

Bài 14: Đồ thị (2/2)

Khoa Công nghệ Thông tin – Đại học Công Nghệ



Nội dung chính

1. Đồ thị và các khái niệm liên quan
2. Cài đặt đồ thị
3. Một số bài toán tiêu biểu

- Đi qua/duyệt đồ thị

- BFS, DFS

- Sắp xếp topo trên đồ thị định hướng không có chu trình

- Tìm đường đi ngắn nhất

- Từ một đỉnh nguồn

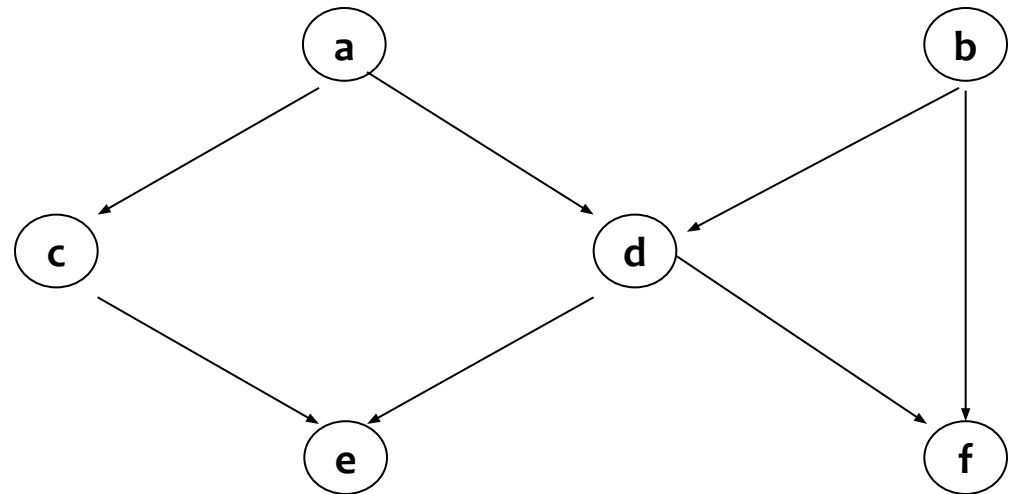
- Giữa mọi cặp đỉnh

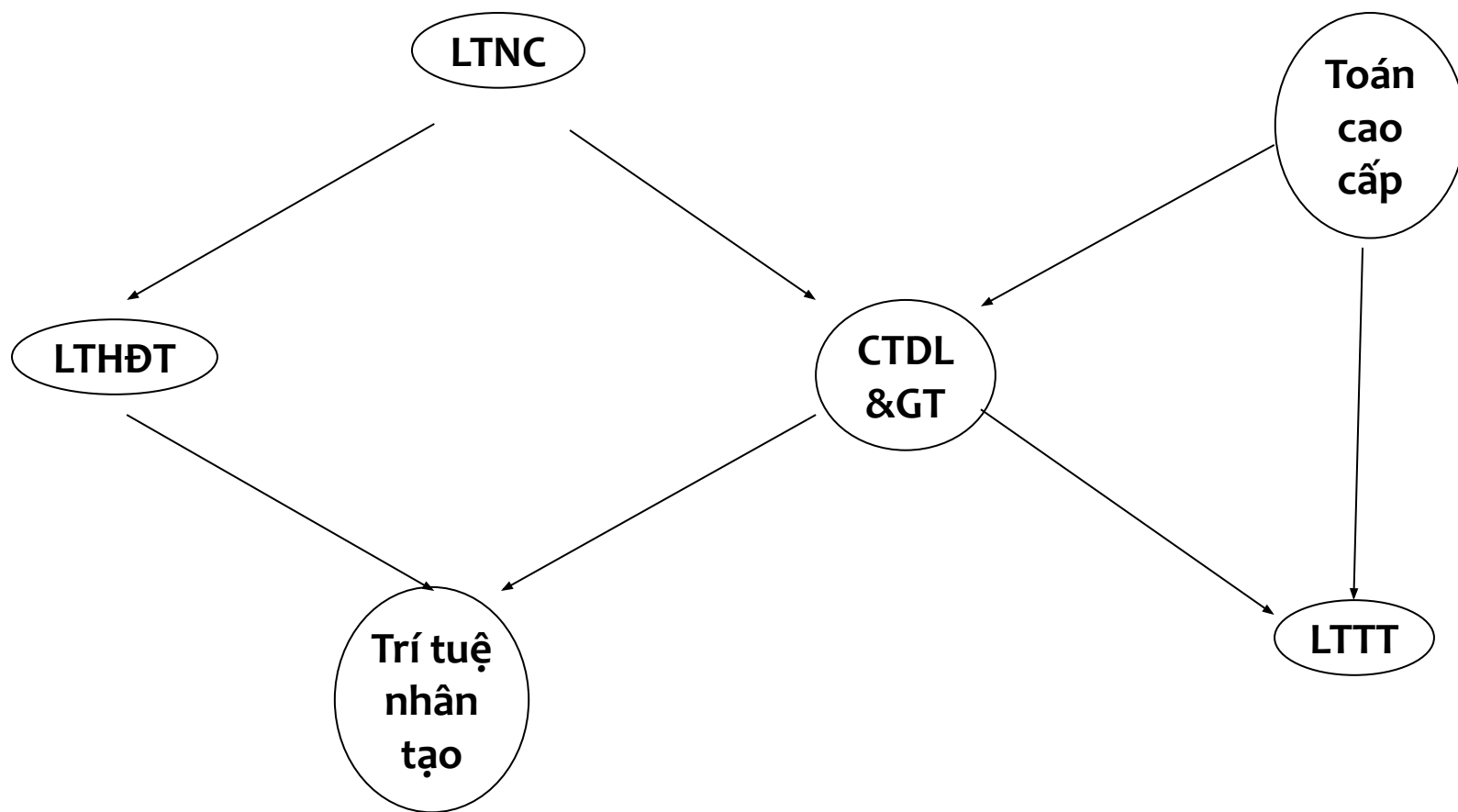
3.1. Đi qua đồ thị

3.2. Sắp xếp topo

Đồ thị định hướng không chu trình

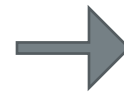
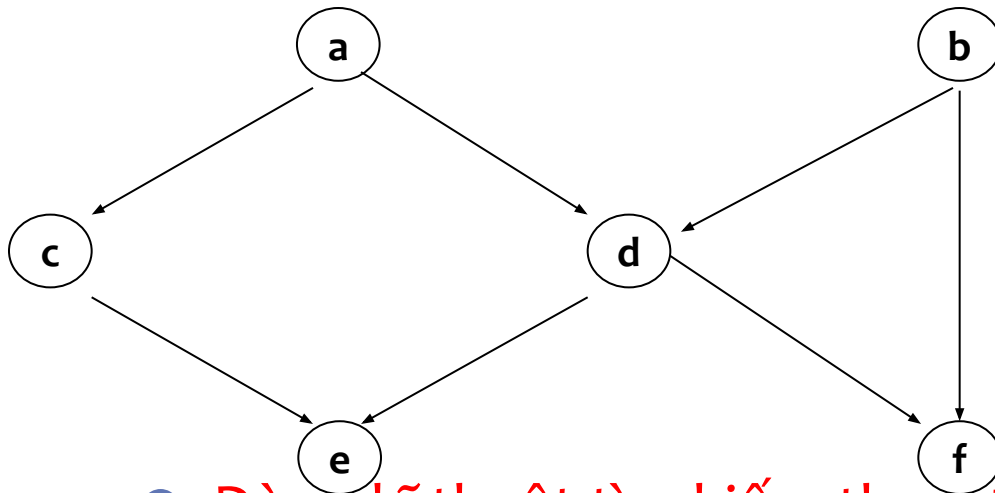
- Thuật ngữ
 - directed acyclic graph (**DAG**)
 - acyclic digraph
- Nhiều dạng quan hệ trên một tập đối tượng có thể biểu diễn bởi DAG. Ví dụ:
 - Quan hệ thứ tự bộ phận trên một tập A
 - Quan hệ thứ tự thời gian giữa các nhiệm vụ trong một đề án
 - Quan hệ thứ tự thời gian giữa các môn học trong một chương trình học





Sắp xếp topo (topological sort)

- Cho $G = (V, E)$ là một DAG, ta cần sắp xếp các đỉnh của đồ thị thành một danh sách
 - sao cho nếu có cung (u, v) thì u cần phải đứng trước v trong danh sách đó.



(a, c, b, d, e, f)
hoặc
(a, b, d, c, e, f)
...

- Dùng kĩ thuật tìm kiếm theo độ sâu?

Ý tưởng toposort dựa trên DFS

Algorithm *DFS(v)*

// Tìm kiếm theo độ sâu xuất phát từ v.

Input: Đỉnh v chưa được thăm

for (mỗi đỉnh u kề v)

if (u chưa được thăm)

Đánh dấu u đã được thăm;

DFS(u)

Algorithm *DFSTraversal(G)*

// Đi qua đồ thị $G=(V, E)$ theo độ sâu

for (mỗi $v \in V$) Đánh dấu v chưa được thăm;

for (mỗi $v \in V$)

if (v chưa được thăm)

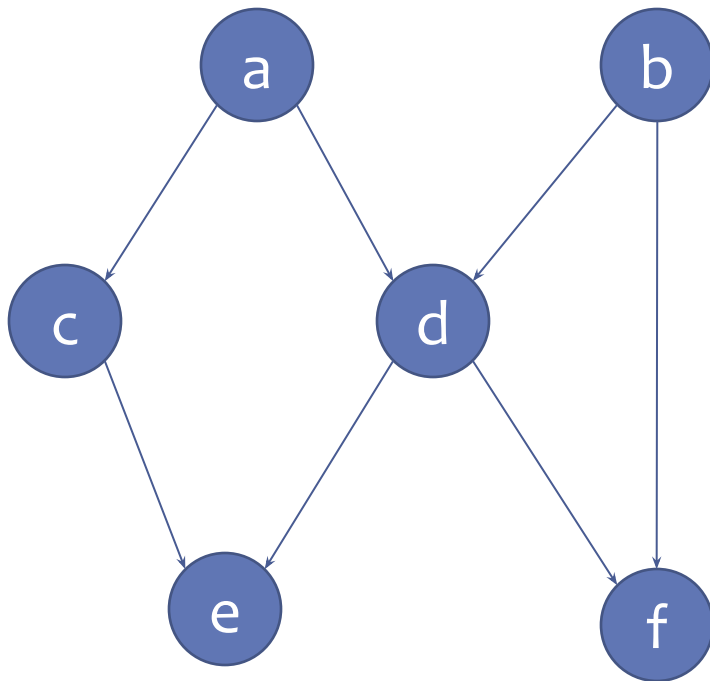
Thăm v và đánh dấu v đã được thăm;

DFS(v);

- Thực hiện DFSTraversal trên đồ thị G, thêm lệnh **L.append(v)** vào cuối hàm DFS(v)
- Đảo ngược L

[Tác giả: Tarjan]

Minh họa TopoSort(G)



DFS(a)

DFS(c)

DFS(e)

$L = (e)$

$L = (e, c)$

DFS(d)

DFS(f)

$L = (e, c, f)$

$L = (e, c, f, d)$

$L = (e, c, f, d, a)$

DFS(b)

$L = (e, c, f, d, a, b)$

$L = (b, a, d, f, c, e)$

3.3. Tìm đường đi ngắn nhất

Tổng quan

- Tìm đường đi ngắn nhất trong đồ thị

- Không trọng số: Dùng BFS

- Có trọng số

- Trọng số có thể âm: Không xét

- Bellman-Ford

- Trọng số không âm

- độ dài cung (u, v) là $c(u, v)$

- không có cung từ u tới v thì $c(u, v) = +\infty$

- Xét hai vấn đề

- Tìm đường đi ngắn nhất từ một đỉnh nguồn tới các đỉnh còn lại.

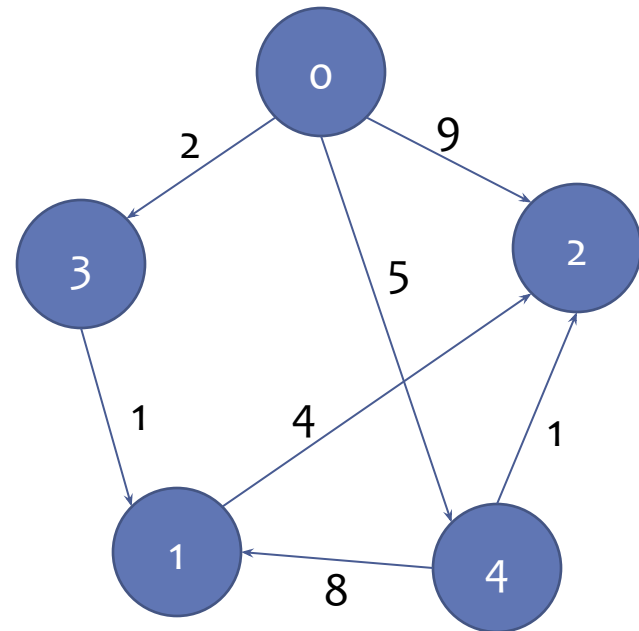
- single-source shortest path problem

- Tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị.

- all-pairs shortest path problem

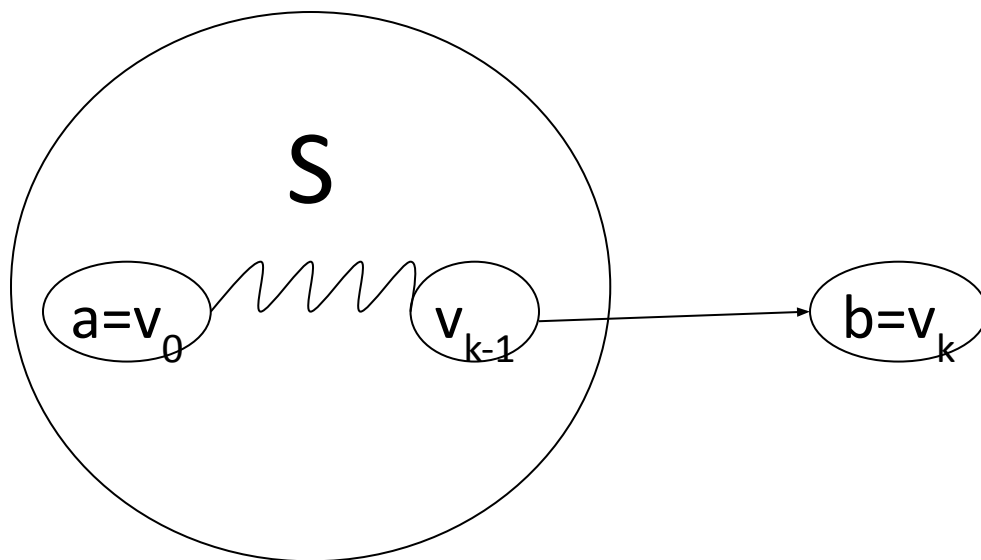
Thuật toán Dijkstra cho bài single-source

- Ví dụ: tìm đường đi ngắn nhất từ đỉnh nguồn là đỉnh 0
- Thiết kế dựa vào kỹ thuật **tham ăn**
- Xác định đường đi ngắn nhất từ **đỉnh nguồn a** tới các đỉnh còn lại qua các bước
 - Mỗi bước ta xác định đường đi ngắn nhất từ a tới một đỉnh
 - Lưu các đỉnh đã xác định đường đi ngắn nhất từ a tới chúng vào tập S
 - Ban đầu tập S chỉ chứa một đỉnh nguồn a



Thuật toán Dijkstra ...

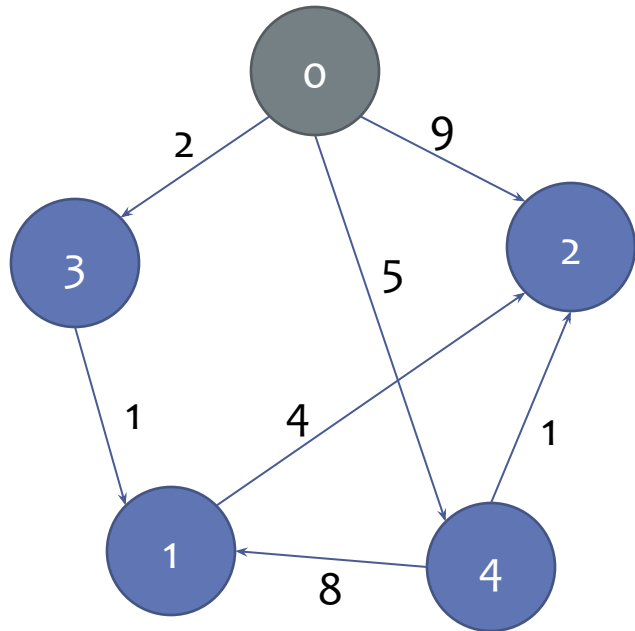
- Gọi đường đi từ a tới đỉnh b là **đường đi đặc biệt** nếu đường đi đó chỉ **đi qua** các đỉnh trong S
- Dùng mảng D : Độ dài đường đi đặc biệt từ a tới b lưu trong $D[b]$
 - Ban đầu $S = \{a\}$, $D[a] = 0$, $D[b] = c(a, b)$ với $b \neq a$



Thuật toán Dijkstra ...

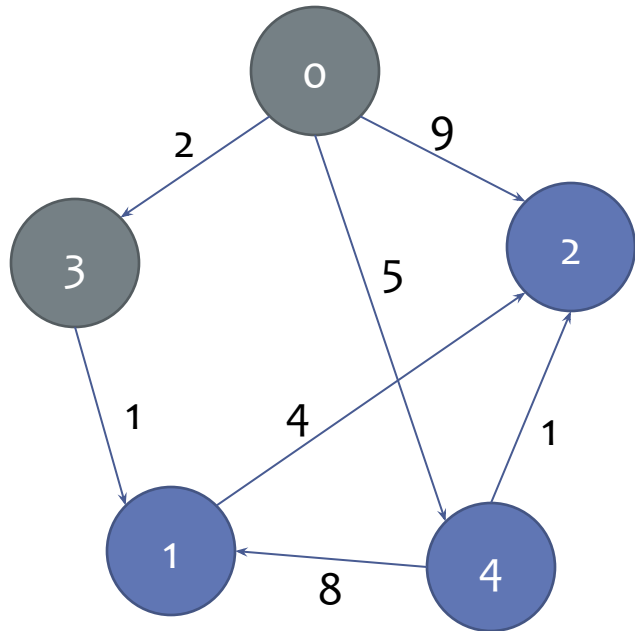
- Dùng mảng D: Độ dài đường đi đặc biệt từ a tới b lưu trong $D[b]$
 - Ban đầu $S = \{a\}$, $D[a] = 0$, $D[b] = c(a, b)$ với $b \neq a$
 - Tại mỗi bước
 - Chọn một đỉnh u không thuộc S mà $D[u]$ nhỏ nhất và thêm u vào S
 - xem $D[u]$ là độ dài đường đi ngắn nhất từ a tới u
 - Sau đó, xác định lại các $D[b]$ với b ở ngoài S
$$D[b] = \min(D[b], D[u] + c(u, b))$$
 - Lặp lại cho tới khi S gồm tất cả các đỉnh của đồ thị

Minh họa thuật toán Dijkstra: Ban đầu



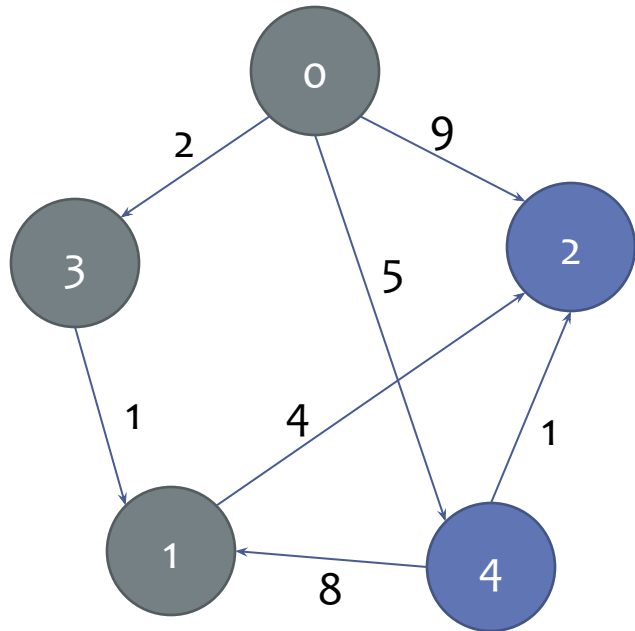
- $S = \{0\}$
- $D[0] = 0$
- $D[1] = \infty$
- $D[2] = 9$
- $D[3] = 2$
- $D[4] = 5$

Minh họa thuật toán Dijkstra: Thêm 3 vào S



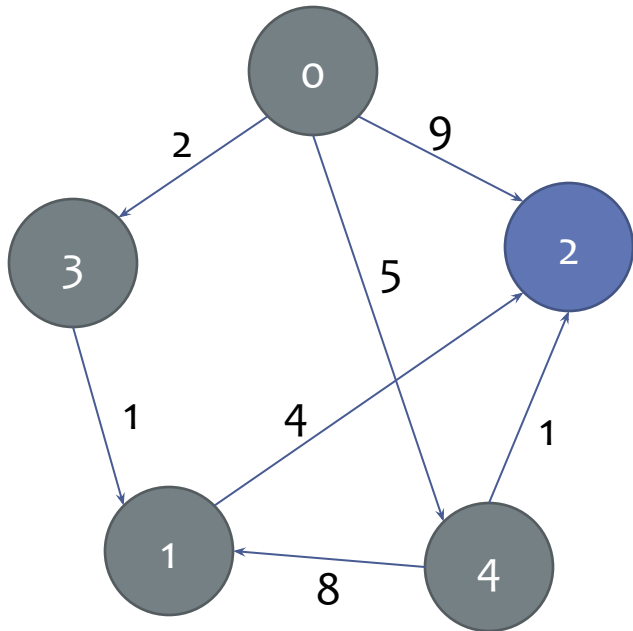
- $S = \{0, 3\}$
- $D[0] = 0$
- $D[1] = \min(\infty, D[3] + 1) = 3$
- $D[2] = \min(9, D[3] + \infty) = 9$
- $D[3] = 2$
- $D[4] = \min(5, D[3] + \infty) = 5$

Minh họa thuật toán Dijkstra: Thêm 1 vào S



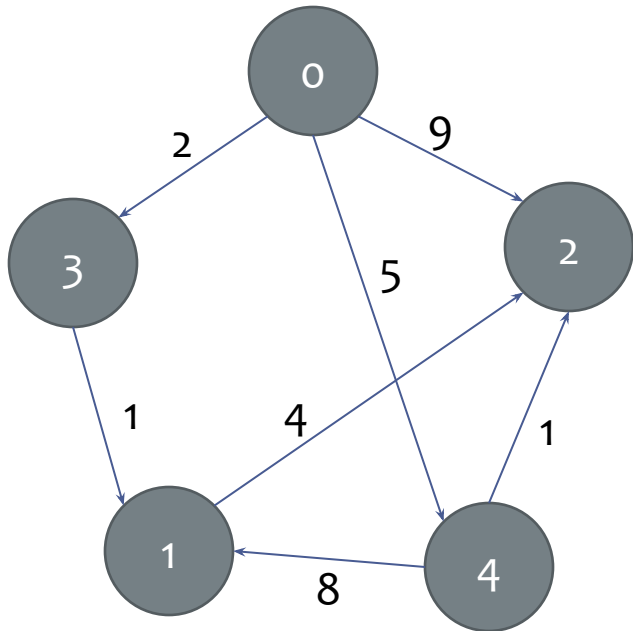
- $S = \{0, 3, 1\}$
- $D[0] = 0$
- $D[1] = 3$
- $D[2] = \min(9, D[1] + 4) = 7$
- $D[3] = 2$
- $D[4] = \min(5, D[1] + \infty) = 5$

Minh họa thuật toán Dijkstra: Thêm 4 vào S



- $S = \{0, 3, 1, 4\}$
- $D[0] = 0$
- $D[1] = 3$
- $D[2] = \min(7, D[4] + 1) = 6$
- $D[3] = 2$
- $D[4] = 5$

Minh họa thuật toán Dijkstra: Thêm 2 vào S



- $S = \{0, 3, 1, 4, 2\}$
- $D[0] = 0$
- $D[1] = 3$
- $D[2] = 6$
- $D[3] = 2$
- $D[4] = 5$
- $D[b]$ lưu độ dài đường đi ngắn nhất từ $a=0$ tới b , với mọi $b \in V$

Các vấn đề khác

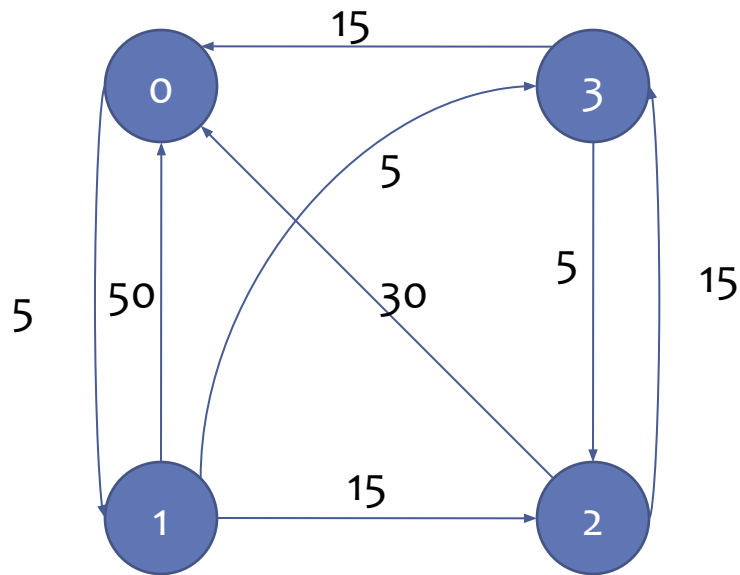
- Ghi lại vết đường đi ngắn nhất từ nguồn tới các đỉnh khác
- Tính đúng đắn của thuật toán Dijkstra
- Dùng hàng ưu tiên lưu tập đỉnh ngoài S để tăng hiệu quả
 - $O(|V|\log|V| + |E|\log|V|)$

Thuật toán Floyd cho bài all-pairs

- Thiết kế dựa trên kỹ thuật quy hoạch động
- Ký hiệu S_k là tập các đỉnh từ 0 đến k
 - $S_k = \{0, 1, \dots, k\}$, $k \leq n-1$
- Gọi $A_k(i, j)$ là độ dài đường đi ngắn nhất từ đỉnh i tới đỉnh j nhưng chỉ đi qua các đỉnh trong tập S_k
 - Khi $k = n-1$ thì $S_{n-1} = V$ \square $A_{n-1}(i, j)$ chính là đường đi ngắn nhất từ i tới j trong đồ thị đã cho
 - Khi $k = -1$, S_k rỗng \square $A_{-1}(i, j) = c(i, j)$

Minh họa: $k = -1$

S_{-1} rỗng, $A_{-1}(i,j)$ cho trong bảng



	0	1	2	3
0	0	5	∞	∞
1	50	0	15	5
2	30	∞	0	15
3	15	∞	5	0

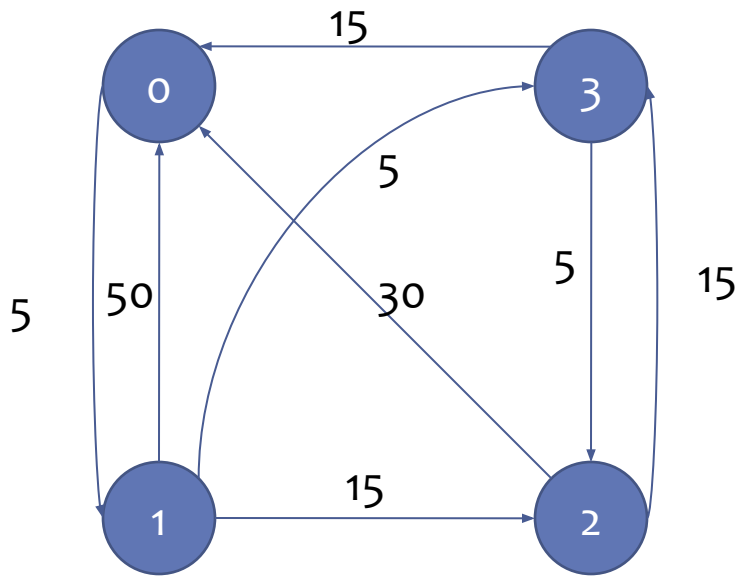
Công thức tính A_k từ A_{k-1}

- Nhận xét quan trọng
 - Nếu đỉnh k nằm trên đường đi ngắn nhất từ đỉnh i tới đỉnh j thì đoạn đường từ i tới k và đoạn đường từ k tới j phải là đường đi ngắn nhất từ i tới k và từ k tới j tương ứng
 - Nếu $A_k(i,j)$ là độ dài đường đi không qua đỉnh k , tức là đường đi này chỉ đi qua các đỉnh trong S_{k-1} thì
$$A_k(i,j) = A_{k-1}(i,j)$$
 - Nếu $A_k(i,j)$ là độ dài của đường đi qua đỉnh k thì trên đường đi này đoạn từ i tới k có độ dài là $A_{k-1}(i,k)$, còn đoạn đường từ k tới j có độ dài là $A_{k-1}(k,j)$

- Do đó

$$A_k(i,j) = \min(A_{k-1}(i,j) , A_{k-1}(i,k) + A_{k-1}(k,j))$$

Minh họa: $k=0\dots?$



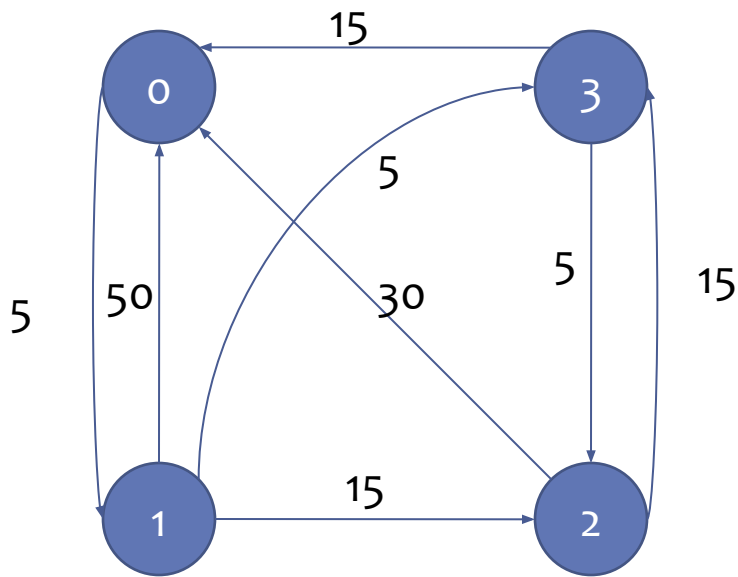
A_{-1}

	0	1	2	3
0	0	5	∞	∞
1	50	0	15	5
2	30	∞	0	15
3	15	∞	5	0

A_0

	0	1	2	3
0	0	5	∞	∞
1	50	0	15	5
2	30	35	0	15
3	15	20	5	0

Minh họa: $k=0\dots?$



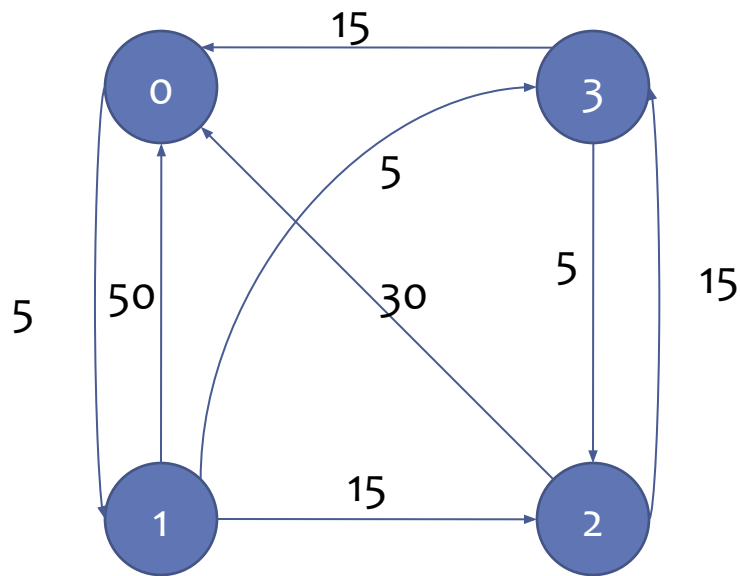
A_0

	0	1	2	3
0	0	5	∞	∞
1	50	0	15	5
2	30	35	0	15
3	15	20	5	0

A_1

	0	1	2	3
0	0	5	20	10
1	50	0	15	5
2	30	35	0	15
3	15	20	5	0

Minh họa: $k=0\dots?$



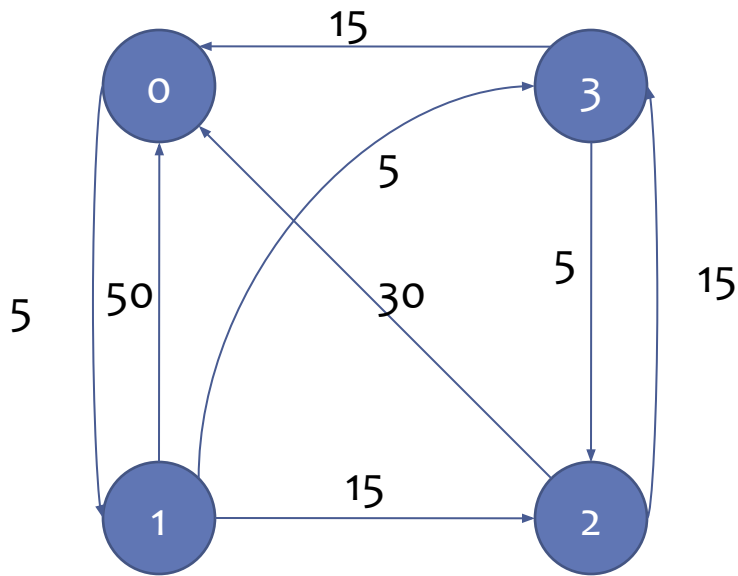
A_1

	0	1	2	3
0	0	5	20	10
1	50	0	15	5
2	30	35	0	15
3	15	20	5	0

A_2

	0	1	2	3
0	0	5	20	10
1	45	0	15	5
2	30	35	0	15
3	15	20	5	0

Minh họa: $k=0\dots?$



A_2

	0	1	2	3
0	0	5	20	10
1	45	0	15	5
2	30	35	0	15
3	15	20	5	0

A_3

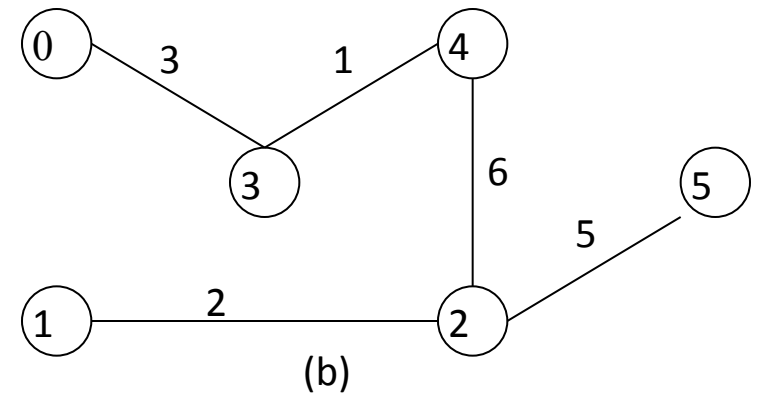
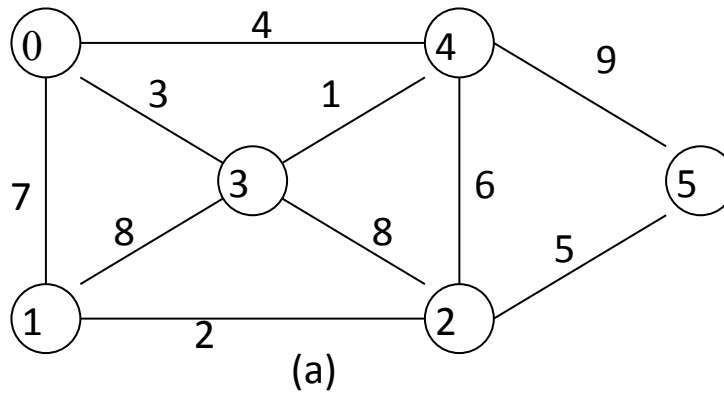
	0	1	2	3
0	0	5	15	10
1	20	0	10	5
2	30	35	0	15
3	15	20	5	0

3.4. Tìm cây bao trùm ngắn nhất

Bài toán

- $G = (V, E)$ là đồ thị vô hướng liên thông
- $G' = (V, T)$ có $T \subseteq E$, liên thông và không có chu trình được gọi là **cây bao trùm** của G
 - Cây này có $|V| - 1$ cạnh
- Ta cần tìm cây bao trùm ngắn nhất của một đồ thị G vô hướng liên thông có trọng số không âm
 - tức là cây bao trùm có tổng độ dài các cạnh là nhỏ nhất
 - Thuật ngữ: minimum spanning tree (MST)

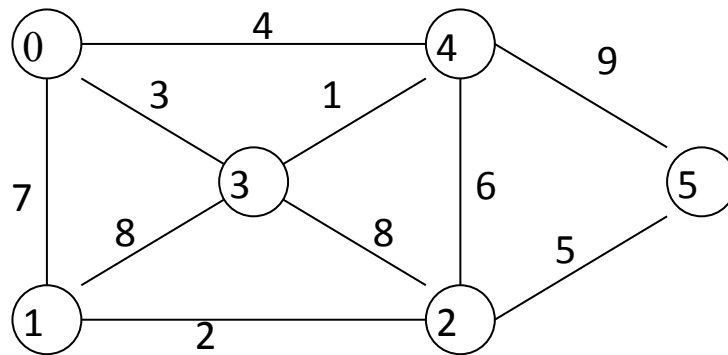
Minh họa một cây bao trùm ngắn nhất



Ý tưởng

- Thiết kế theo kỹ thuật tham ăn
- Xây dựng tập T các cạnh dần từng bước xuất phát từ T rỗng
- Trong mỗi bước lặp, ta sẽ chọn cạnh (u,v) ngắn nhất trong các cạnh còn lại để đưa vào tập T
 - Prim: $T \cup (u, v)$ phải liên thông, không có chu trình
 - Kruskal: $T \cup (u, v)$ không có chu trình
 - Sử dụng KDLTT họ các tập con không cắt nhau (disjoint set ADT) [chương 13]

Minh họa



Các vấn đề khác

- Độ phức tạp thời gian
- Tính đúng đắn

Tóm tắt

3.2. Sắp xếp topo trên DAG: Thuật toán của Tarjan

3.3. Tìm đường đi ngắn nhất

- Single-source: Thuật toán tham ăn Dijkstra
- All-pairs: Thuật toán quy hoạch động Floyd

3.4. Tìm cây bao trùm ngắn nhất

- Thuật toán tham ăn Prim
- Thuật toán tham ăn Kruskal