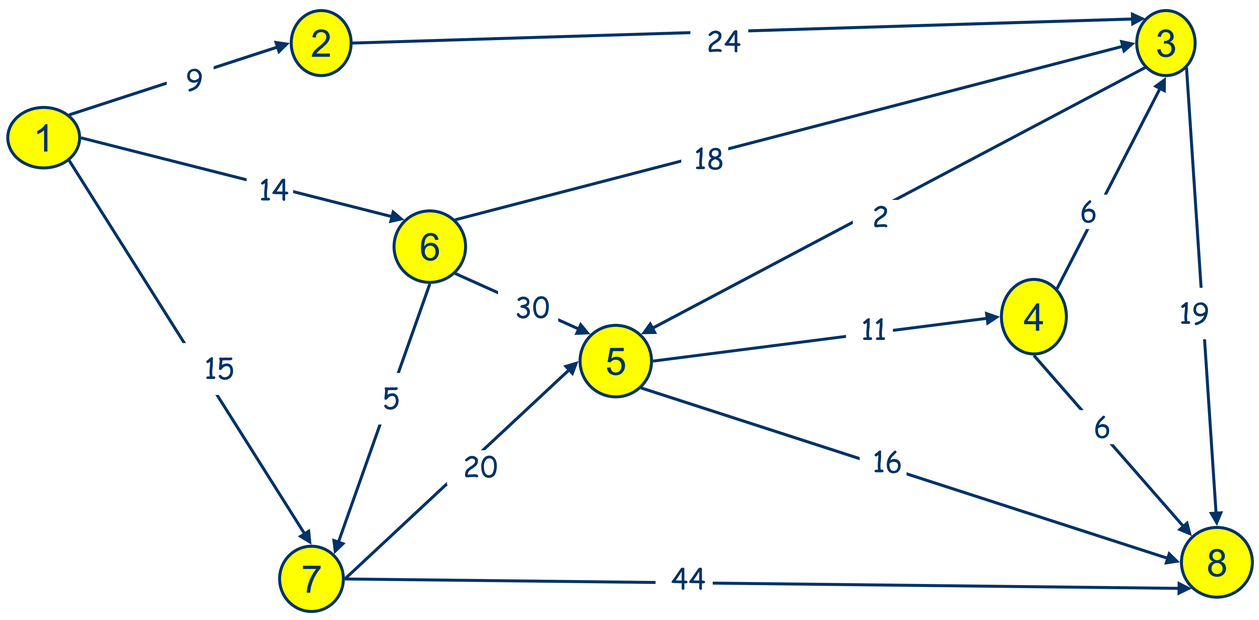
- Đặc trưng
  - Giải quyết bài toán tìm đường đi ngắn nhất giữa 1 cặp đỉnh và bài toán tìm đường đi ngắn nhất từ một nguồn tới mọi đích
  - Chỉ áp dụng trên đồ thị có trọng số dương
- Ý tưởng:
  - Với mỗi đỉnh v sẽ duy trì các thông số sau
    - D[v] : Khoảng cách ngắn nhất biết được tại thời điểm hiện tại từ đỉnh nguồn s tới đỉnh v.
    - P[v] : Đỉnh trước của đỉnh v trên đường đi từ đỉnh nguồn s tới v

Giải thuật Dijkstra

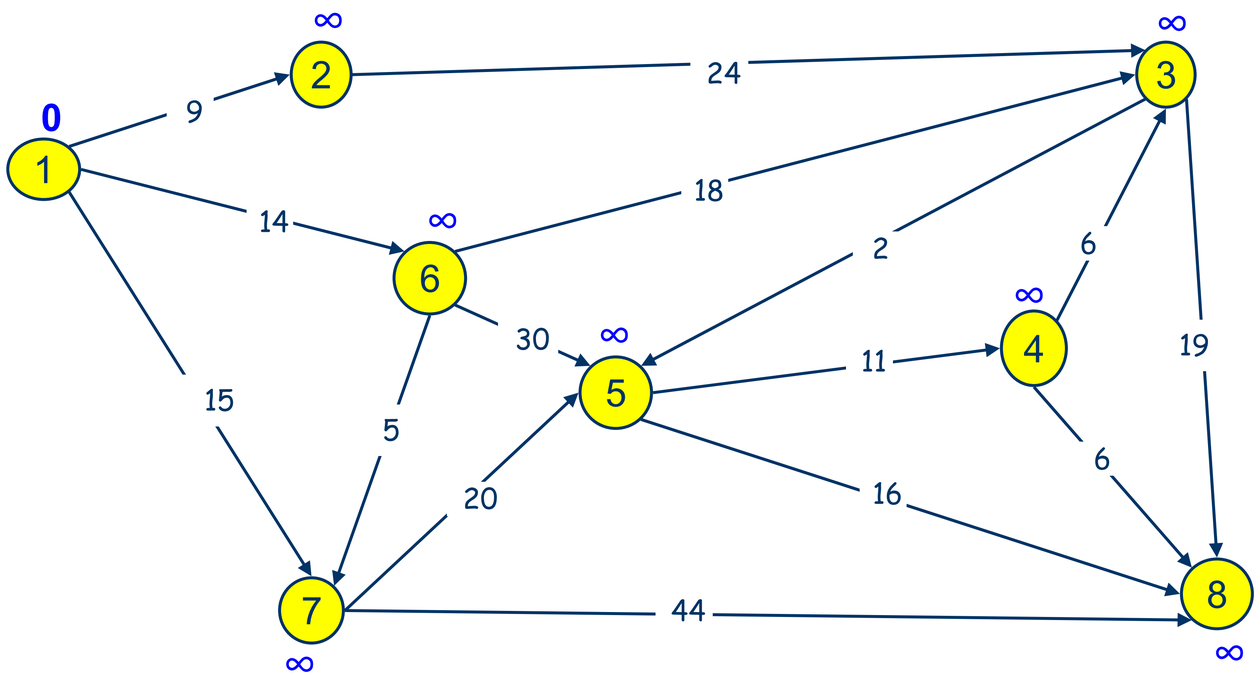
- Thực hiện
  - Duy trì một cụm C chứa các đỉnh, cụm này lúc đầu chứa đỉnh xuất phát đã cho. Dần dần thêm các đỉnh vào trong cụm
  - Tại mỗi bước của giải thuật
    - xác định đỉnh u chưa ở trong C có giá trị d[u] nhỏ nhất đưa vào trong C.
    - Cập nhật lại giá trị d của các đỉnh lân cận của u.

Giải thuật Dijkstra

- Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh khác

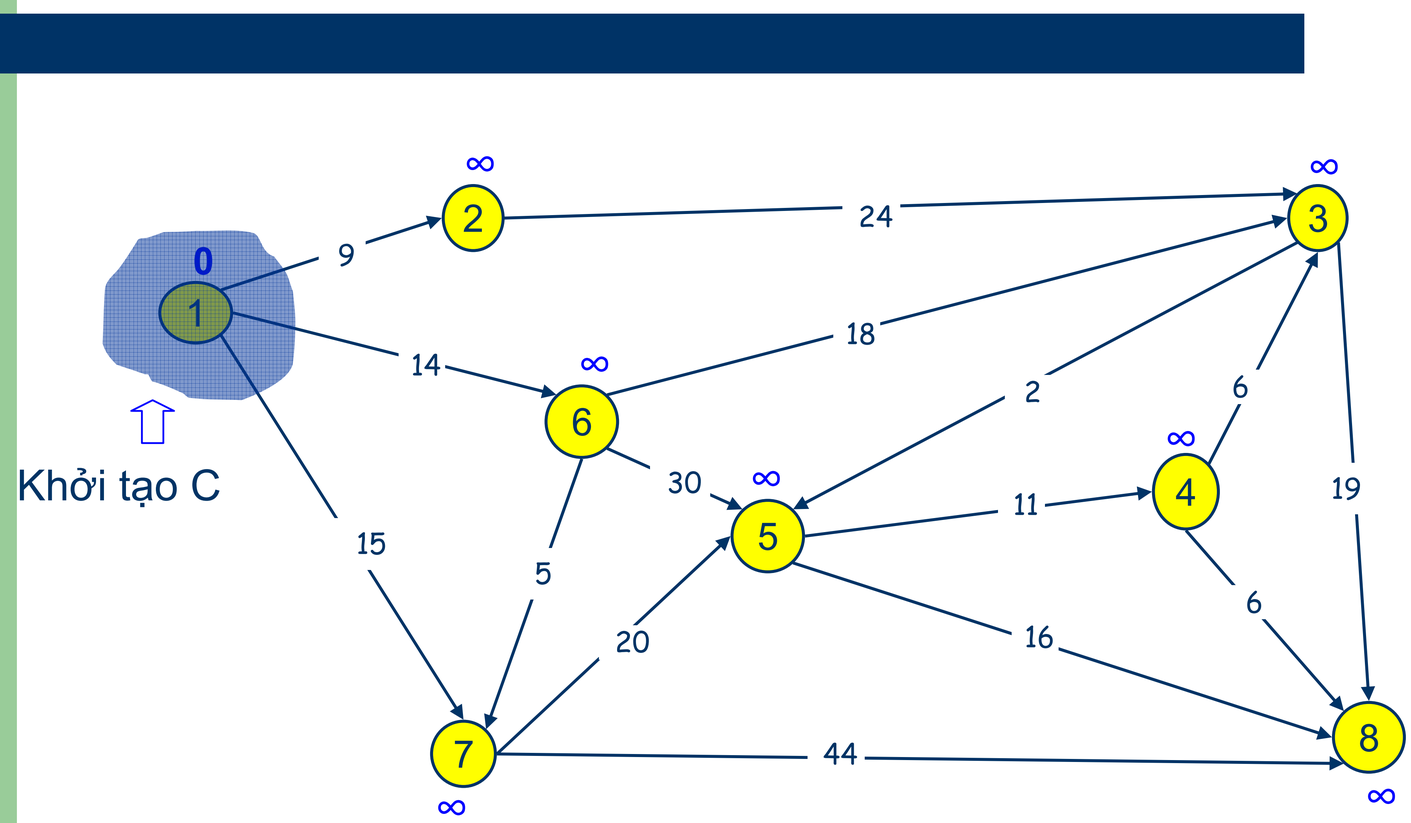


Giải thuật Dijkstra

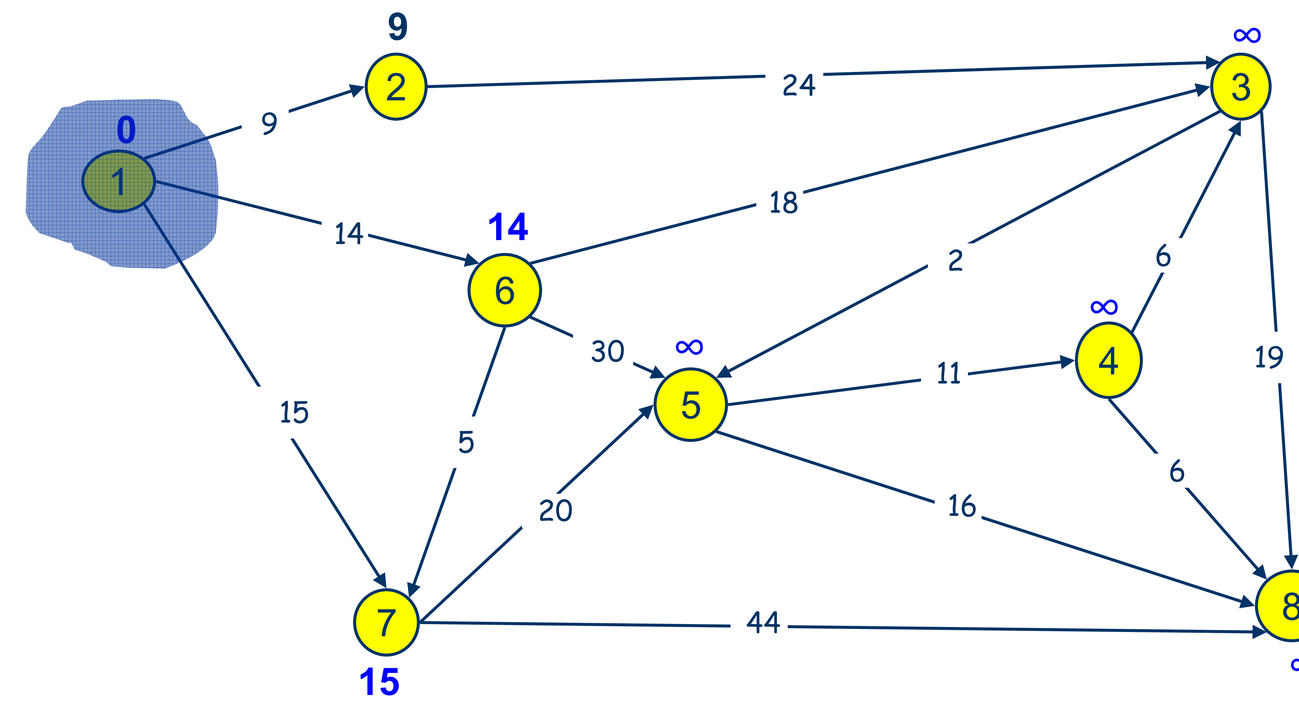


Khởi tạo các giá trị d cho tất cả các đỉnh

Giải thuật Dijkstra

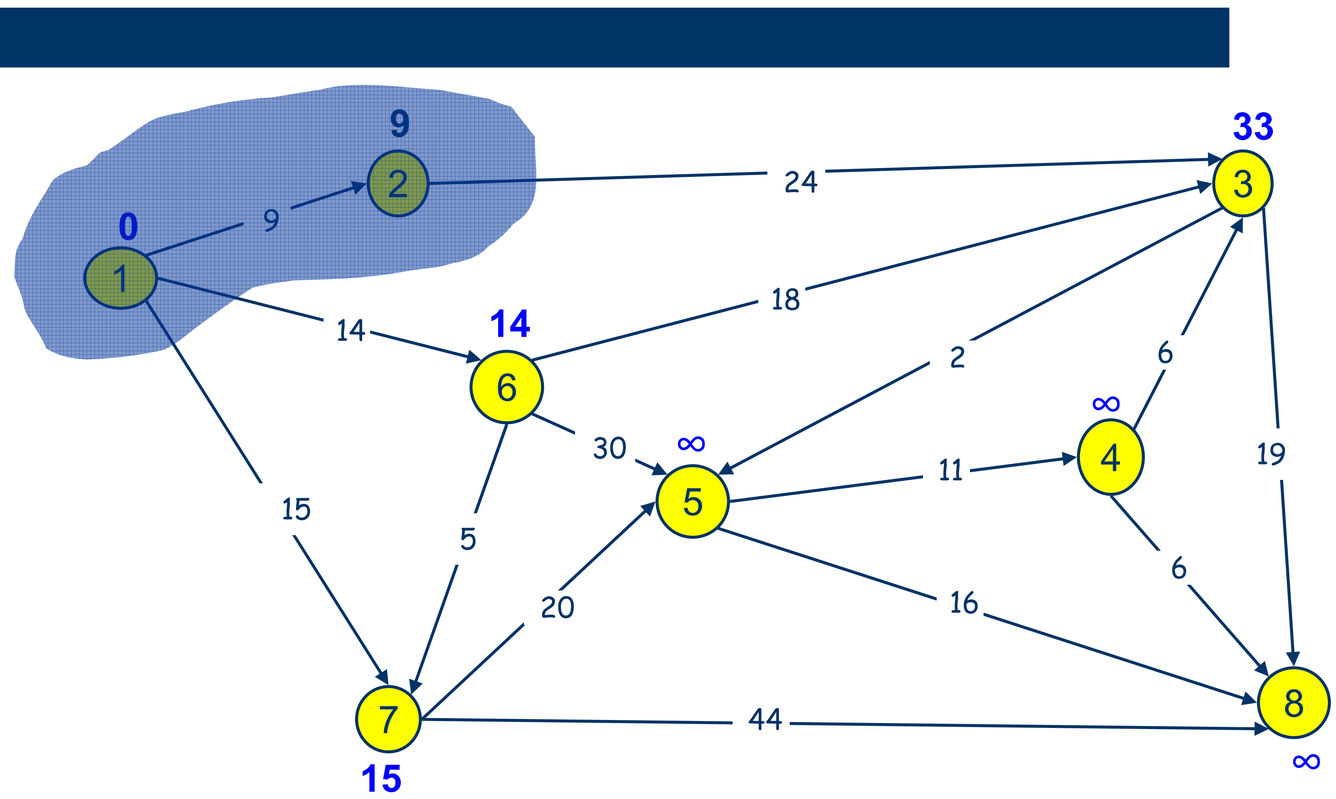


Giải thuật Dijkstra



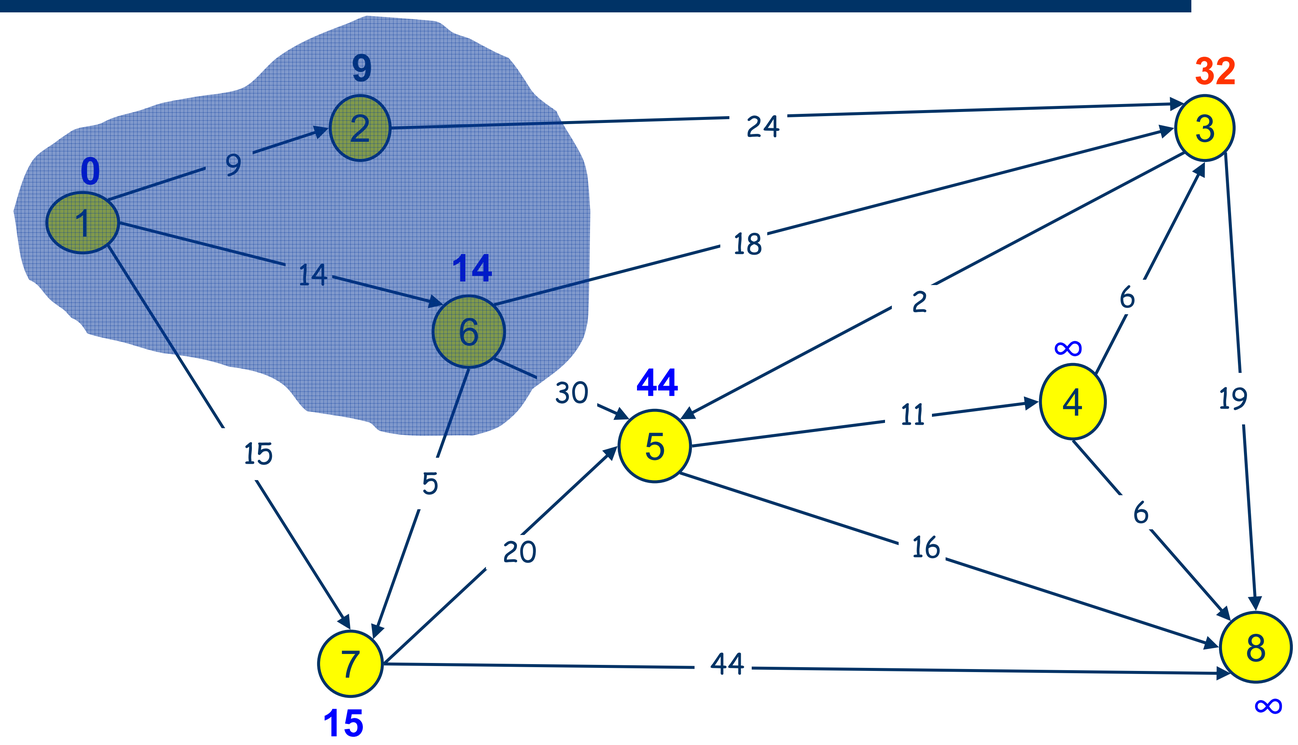
Cập nhật các giá trị d[2] = 9, d[6] = 14, d[7] = 15

Giải thuật Dijkstra



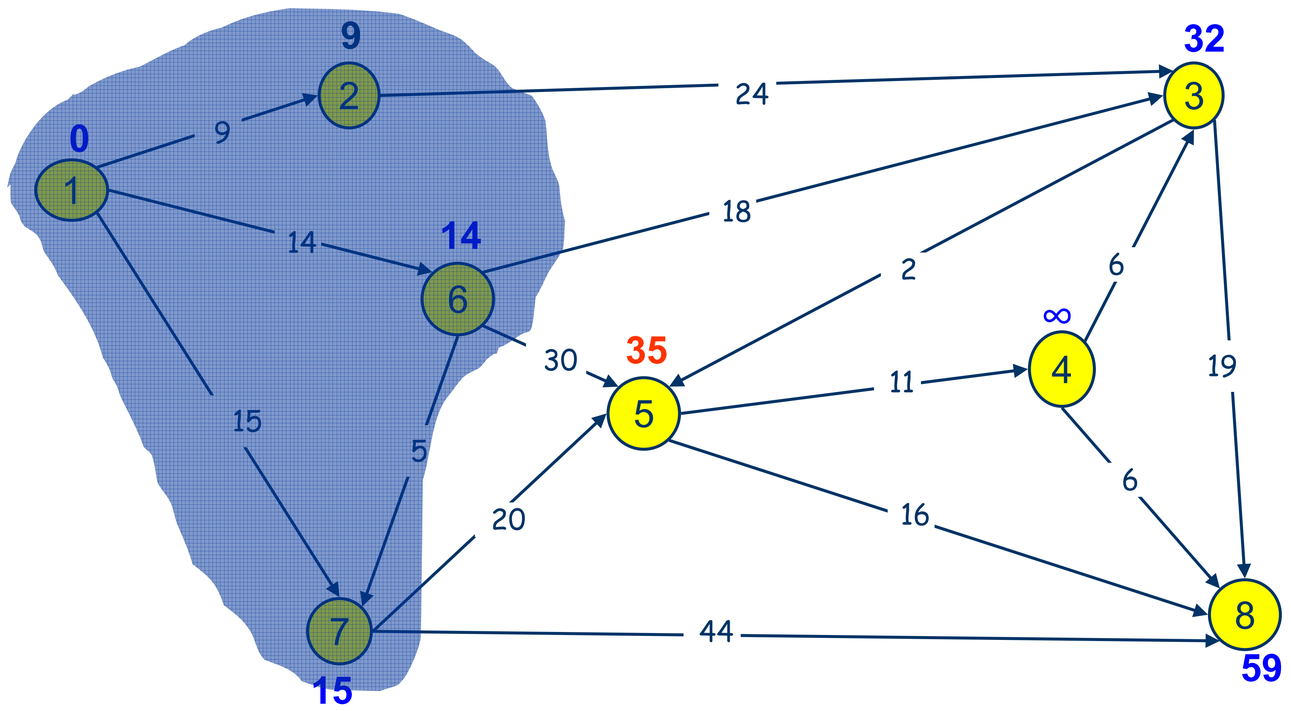
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 2 có độ dài 9
Cập nhật giá trị d của các đỉnh lân cận của 2

Giải thuật Dijkstra



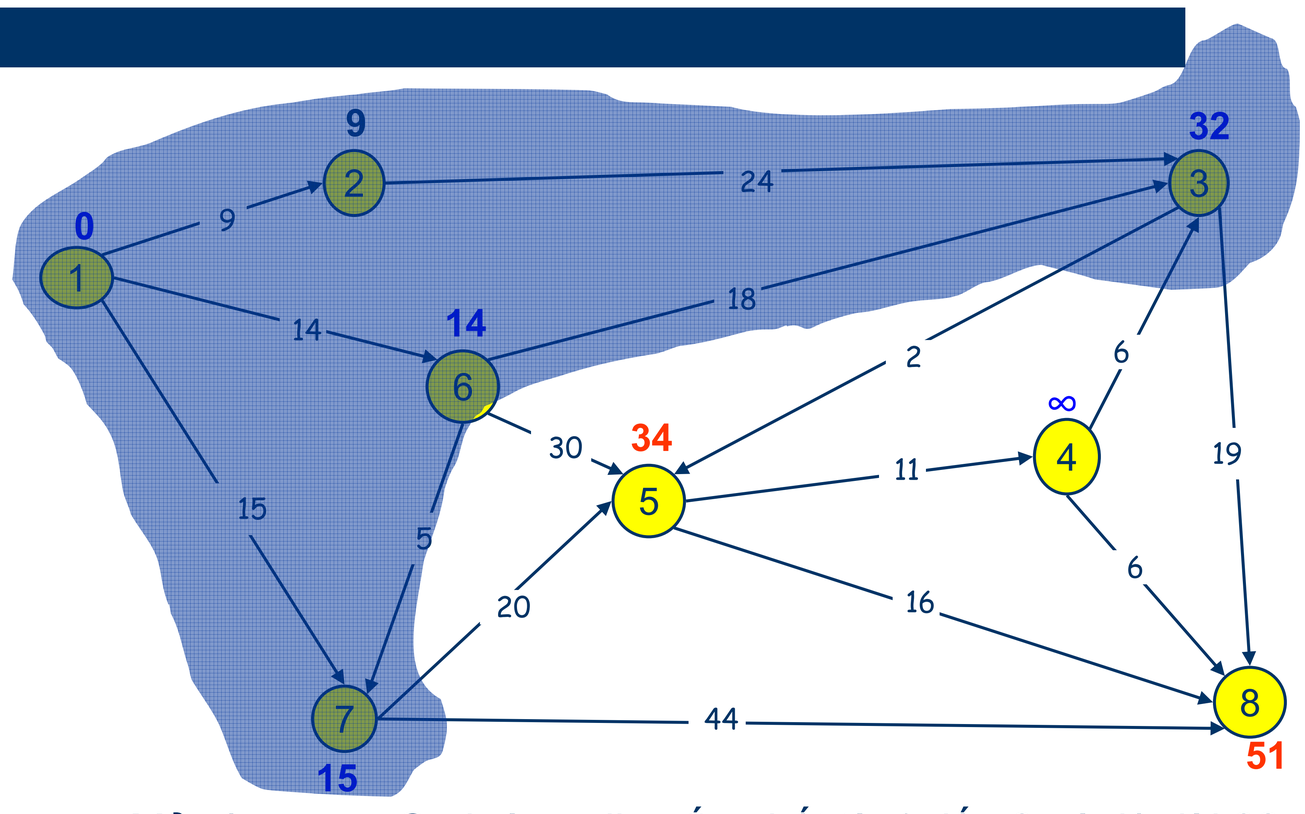
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 6 có độ dài 14
Cập nhật giá trị d của các đỉnh lân cận với 6

Giải thuật Dijkstra



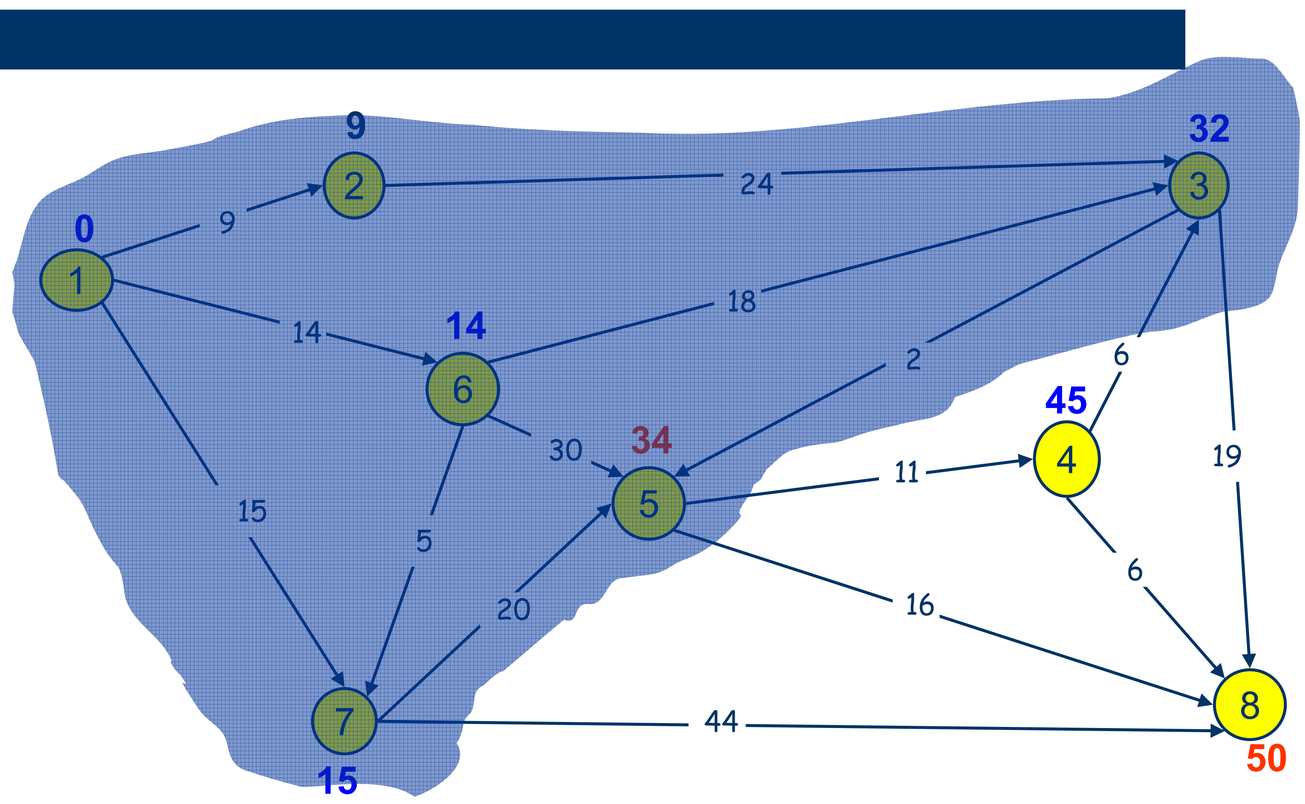
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 7 có độ dài 15
Cập nhật giá trị d của các đỉnh lân cận với 7

Giải thuật Dijkstra



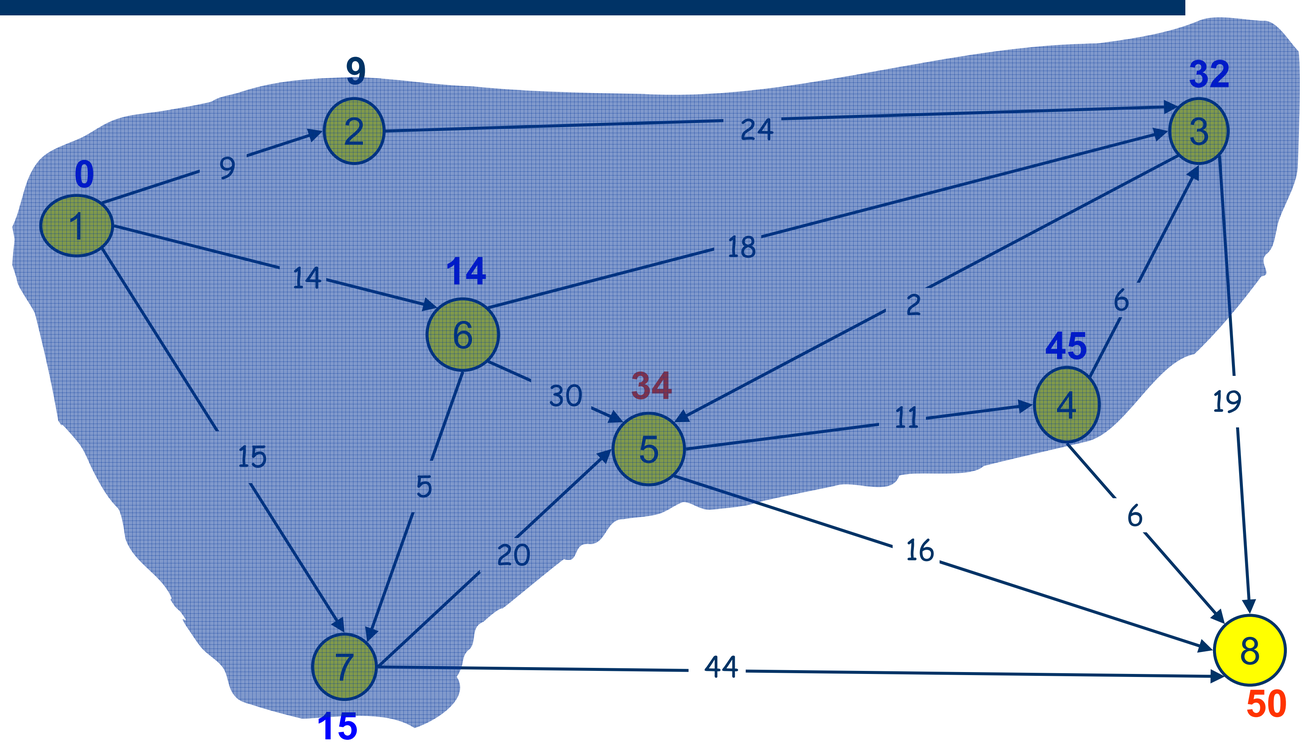
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 3 có độ dài 32, đi qua 6
Cập nhật giá trị d của các đỉnh lân cận với 3

Giải thuật Dijkstra



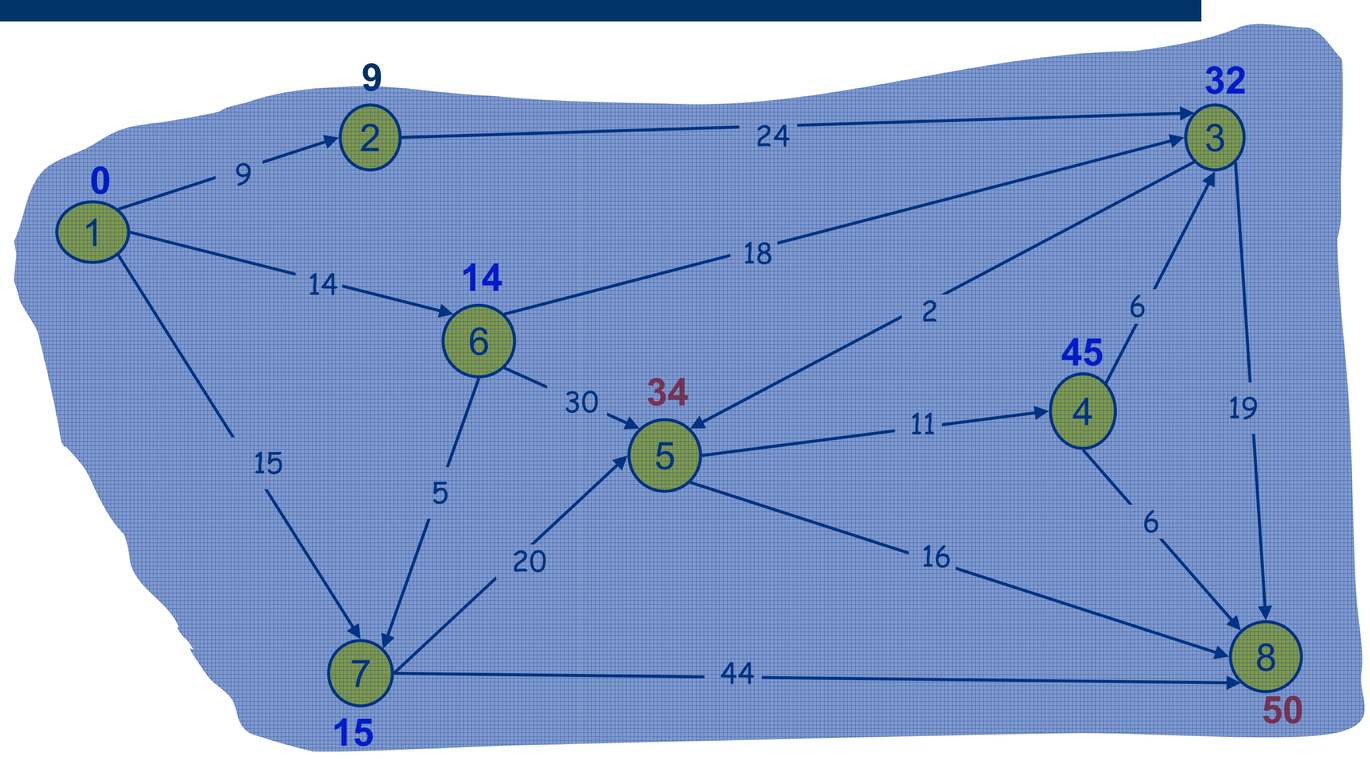
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 5 có độ dài 34, đi qua 6,3
Cập nhật giá trị d của các đỉnh lân cận với 5

Giải thuật Dijkstra



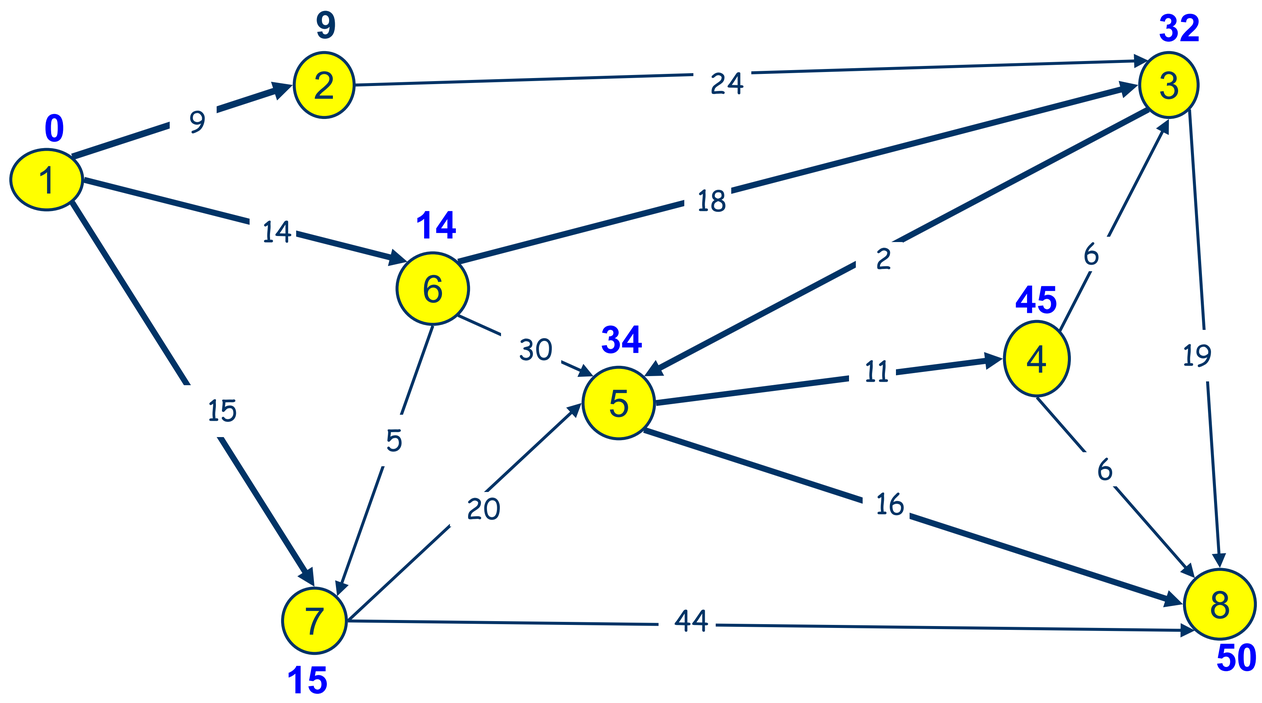
Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 4 có độ dài 45, đi qua 6,3,5
Cập nhật giá trị d của các đỉnh lân cận với 4

Giải thuật Dijkstra



Mở rộng cụm C, đường đi ngắn nhất từ 1 đến 8 có độ dài 50, đi qua 1,6,3,5

Giải thuật Dijkstra



Giải thuật Dijkstra

Algorithm Dijkstra(G, s)

{Sử dụng hai mảng trung gian D và P gồm n phần tử. Với n là số đỉnh trong đồ thị. D[i] chứa khoảng cách từ đỉnh s đến đỉnh i, P[i] chứa đỉnh ngay trước i trong đường đi ngắn nhất từ s đến i tại một thời điểm. Kết thúc giải thuật, thông tin về đường đi ngắn nhất từ đỉnh s đến các đỉnh khác nằm trong P, độ dài các đường đi nằm trong D}

1. {Khởi tạo D và P} **for each đỉnh v trong G do begin D[v] = ∞; P[v] = Null; end**.
2. **D[s] = 0; Q = V ;**
3. **While (Q ≠ rỗng) do begin**
    1. **Xác định đỉnh u trong Q mà D[u] có giá trị nhỏ nhất ; Q= Q – {u};**
    2. **Với lần lượt các đỉnh w là lân cận của u mà w còn nằm trong Q**
        1. **temp= D[u] + weight(u,w) ;**
        2. **If (temp < D[w]) then begin D[w] = temp; P[w] = u; end;**

end.

Bài toán bao đóng truyền ứng

- Mục tiêu:
  - Xác định xem có đường đi nào giữa các cặp đỉnh trong đồ thị G(V,E) cho trước hay không
- Hướng giải quyết:
  - Sử dụng ma trận lân cận
  - Xác định ma trận đường đi
- Giải thuật: Floyd-Washall

Bài toán bao đóng truyền ứng

- Ma trận đường đi của một đồ thị
  - Ma trận đường đi P có kích thước nxn, được xác định sử dụng công thức

$$P = A \vee A^{(2)} \vee A^{(3)} \vee ... \vee A^{(n)}$$

    - Nếu P_{ij} = 1 thì tồn tại một **đường đi từ đỉnh i đến đỉnh j**
    - Nếu P_{ij} = 0 thì không tồn tại bất kỳ một đường đi nào từ i đến j trong đồ thị G(V,E)
  - Ma trận đường đi P là ma trận lân cận của một đồ thị G' trong đó mỗi cung trong G' chỉ ra rằng có một mối quan hệ liên thông giữa 2 đỉnh.
  - G' gọi là bao đóng truyền ứng của G

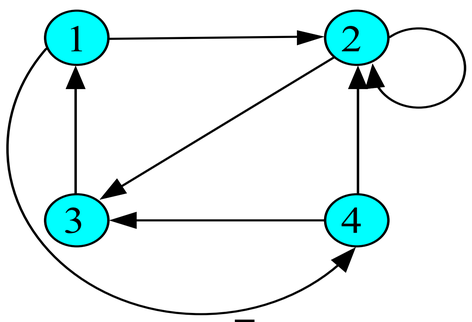
Bài toán bao đóng truyền ứng

- Giải thuật xác định ma trận đường đi của một đồ thị

```
Procedure FLOYD-WARSHALL(A,P,n)
1.  P:= A;
2.  for k:= 1 to n do
        for i:=1 to n do
            for j:=1 to n do
                P[i,j] := P[i,j] OR (P[i,k] AND P[k,j]);
3. return
```

Bài toán bao đóng truyền ứng

- Ví dụ: Cho đồ thị G và ma trận lân cận A



$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$A^{(2)} = A \wedge A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad A^{(3)} = A \wedge A^{(2)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Bài toán bao đóng truyền ứng

$$A^{(4)} = A \wedge A^{(3)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Ma trận đường đi P chỉ chứa các giá trị 1, chứng tỏ trong ma trận đã cho, giữa 2 đỉnh bất kỳ đều tồn tại đường đi

Bài toán sắp xếp Topo

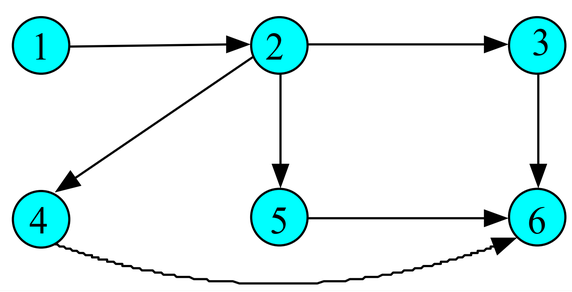
- Thứ tự bộ phận (Partial Order) là một quan hệ có 3 tính chất sau
  - Tính bắc cầu: x<y và y<z thì x<z
  - Tính không đối xứng: x<y thì không tồn tại y<x
  - Tính không phản xạ: không tồn tại x<x
- Một tập S có các phần tử mà giữa các phần tử có một thứ tự bộ phận thì S được gọi là Tập có thứ tự bộ phận

Bài toán sắp xếp Topo

- Sắp xếp tô pô là bài toán đặt ra trên một tập có thứ tự bộ phận
  - Mục đích: Sắp xếp các phần tử trong tập đã cho theo một thứ tự tuyến tính sao cho thứ tự bộ phận vẫn đảm bảo

Bài toán sắp xếp Topo

- Biểu diễn tập có thứ tự bộ phận bằng 1 đồ thị có hướng
  - Mỗi phần tử của tập S là một đỉnh
  - Nếu tồn tại một quan hệ i < j thì ta tạo một cung từ đỉnh i đến j trên đồ thị
  - Ví dụ:
    - Xây móng (1) < Xây tường (2) ;
    - Xây tường (2) < Đổ mái (3); Xây tường(2) < Lắp cửa(4)
    - Xây tường(2) < Làm điện(5); Lắp cửa(4) < Quét vôi(6)
    - Đổ mái (3) <Quét vôi(6) ; Làm điện(5) <Quét vôi(6)



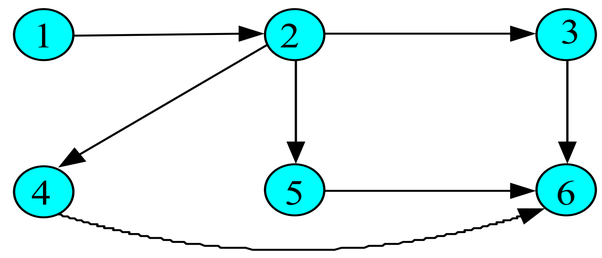
Bài toán sắp xếp Topo

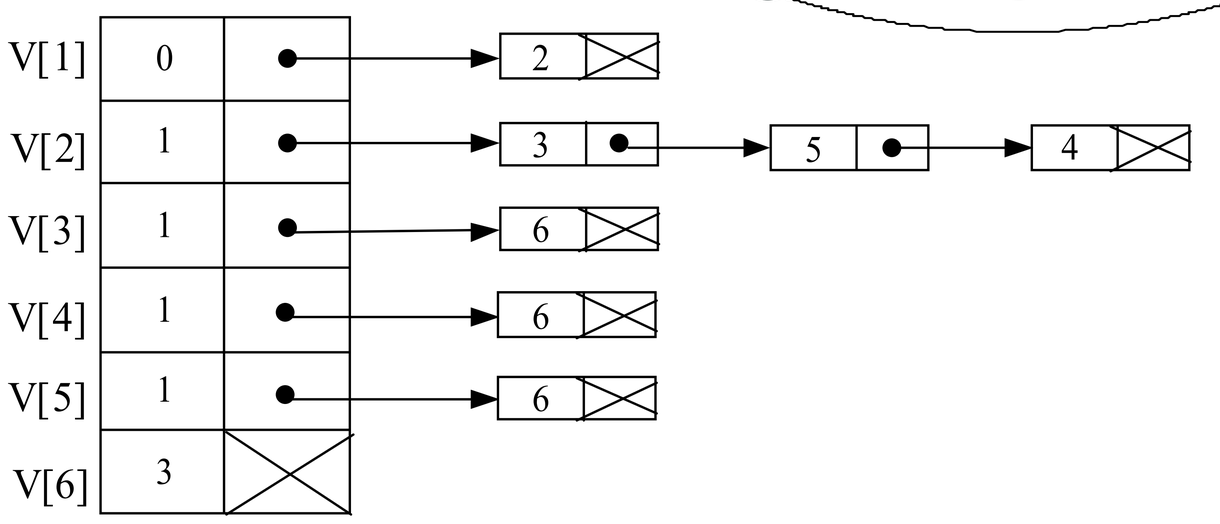
- Quy tắc chung
  - Chọn một đỉnh không có cung nào đi tới nó đưa đỉnh đó ra sắp xếp
  - Loại đỉnh vừa chọn ra khỏi đồ thị, xóa toàn bộ các cung xuất phát từ nó
  - Với đồ thị còn lại, lặp lại 2 bước trên
  - Công việc kết thúc khi tất cả các đỉnh được đưa ra sắp xếp

Bài toán sắp xếp Topo

- Để cài đặt việc sắp xếp trên, ta cần biết
  - Số các cung đi đến một đỉnh. Đỉnh được chọn có giá trị này là 0
  - Các đỉnh lân cận của một đỉnh
  - Lưu trữ đồ thị bằng danh sách lân cận với một bổ sung sau
    - Nút đầu danh sách lân cận của một đỉnh lưu trữ trong một vector và mỗi nút có 2 trường
    - Trường LINK
    - Trường COUNT: lưu trữ số cung đi tới đỉnh đó

Bài toán sắp xếp Topo





Bài toán sắp xếp Topo

```
Procedure TOPO-ORDER(V,n)
1.  for i:=1 to n do if COUNT(V[i]) =  0 then nạp i vào trong Q;
2. Repeat
     Đưa đỉnh j ở lối trước của Q ra;
     ptr := LINK(V[j]); { TÌm đến nút đầu trong danh sách lân cận của j}
     while ptr khác NULL do begin
           k:= VERTEX(ptr) ; {k là đỉnh lân cận của j}
           COUNT(V[k]) := COUNT(V[k]) -1;
           if COUNT(V[k]) = 0 then nạp k vào trong Q;
           ptr := LINK(ptr); {Đi đến nút tiếp trong danh sách lân cận của j }
     end
until Q rỗng
3. return
```