Đồ thị nâng cao

THUẬT TOÁN ỨNG DỤNG

Đỗ Phan Thuận
thuandp.sinhvien@gmail.com

Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 12 tháng 5 năm 2020

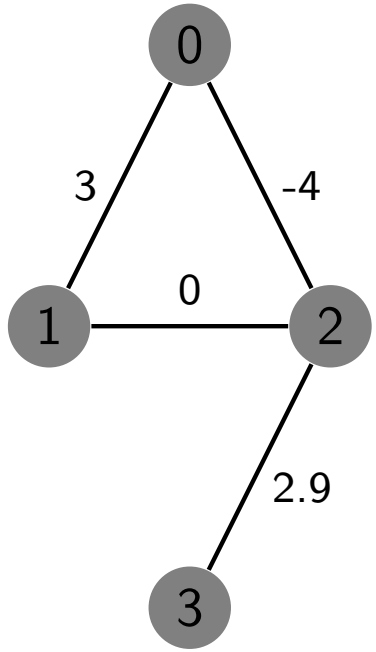
Đồ Thị Nâng Cao

Đồ thị có trọng số

- Trong phần này ta xét đồ thị mà mỗi cạnh của nó có trọng số đi kèm
  - khoảng cách của con đường được tượng trưng bởi cạnh đồ thị
  - giá trị trên cạnh
  - trọng số trên cạnh
- biểu diễn đồ thị có trọng số bởi danh sách kề sửa đổi

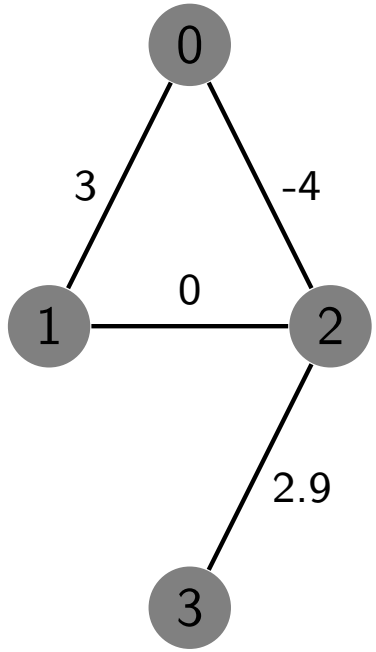
Đồ thị có trọng số

```
struct edge {
    int u, v;
    int weight;

    edge(int _u, int _v, int _w) {
        u = _u;
        v = _v;
        weight = _w;
    }
};
```



Đồ thị có trọng số

```
vector<edge> adj[4];

adj[0].push_back(edge(0, 1, 3));
adj[0].push_back(edge(0, 2, -4));

adj[1].push_back(edge(1, 0, 3));
adj[1].push_back(edge(1, 2, 0));

adj[2].push_back(edge(2, 0, -4));
adj[2].push_back(edge(2, 1, 0));
adj[2].push_back(edge(2, 3, 2.9));

adj[3].push_back(edge(3, 2, 2.9));
```



Đồ Thị Nâng Cao

Union-Find

- Cho n phần tử
- Cần quản lý vào các tập không giao nhau
- Mỗi phần tử ở trong đúng 1 tập

- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$
- $collections = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$

- Hai toán tử hiệu quả: `find(x)` và `union(x,y)`.

Union-Find

- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$

- `find(x)` trả về phần tử đại diện của tập chứa x
  - `find(1) = 1`
  - `find(4) = 1`
  - `find(3) = 5`
  - `find(5) = 5`
  - `find(6) = 5`
  - `find(2) = 2`

- a và b thuộc cùng một tập khi và chỉ khi `find(a) == find(b)`

Union-Find

- $items = \{1, 2, 3, 4, 5, 6\}$
- $collections = \{1, 4\}, \{3, 5, 6\}, \{2\}$

- `union(x, y)` trộn tập chứa x và tập chứa y vào nhau
  - `union(4, 2)`
  - $collections = \{1, 2, 4\}, \{3, 5, 6\}$
  - `union(3, 6)`
  - $collections = \{1, 2, 4\}, \{3, 5, 6\}$
  - `union(2, 6)`
  - $collections = \{1, 2, 3, 4, 5, 6\}$

Cài đặt Union-Find

- Hợp nhất nhanh với kỹ thuật nén đường (Quick Union with path compression)
- Cực kỳ dễ cài đặt
- Cực kỳ hiệu quả

```
struct union_find {
    vector<int> parent;
    union_find(int n) {
        parent = vector<int>(n);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    // find and union
};
```

Cài đặt Union-Find

```
// find and union

int find(int x) {
    if (parent[x] == x) {
        return x;
    } else {
        parent[x] = find(parent[x]); //nen parent
        return parent[x];
    }
}

void unite(int x, int y) {
    parent[find(x)] = find(y);
}
```

Cài đặt Union-Find phiên bản rút gọn

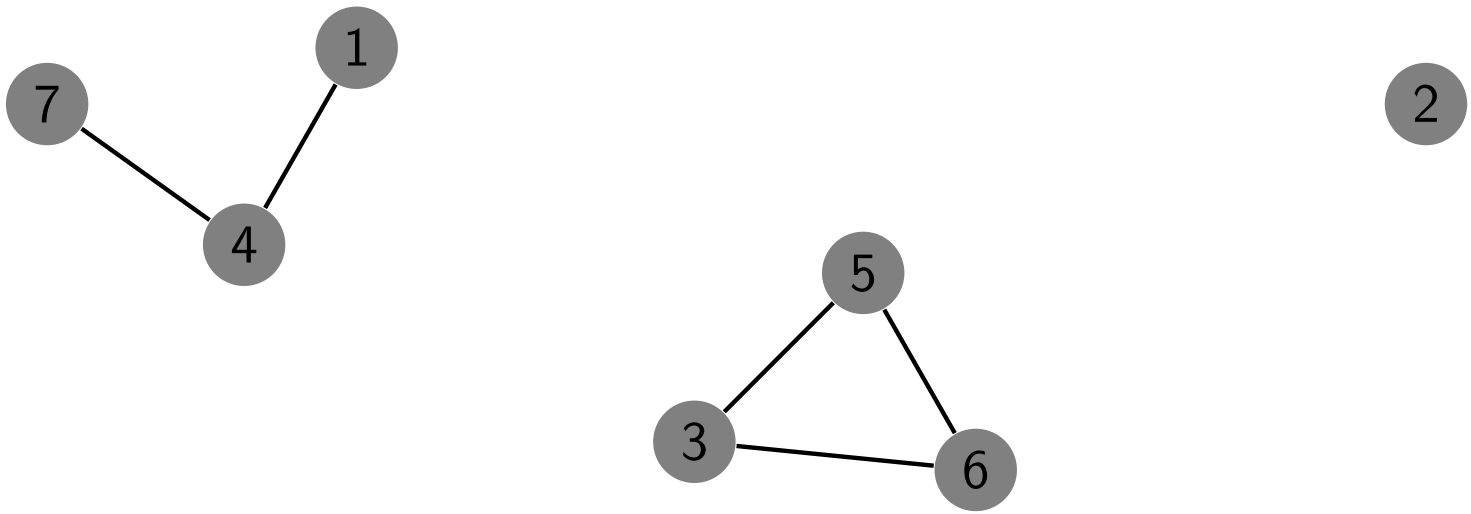
- Nếu vội...

```
#define MAXN 1000
int p[MAXN];

int find(int x) {
    return p[x] == x ? x : p[x] = find(p[x]); }
void unite(int x, int y) { p[find(x)] = find(y); }

for (int i = 0; i < MAXN; i++) p[i] = i;
```

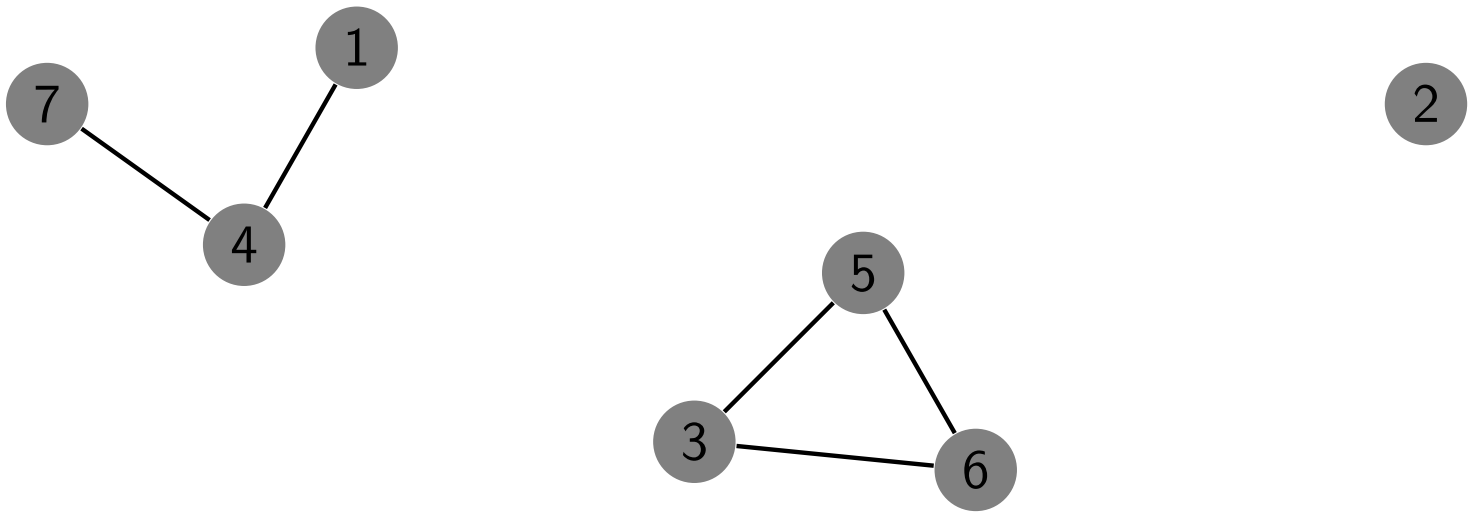
Ứng dụng của Union-Find

- Union-Find quản lý các tập không giao nhau
- Xử lý từng loại các tập không giao nhau tùy thời điểm
- Các bài toán ứng dụng quen thuộc thường trên đồ thị

Union-find trên đồ thị

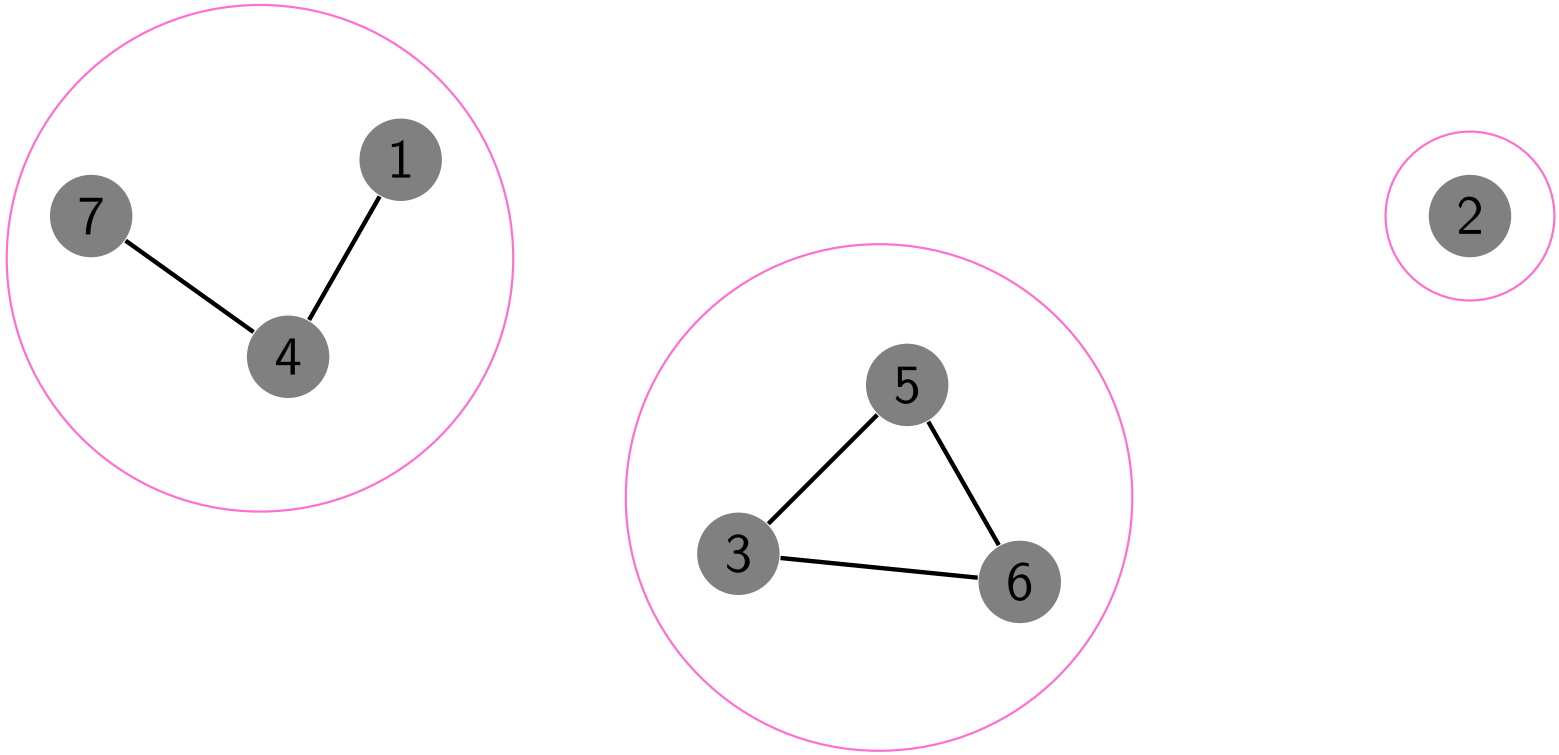


Union-find trên đồ thị



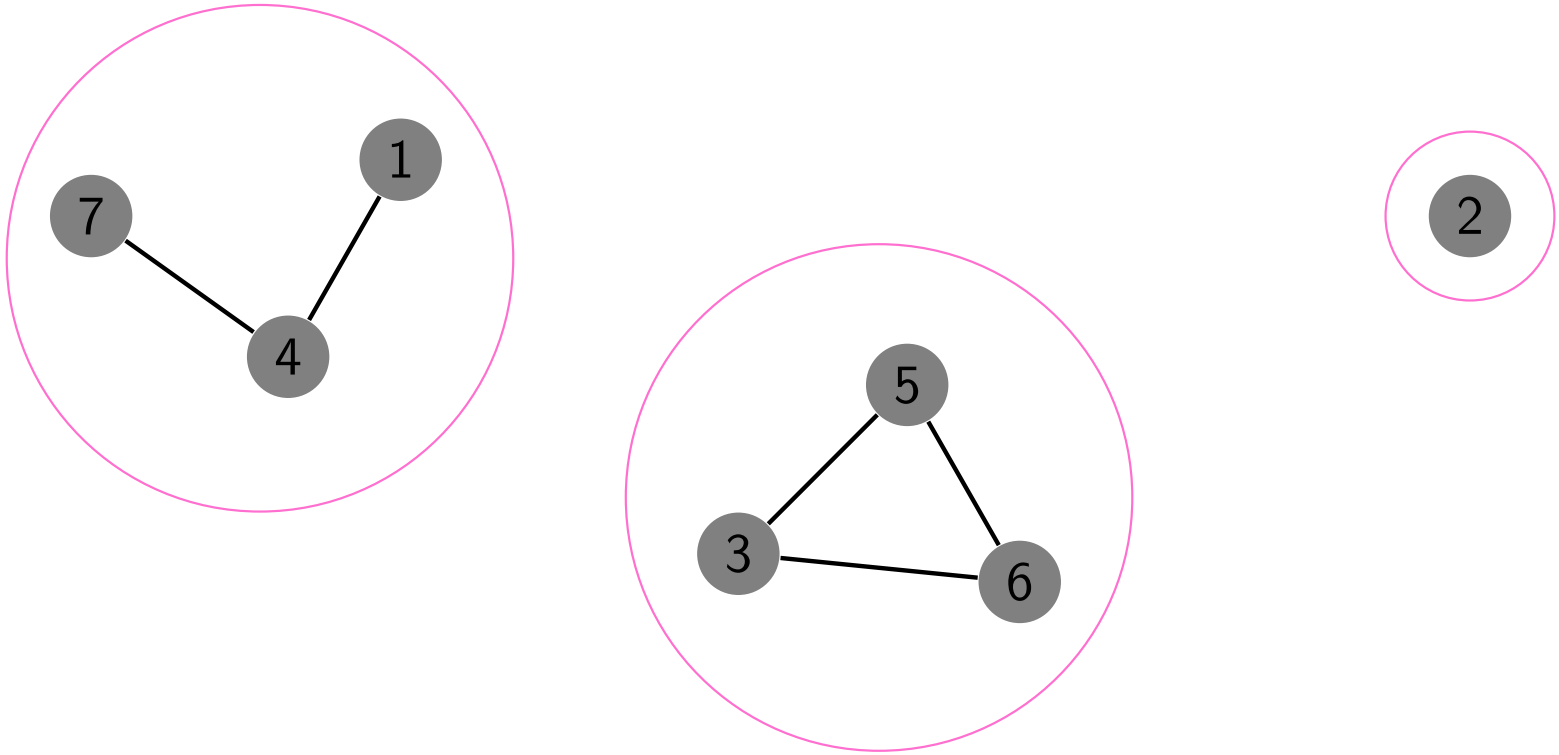
- $items = \{1, 2, 3, 4, 5, 6, 7\}$

Union-find trên đồ thị



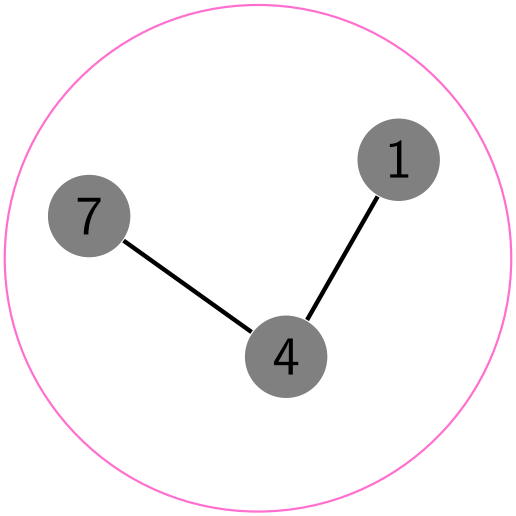
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$

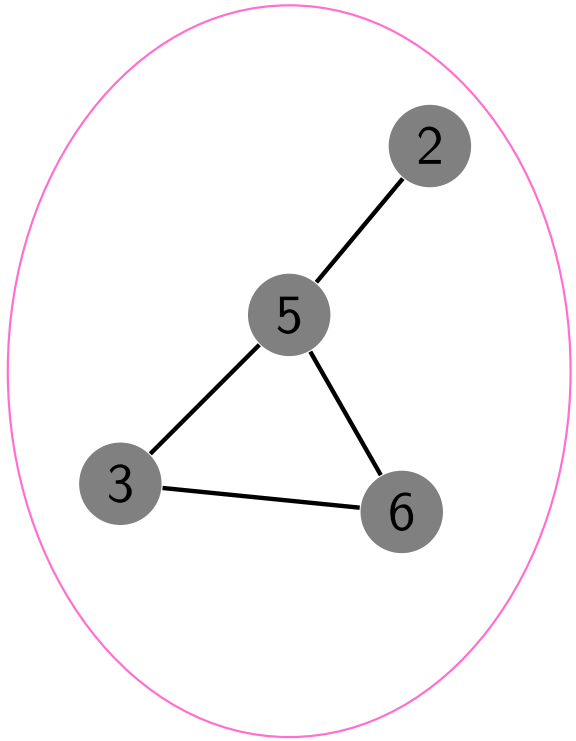
Union-find trên đồ thị



- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2\}, \{3, 5, 6\}$
- `union(2, 5)`

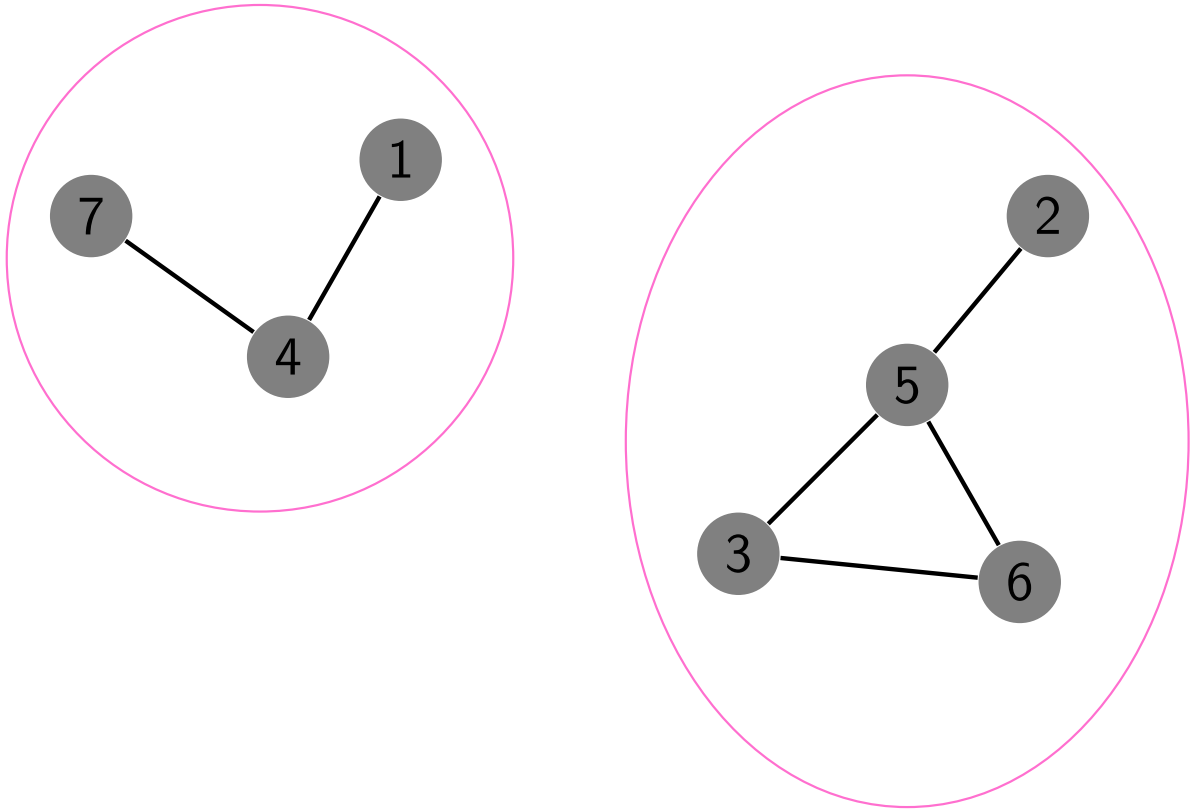
Union-find trên đồ thị





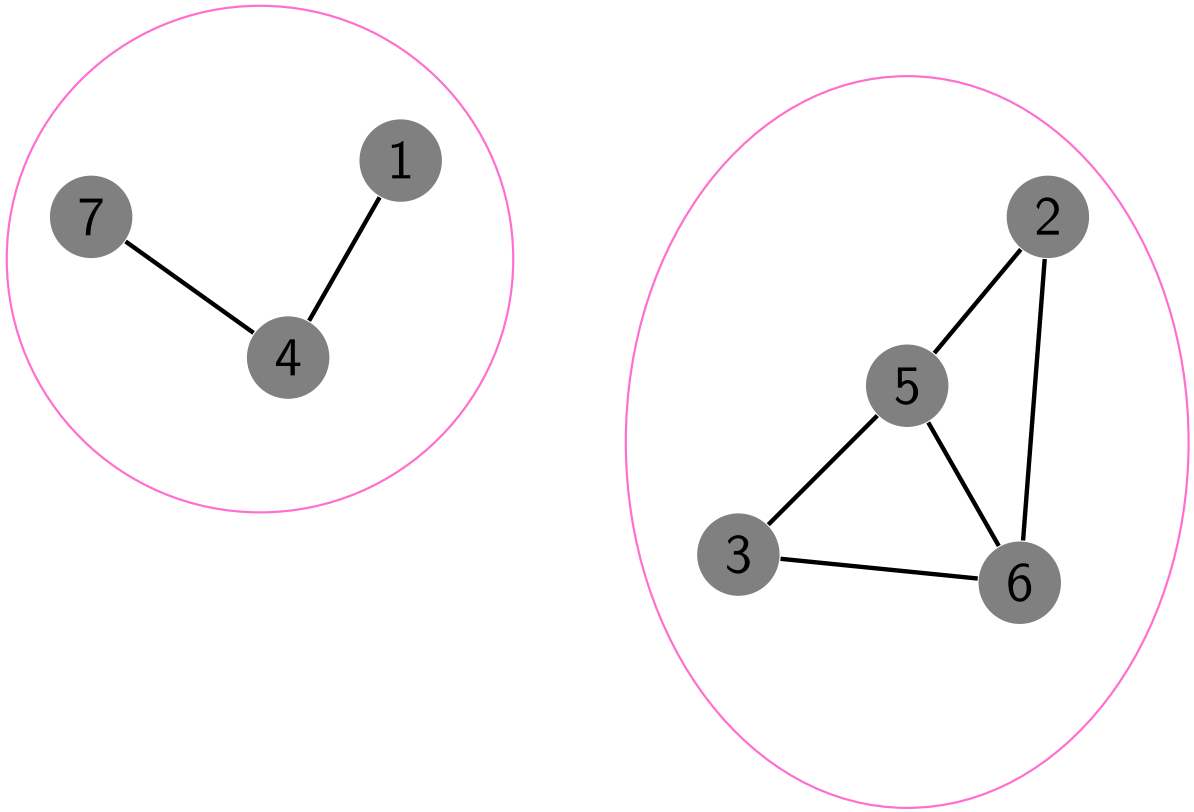
- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

Union-find trên đồ thị



- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$
- `union(6, 2)`

Union-find trên đồ thị



- $items = \{1, 2, 3, 4, 5, 6, 7\}$
- $collections = \{1, 4, 7\}, \{2, 3, 5, 6\}$

Bài toán ví dụ: Friends

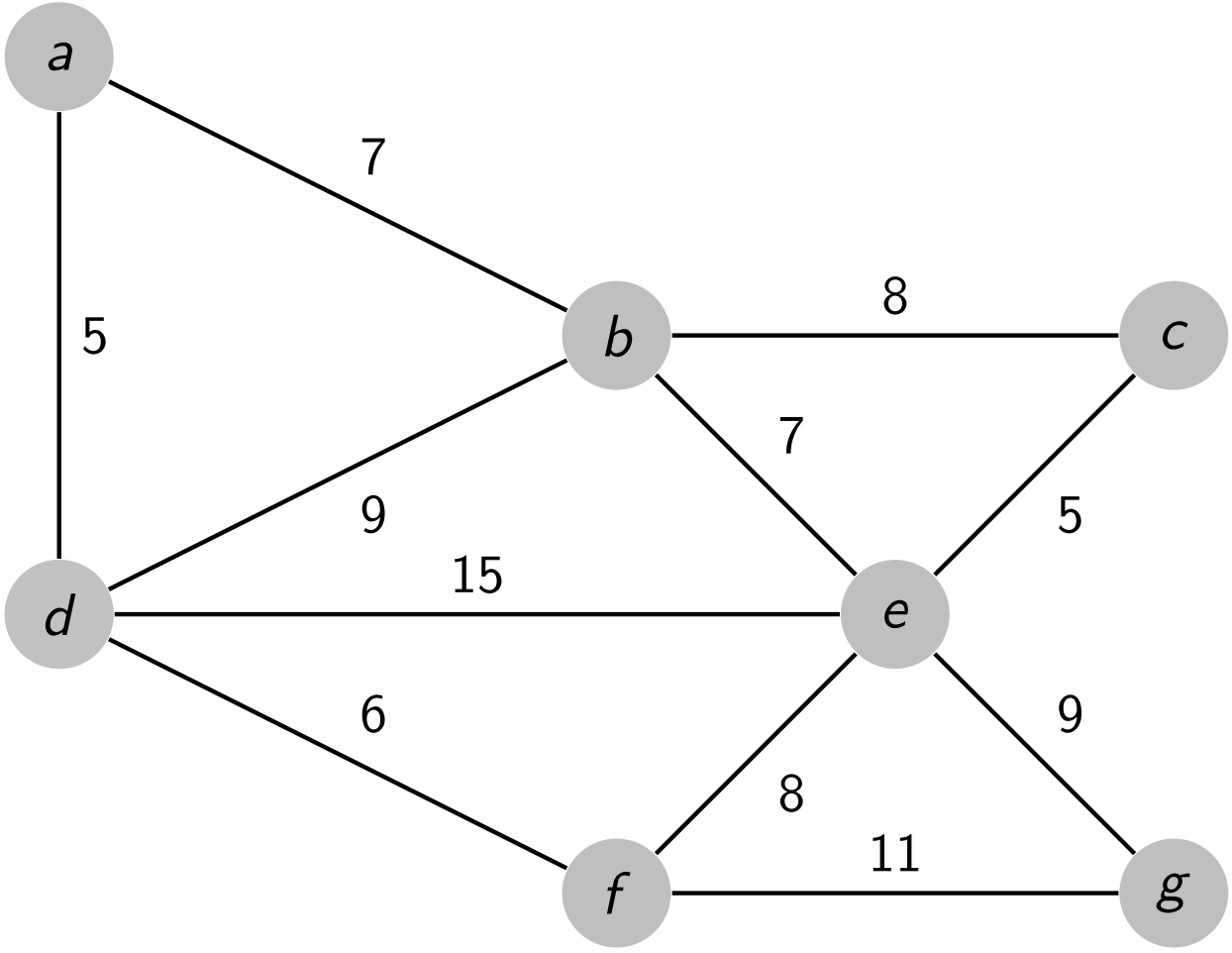
- http://uva.onlinejudge.org/external/106/10608.html

Đồ Thị Nâng Cao

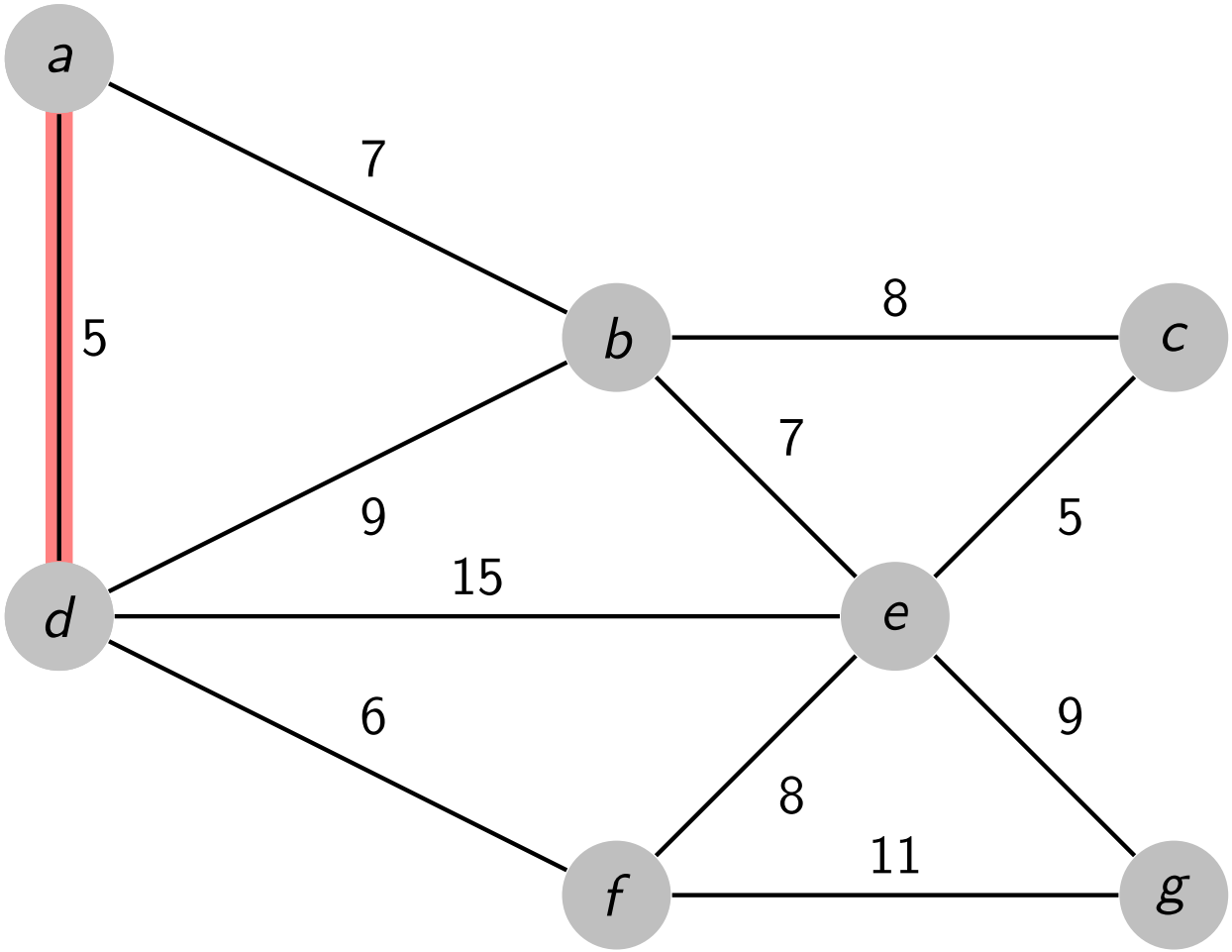
Cây khung nhỏ nhất - MST

- Cho đồ thị vô hướng có trọng số $G = (V, E)$
- $T = (V, F)$ với $F \subset E$ gọi là cây khung của G nếu như T là một cây (nghĩa là T không chứa chu trình và liên thông)
- Trọng số của T là tổng trọng số các cạnh thuộc F
- Bài toán đặt ra là tìm T có trọng số nhỏ nhất

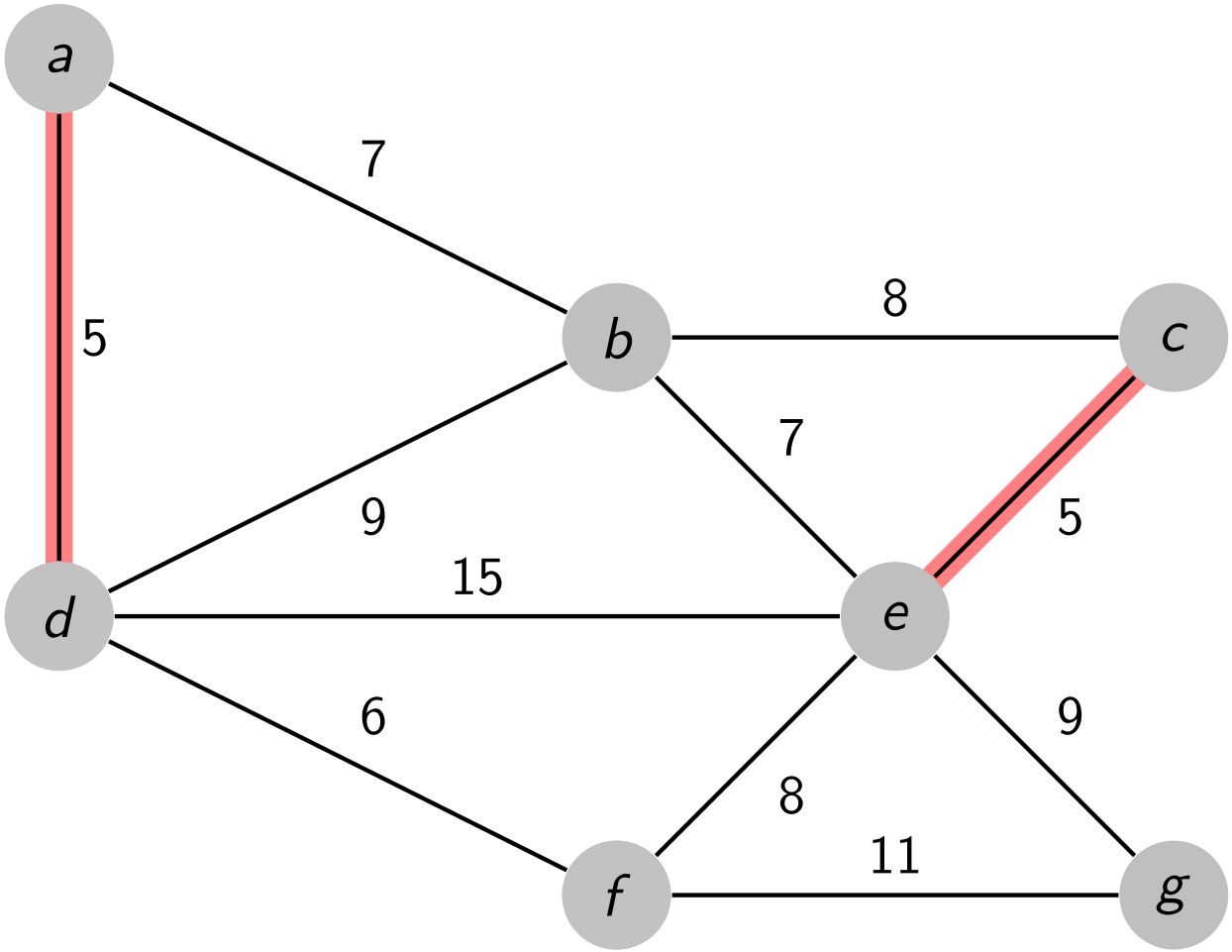
Demo thuật toán Kruskal



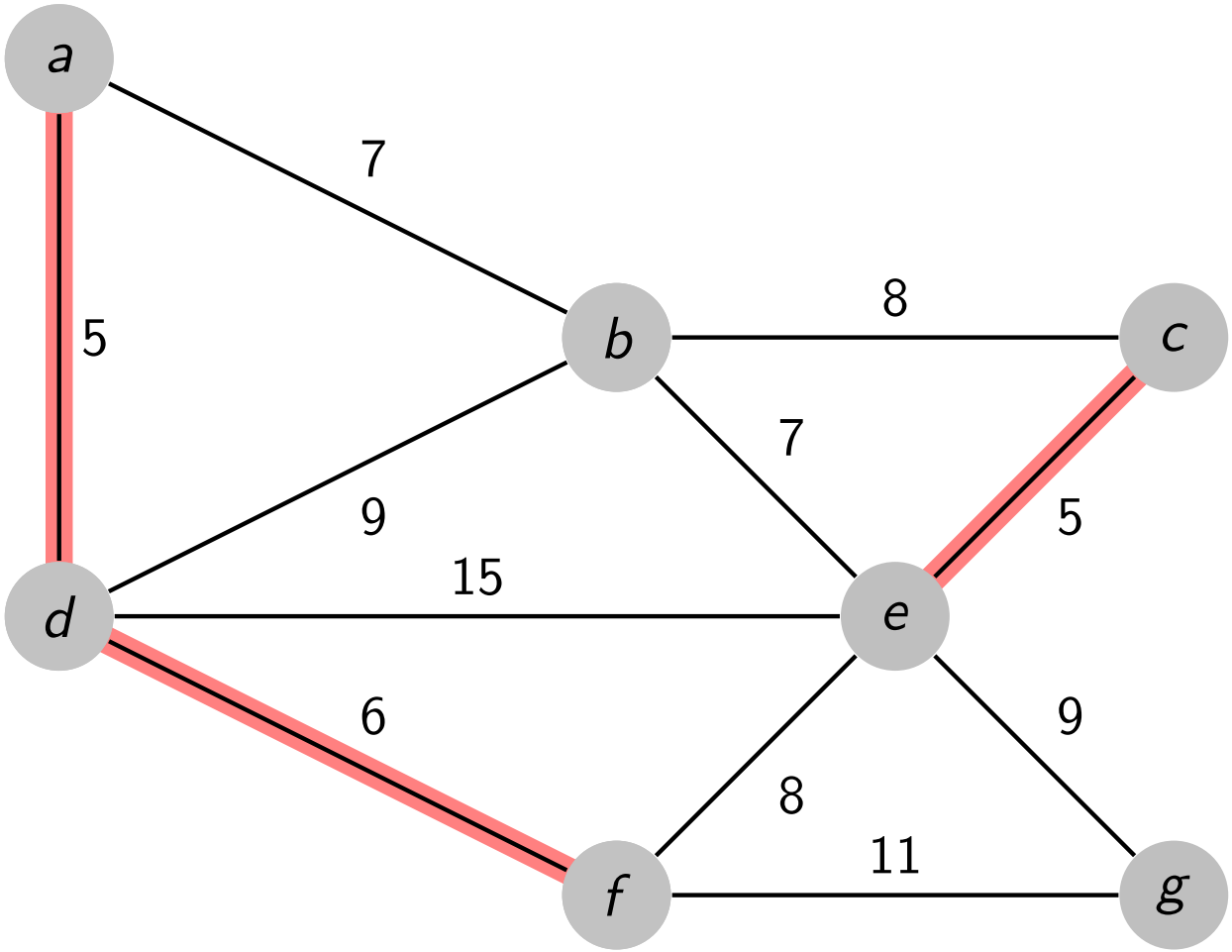
Demo thuật toán Kruskal



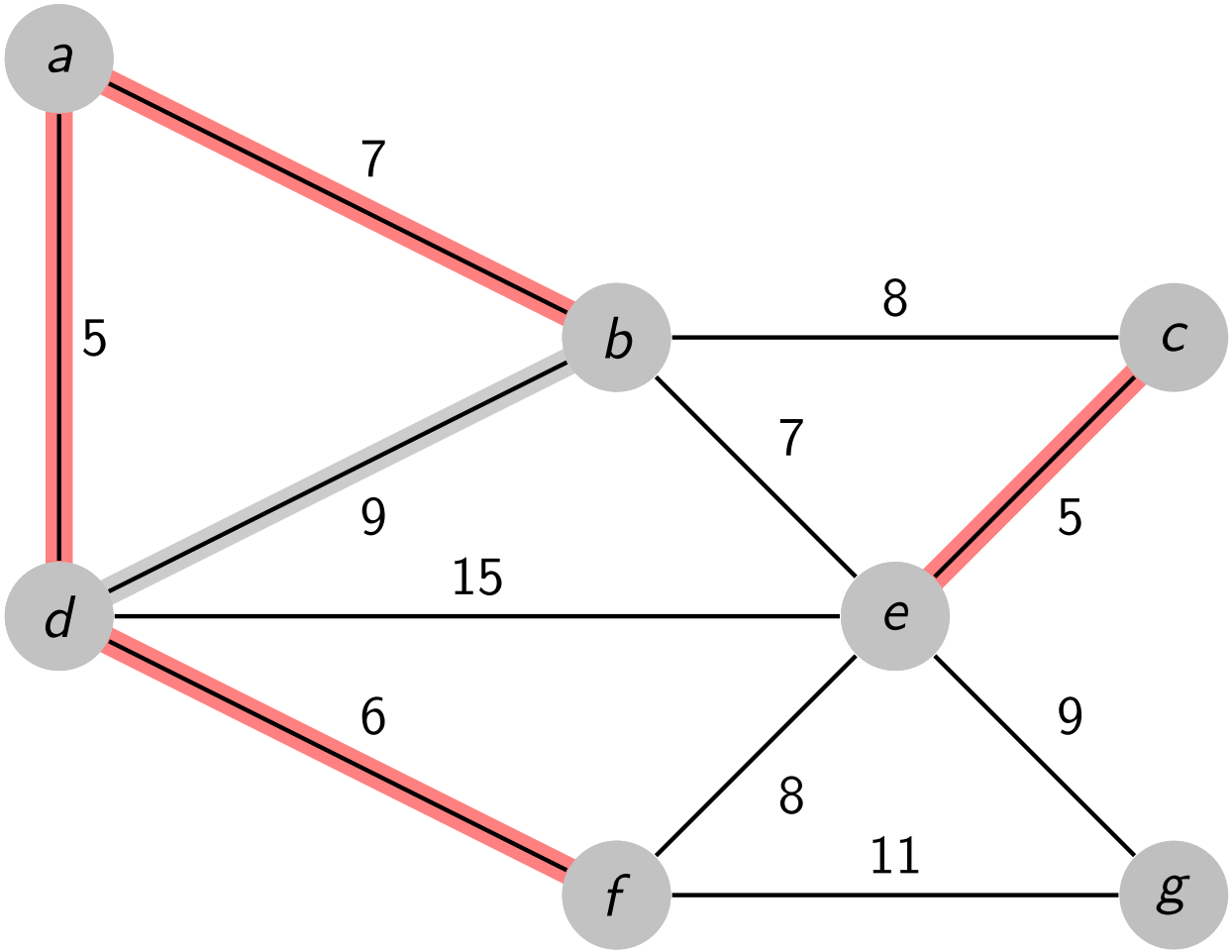
Demo thuật toán Kruskal



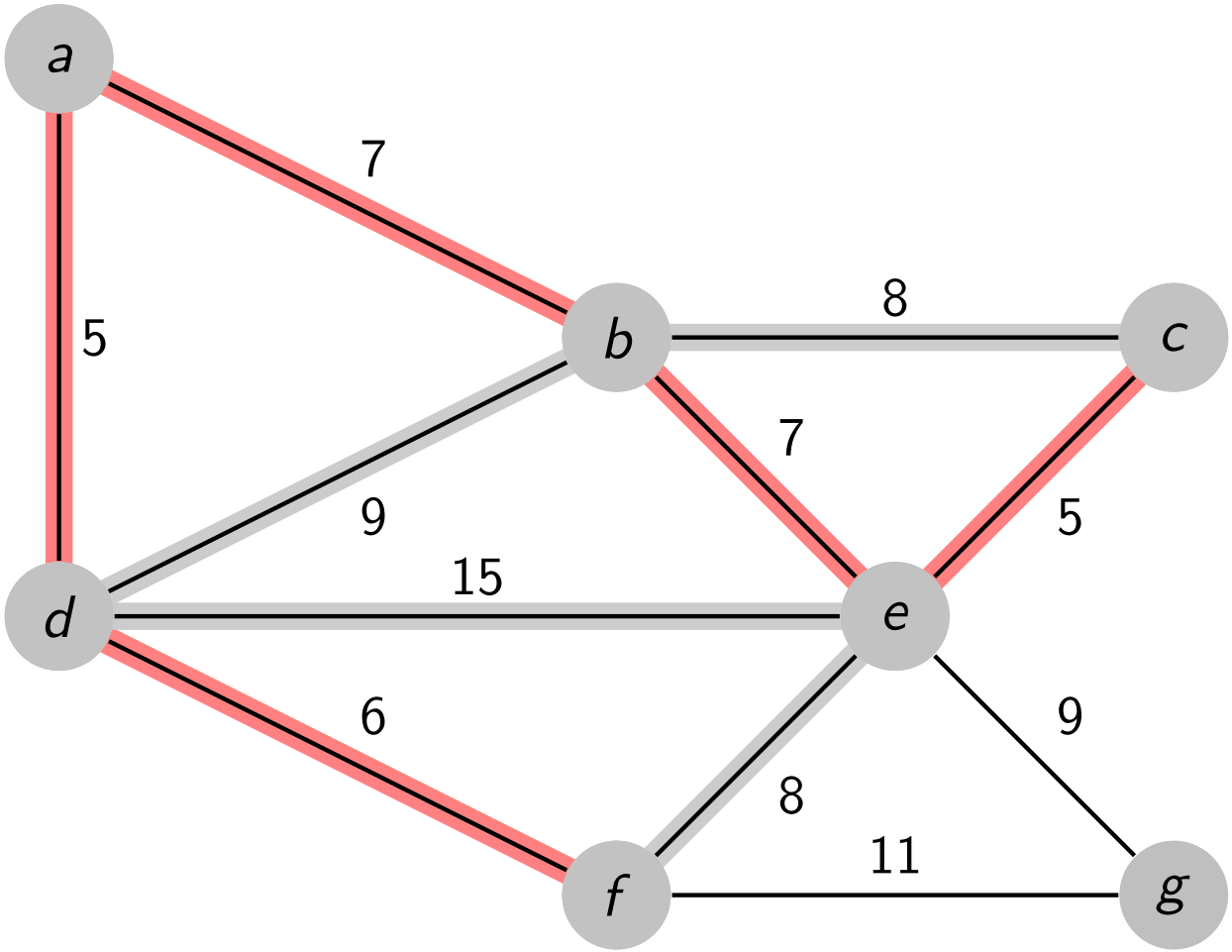
Demo thuật toán Kruskal



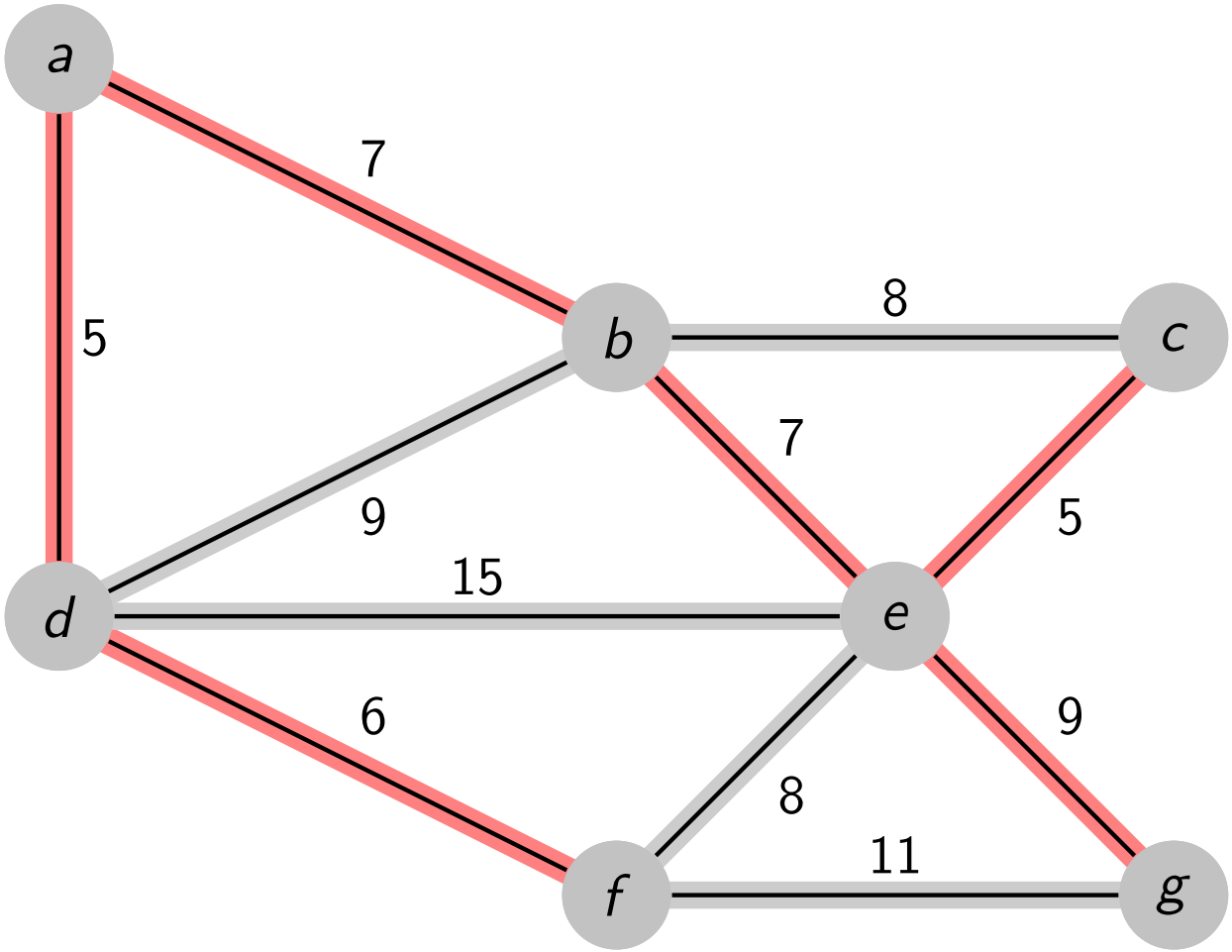
Demo thuật toán Kruskal



Demo thuật toán Kruskal



Demo thuật toán Kruskal



Thuật toán Kruskal

- Bài toán có thể giải bằng thuật toán tham lam sau

- Khởi tạo $F = \emptyset$
- Duyệt lần lượt các cạnh của đồ thị theo chiều tăng dần của trọng số
- Kết nạp vào T nếu như cạnh đó không tạo ra chu trình với các cạnh đã có trong T (có thể sử dụng Union-Find để lưu vết kiểm tra chu trình)
- Khi duyệt qua hết một lượt các cạnh ta sẽ thu được cây khung nhỏ nhất hoặc xác định không tồn tại cây khung của đồ thị

- Độ phức tạp $O(|E| \log |V|)$

Hai vấn đề quan trọng khi cài đặt thuật toán Kruskal:

1. Làm thế nào để xét được các cạnh từ cạnh có trong số nhỏ tới cạnh có trọng số lớn. Ta có thể thực hiện bằng cách **sắp xếp danh sách cạnh** theo thứ tự không giảm của trọng số, sau đó duyệt từ đầu đến cuối danh sách cạnh $\Rightarrow$ Có thể sử dụng thuật toán HeapSort cho phép chọn lần lượt các cạnh từ cạnh trọng số nhỏ nhất tới cạnh trọng số lớn nhất ra khỏi Heap.
2. Làm thế nào kiểm tra xem việc thêm một cạnh có tạo thành chu trình đơn trong T hay không. Để ý rằng các cạnh trong T ở các bước sẽ tạo thành một rừng (đồ thị không có chu trình đơn). Vì vậy muốn thêm một cạnh (u, v) vào T mà không tạo thành chu trình đơn thì (u, v) phải **nối hai cây khác nhau** của rừng T, bởi nếu u, v thuộc cùng một cây thì sẽ tạo thành chu trình đơn trong cây đó.
   Ban đầu, khởi tạo rừng T gồm n cây, mỗi cây chỉ gồm đúng một đỉnh $\rightarrow$ mỗi khi xét đến cạnh nối hai cây khác nhau của rừng T thi ta sẽ kết nạp cạnh đó vào $T \Rightarrow$ hợp nhất hai cây đó lại thành một cây.

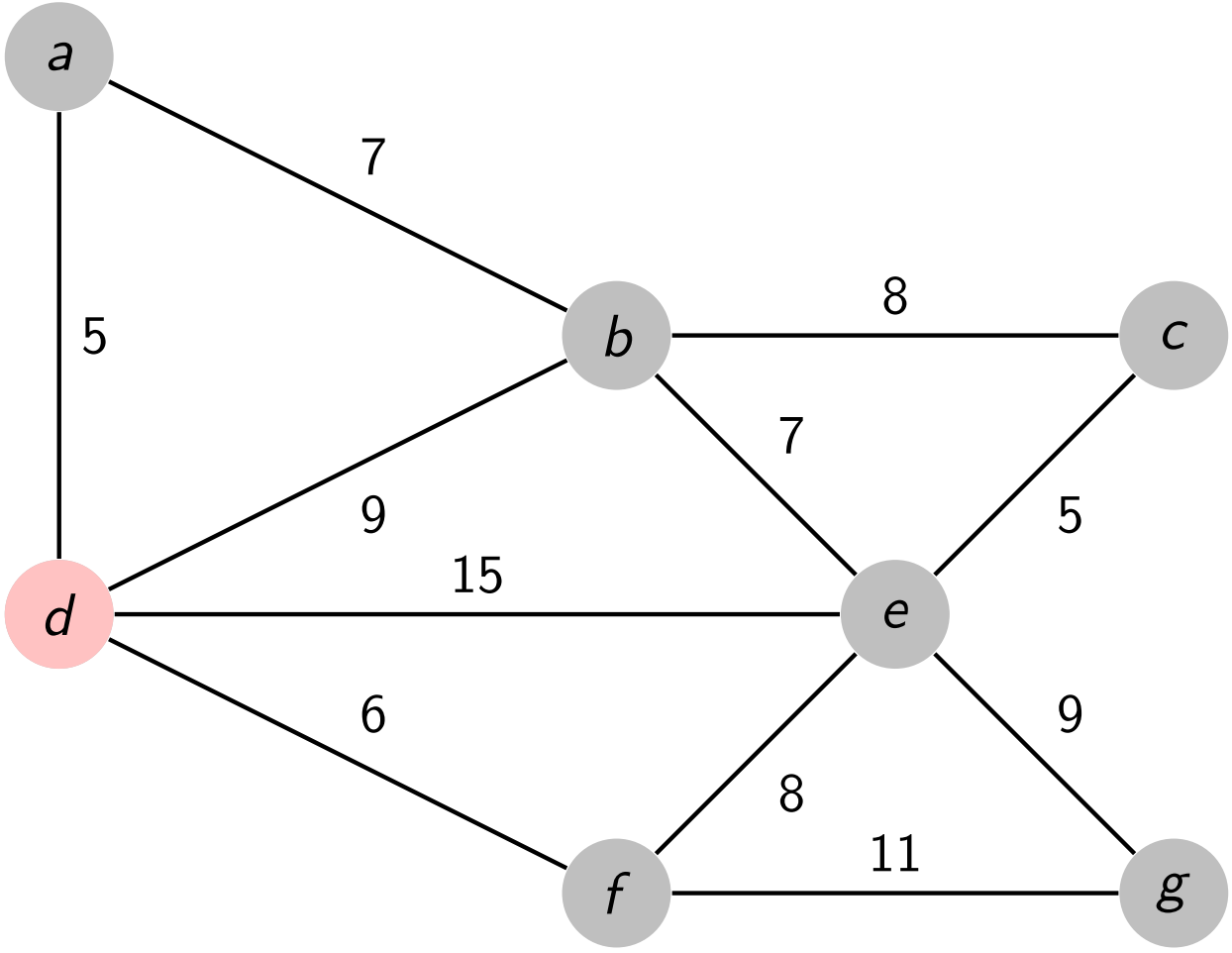
Kruskal

```
bool edge_cmp(const edge &a, const edge &b) {
    return a.weight < b.weight;
}

vector<edge> mst(int n, vector<edge> edges) {
    union_find uf(n);
    sort(edges.begin(), edges.end(), edge_cmp);

    vector<edge> res;
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i].u,
            v = edges[i].v;

        if (uf.find(u) != uf.find(v)) {
            uf.unite(u, v);
            res.push_back(edges[i]);
        }
    }
    return res;
}
```

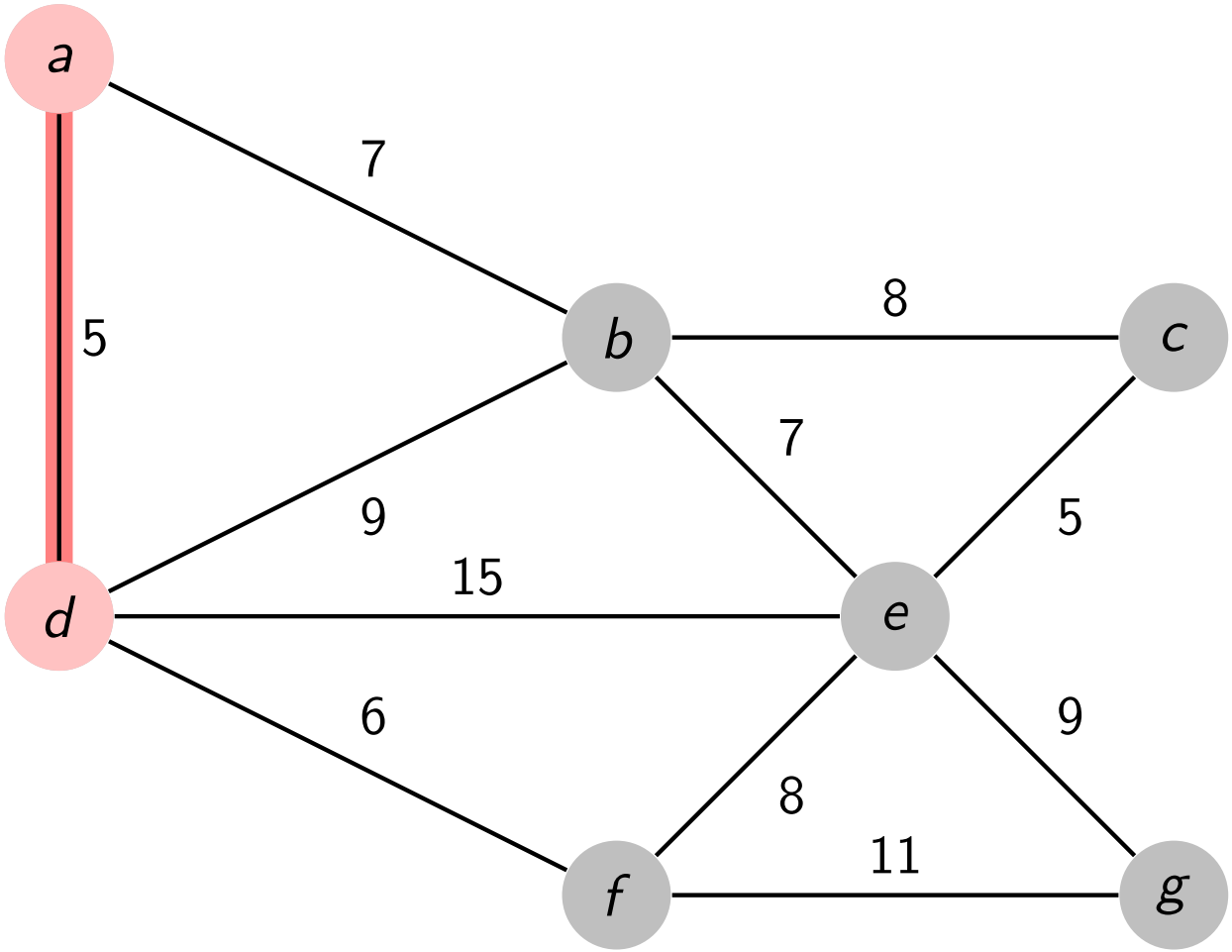
Bài toán ví dụ: Dark roads

- http://uva.onlinejudge.org/external/116/11631.html

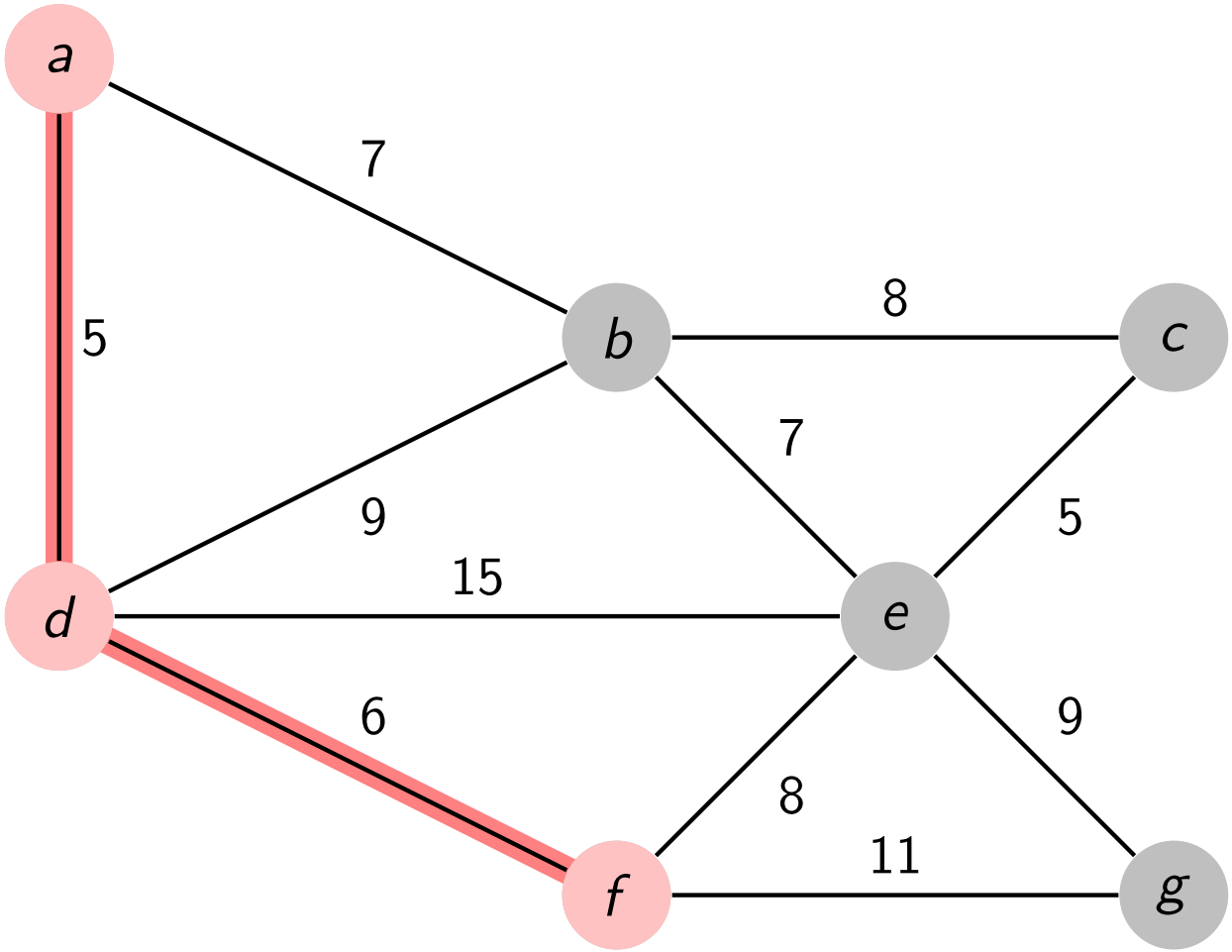
Demo thuật toán Prim



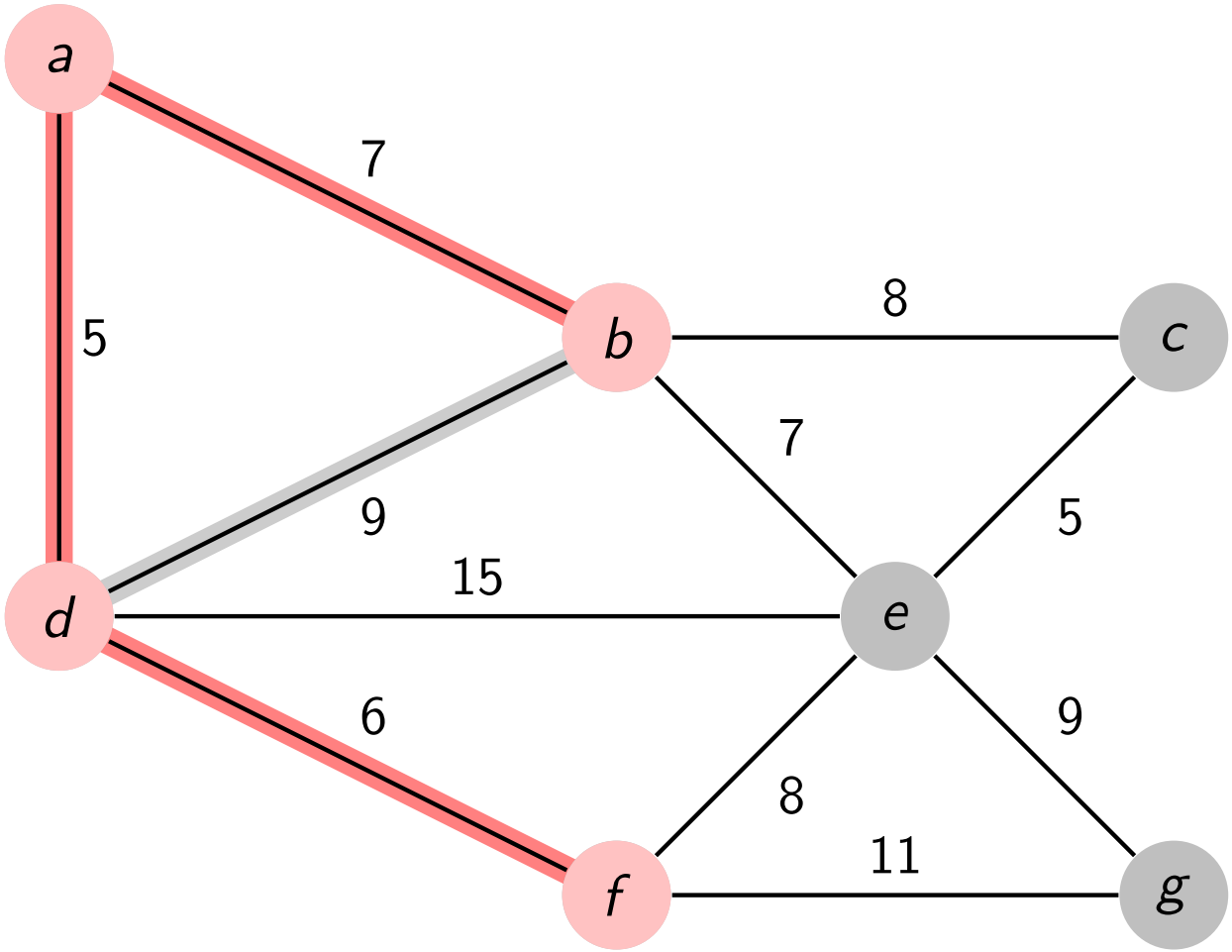
Demo thuật toán Prim



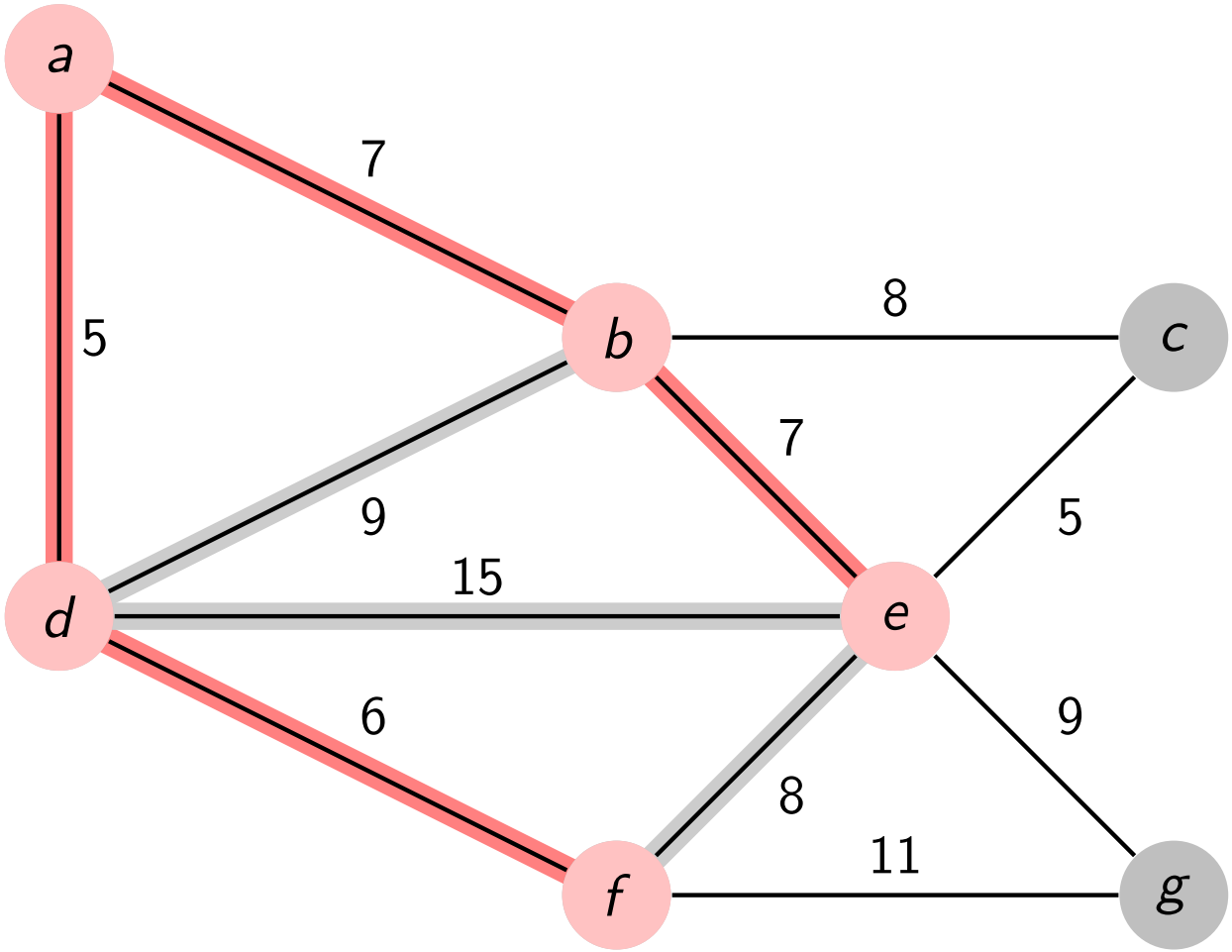
Demo thuật toán Prim



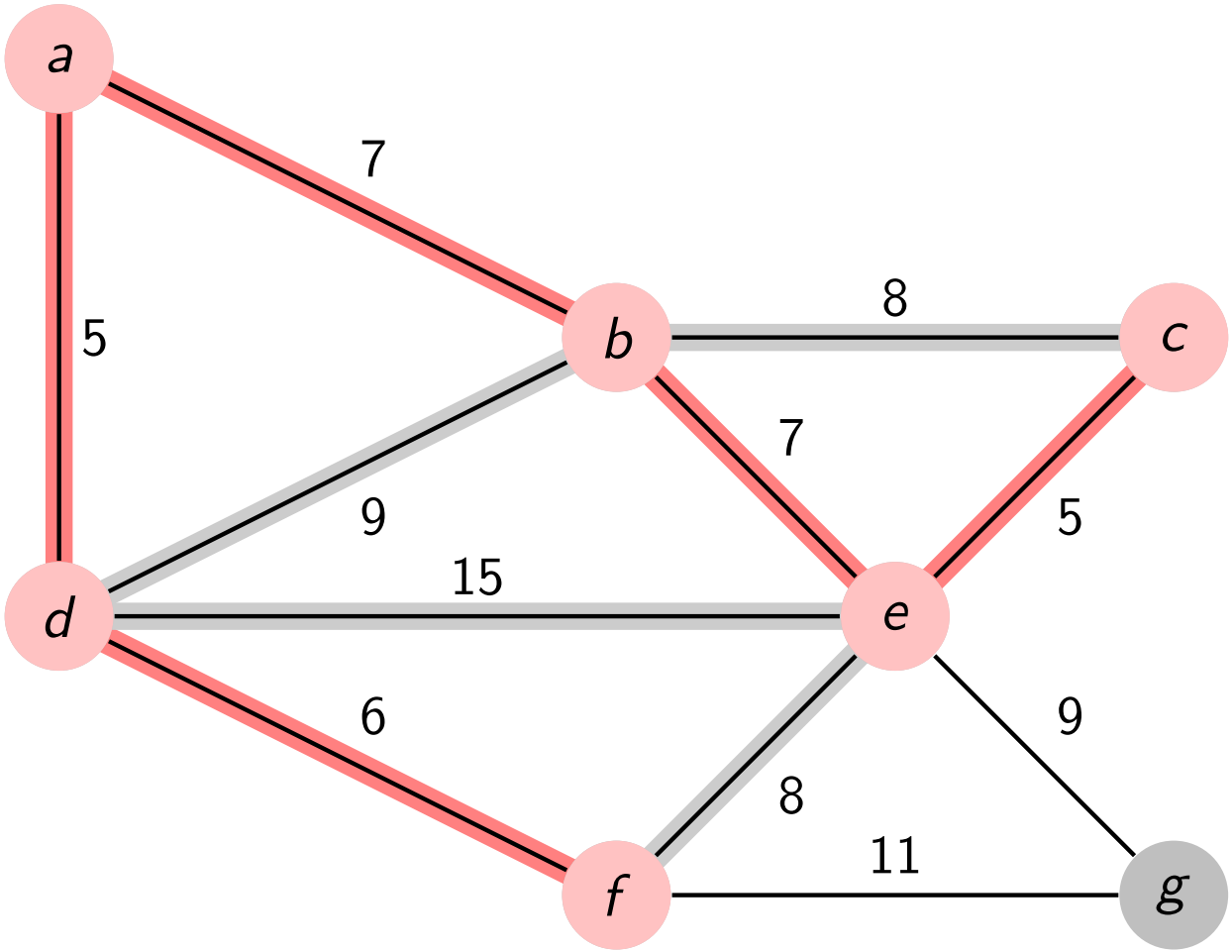
Demo thuật toán Prim



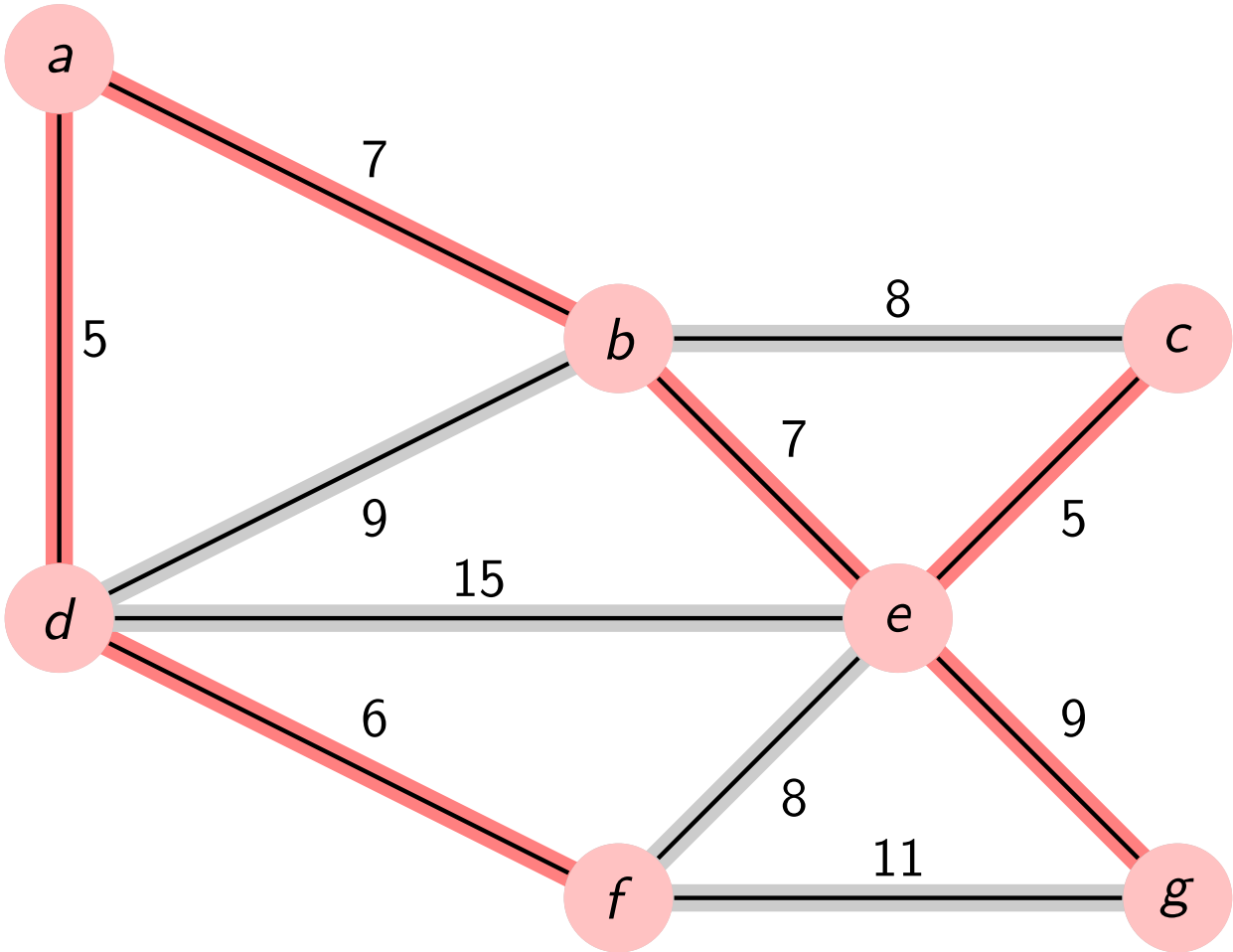
Demo thuật toán Prim



Demo thuật toán Prim



Demo thuật toán Prim



Prim

- Bài toán MST có thể giải bằng thuật toán tìm kiếm theo độ ưu tiên

- B1: Chọn tùy ý một đỉnh s, khởi tạo $V_T = \{s\}$, $E_T = \emptyset$
- B2: Nếu $|E_T| = |V| - 1$ thì đưa ra cây $T = (V_T, E_T)$, kết thúc thuật toán
- B3: Chọn cạnh $e = (u, v, w)$ có $u \in V_T$, $v \notin V_T$, w nhỏ nhất có thể
- B4: Nạp cạnh e vào cây, tức là gán $V_T = V_T \cup \{v\}$, $E_T = E_T \cup \{e\}$. Quay lại B2

- Độ phức tạp $O(\min(|V|^2, (|V| + |E|) \log |V|))$

Về mặt kỹ thuật cài đặt, ta có thể làm như sau:

- Sử dụng mảng đánh dấu inT. $inT[v] = 0$ nếu như đỉnh v chưa bị kết nạp vào T.
- Gọi $bestW[v]$ là khoảng cách từ v tới T. Ban đầu khởi tạo $bestW[1] = 0$ còn $bestW[2] = bestW[3] = \ldots = bestW[n] = +\infty$.
- Gọi $bestAdj[v]$ là đỉnh gần v nhất của cây khung.
- Tại mỗi bước chọn đỉnh đưa vào T, ta sẽ chọn đỉnh u nào ngoài T và có $bestW[u]$ nhỏ nhất. Khi kết nạp u vào T rồi thì rõ ràng các nhãn $bestW[v]$ sẽ thay đổi: $bestW[v]_{new} = min(bestW[v]_{old}, W[u, v])$.

Prim $O(|V|^2)$

```
vector<edge> mst(int sn, vector<vector<edge>> adj) {
    vector<bool> inT(n+1, false);//vertices set of mst
    vector<edge> res;//edges set of mst
    vector<int> bestW(n+1, 1e9), bestA(n+1, -1);
    bestW[1] = 0;
    while (res.size() < n-1){
        int u = -1, v = -1, w = 1e9;
        for (int x = 1; x <= n; ++x)
            if (inT[x] == false && bestW[x] < w){
                u = bestAdj[x], v = x, w = bestW[x];
            }
        if (v == -1) return res;//Given graph is not connected
        inT[v] = true;
        for (edge e : adj[v])
            if (bestW[e.v] > e.weight){
                bestW[e.v] = e.weight;
                bestAdj[e.v] = e.u;
            }
        if (v != 1) res.push_back({u, v, w});
    }
    return res;
}
```

Prim $(|V| + |E|) \log |V|)$

```
vector<edge> mst(int n, vector<vector<edge>> adj) {
    priority_queue< pair<int,int>, vector<pair<int,int>>,
                    greater<pair<int,int>> > pq;
    vector<edge> res;//edges set of mst
    vector<int> bestW(n+1, 1e9), bestAdj(n+1, -1);
    bestW[1] = 0;
    pq.push({bestW[1], 1});
    while (res.size() < n-1){
        while(!pq.empty() &&
          pq.top().first != bestW[pq.top().second]) pq.pop();
        if (pq.empty()) return res;//Given graph is not connect
        int w=pq.top().first, v=pq.top().second, u=bestAdj[v];
        for (edge e : adj[v])
            if (bestW[e.v] > e.weight){
                bestW[e.v] = e.weight;
                bestAdj[e.v] = e.u;
                pq.push({bestW[e.v], e.v});
            }
        if (v != 1) res.push_back({u, v, w});
    }
    return res;
}
```

Đồ Thị Nâng Cao

Đường đi ngắn nhất

- Cho đồ thị có trọng số $G = (V, E)$ (vô hướng hoặc có hướng)
- Cho hai đỉnh u, v, hãy tìm đường đi ngắn nhất từ u đến v?

- Nếu tất cả trọng số trên các cạnh đều bằng nhau, bài toán có thể giải bằng tìm kiếm theo chiều rộng BFS
- Tất nhiên là đại đa số các trường hợp không như vậy...

Đường đi ngắn nhất

- Có rất nhiều thuật toán hiện biết giải bài toán tìm đường đi ngắn nhất
- Cũng giống như thuật toán tìm kiếm theo chiều rộng BFS, các thuật toán này tìm đường đi ngắn nhất từ đỉnh xuất phát đến tất cả các đỉnh còn lại
- Ở đây ta xem xét các thuật toán Dijkstra, Bellman-Ford, và Floyd-Warshall

Thuật toán Dijkstra

```
vector<edge> adj[100];
vector<int> dist(100, INF);

void dijkstra(int start) {
    dist[start] = 0;
    priority_queue<pair<int, int>,
                   vector<pair<int, int> >,
                   greater<pair<int, int> > > pq;
    pq.push(make_pair(dist[start], start));

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (int i = 0; i < adj[u].size(); i++) {
            int v = adj[u][i].v;
            int w = adj[u][i].weight;

            if (w + dist[u] < dist[v]) {
                dist[v] = w + dist[u];
                pq.push(make_pair(dist[v], v));
            }
        }
    }
}
```

Thuật toán Dijkstra

- Độ phức tạp thuật toán $O(|E| \log |V|)$
- Lưu ý là thuật toán chỉ đúng trong trường hợp trọng số không âm

Thuật toán Bellman-Ford

```
void bellman_ford(int n, int start) {

    dist[start] = 0;

    for (int i = 0; i < n - 1; i++) {
        for (int u = 0; u < n; u++) {
            for (int j = 0; j < adj[u].size(); j++) {
                int v = adj[u][j].v;
                int w = adj[u][j].weight;
                dist[v] = min(dist[v], w + dist[u]);
            }
        }
    }
}
```

Thuật toán Bellman-Ford

- Độ phức tạp $O(|V| \times |E|)$
- Có thể sử dụng để tìm ra các chu trình trọng số âm

Thuật toán Floyd-Warshall

- Sử dụng Qui hoạch động để tính đường đi ngắn nhất thế nào?
- Gọi $\mathrm{sp}(k, i, j)$ là trọng số đường đi ngắn nhất từ i đến j nếu như chỉ cho phép đi trong những đỉnh $0, \ldots, k$
- Điều kiện biên 1: $\mathrm{sp}(k, i, j) = 0$ nếu $i = j$
- Điều kiện biên 2: $\mathrm{sp}(-1, i, j) = \mathrm{weight}[a][b]$ nếu $(i, j) \in E$
- Điều kiện biên 3: $\mathrm{sp}(-1, i, j) = \infty$
- $$\mathrm{sp}(k, i, j) = \min \begin{cases} \mathrm{sp}(k-1, i, k) + \mathrm{sp}(k-1, k, j) \\ \mathrm{sp}(k-1, i, j) \end{cases}$$

Thuật toán Floyd-Warshall

```
int dist[1000][1000];
int weight[1000][1000];

void floyd_warshall(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = i == j ? 0 : weight[i][j];
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                dist[i][j] =
                min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
```

Thuật toán Floyd-Warshall

- Tính toàn bộ các đường đi ngắn nhất giữa các cặp đỉnh
- Độ phức tạp $O(|V|^3)$
- Dễ cài đặt

Đồ Thị Nâng Cao

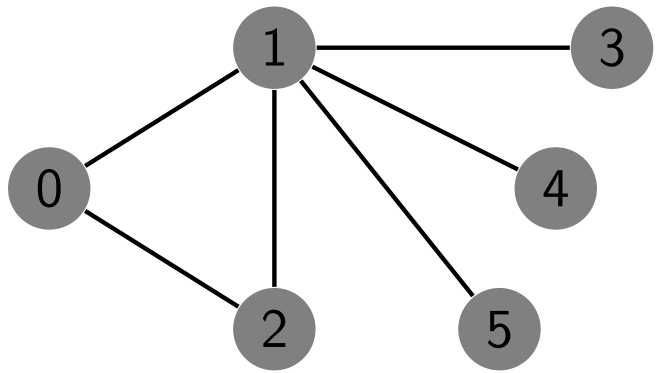
Một số bài toán quen thuộc trên đồ thị

- Đây là những bài toán cơ bản rất hay được sử dụng để ra bài trong các kỳ thi
- Thường xuyên được ẩn chứa kín trong phát biểu bài toán

- Xét một số ví dụ

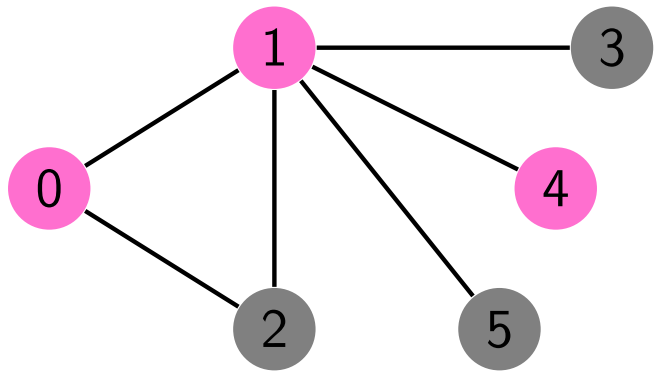
Bài toán phủ đỉnh - MVC

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S, sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



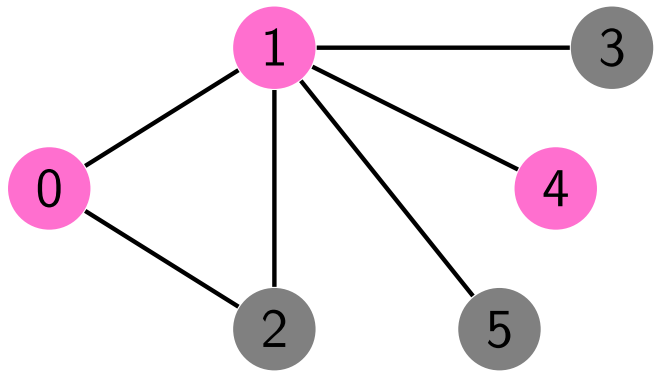
Bài toán phủ đỉnh - MVC

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S, sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



Bài toán phủ đỉnh - MVC

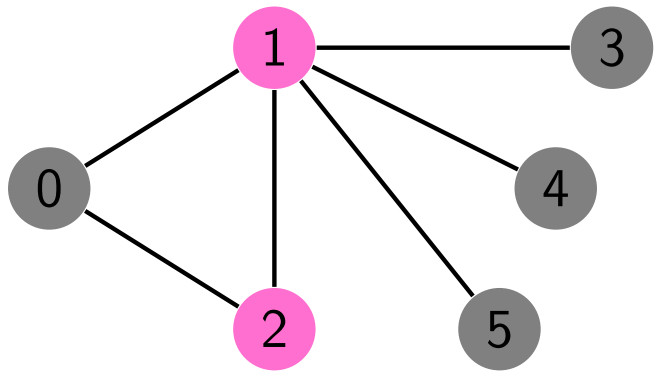
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S, sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất

Bài toán phủ đỉnh - MVC

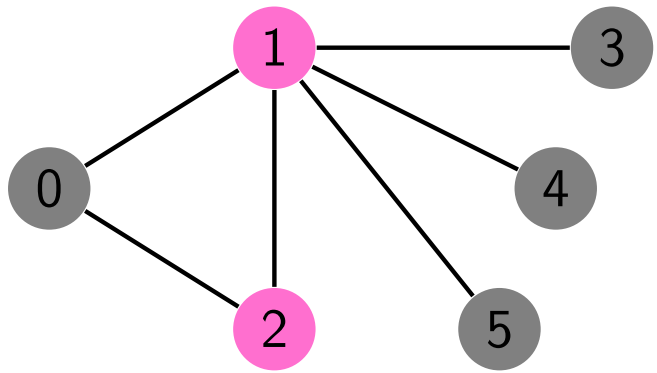
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S, sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất

Bài toán phủ đỉnh - MVC

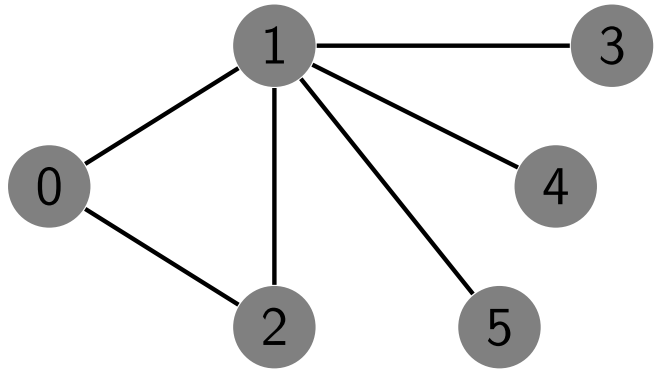
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một phủ đỉnh là một tập con các đỉnh S, sao cho với mỗi cạnh $(u, v) \subset E$, hoặc u hoặc v (hoặc cả hai) thuộc S



- Hãy tìm một phủ đỉnh có lực lượng nhỏ nhất
- Đây là bài toán NP-khó đối với đồ thị bất kỳ

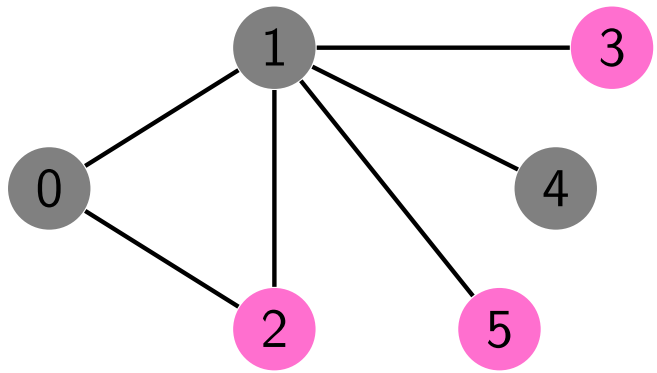
Bài toán tập độc lập - MIS

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S, sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



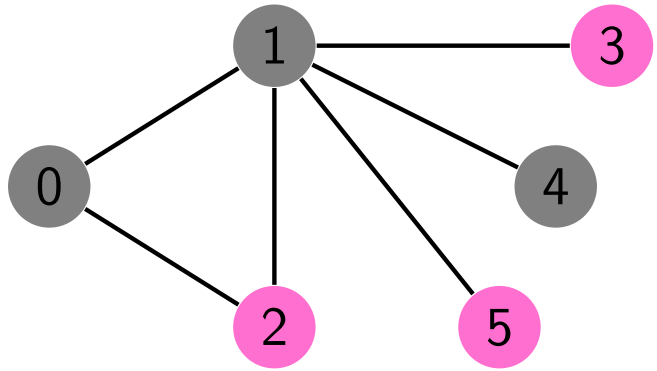
Bài toán tập độc lập - MIS

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S, sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



Bài toán tập độc lập - MIS

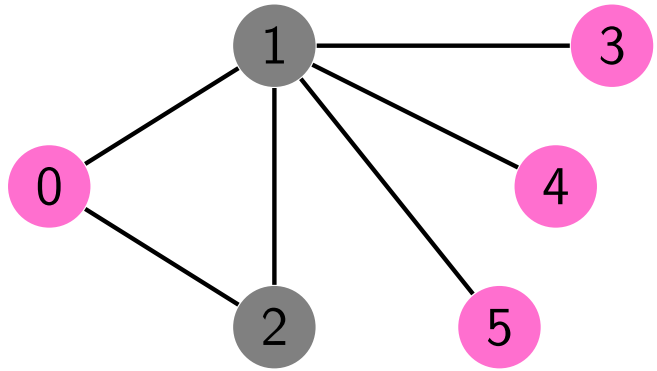
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S, sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất

Bài toán tập độc lập - MIS

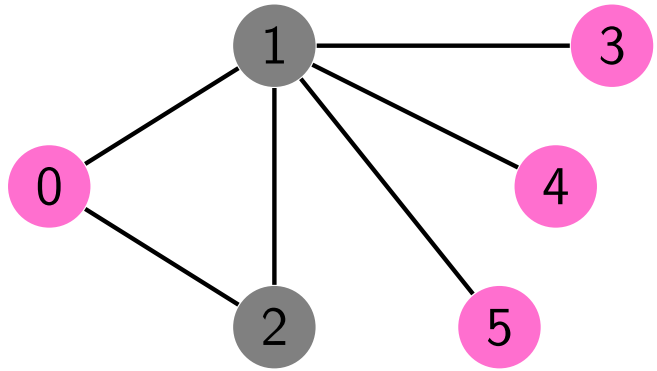
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S, sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất

Bài toán tập độc lập - MIS

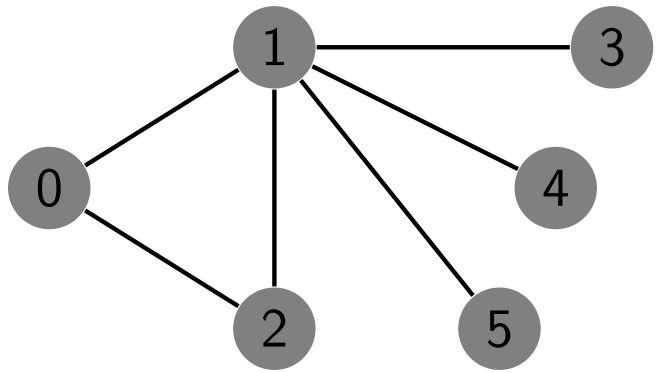
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tập độc lập là một tập con các đỉnh S, sao cho không có hai đỉnh u, v nào trong S kề với nhau trong G



- Hãy tìm một tập độc lập có lực lượng lớn nhất
- Đây là bài toán NP-khó đối với đồ thị bất kỳ

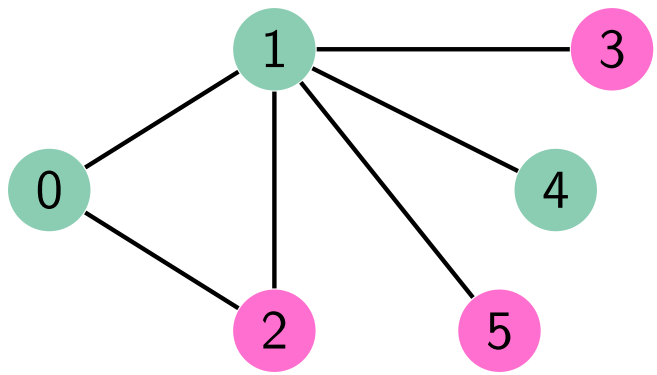
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



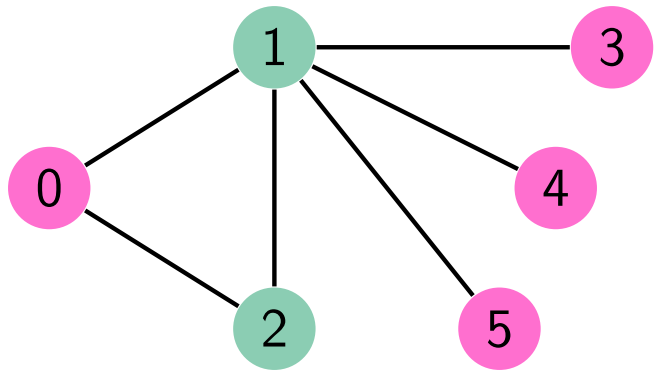
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



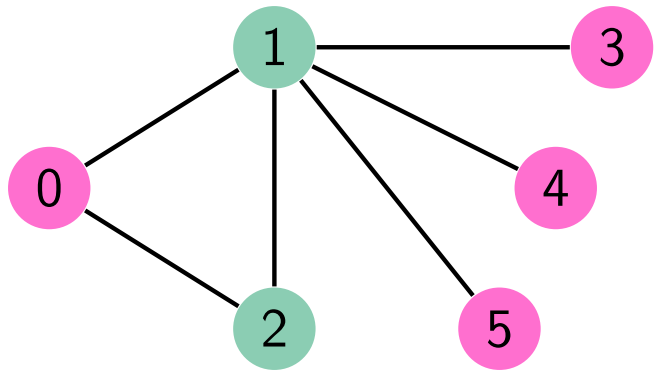
Mối quan hệ giữa MVC và MIS

- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



Mối quan hệ giữa MVC và MIS

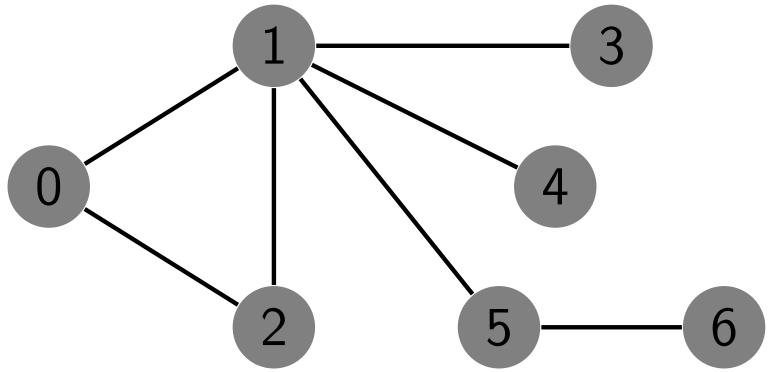
- Hai bài toán trên có mối liên quan chặt chẽ với nhau
- Một tập con của các đỉnh là một phủ đỉnh khi và chỉ khi tập bù của nó là tập độc lập



- Lực lượng của một phủ tập có kích thước nhỏ nhất cộng với lực lượng của tập độc lập có kích thước lớn nhất bằng tổng số đỉnh của đồ thị

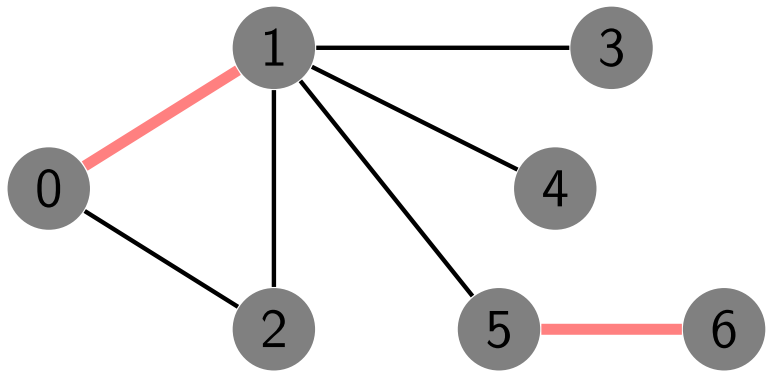
Bài toán ghép cặp

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một ghép cặp là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



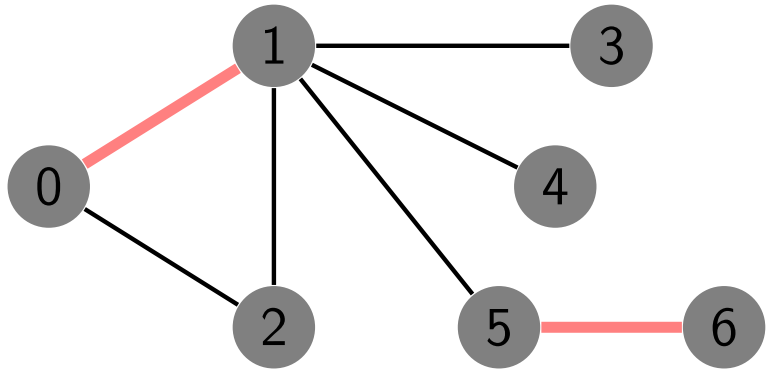
Bài toán ghép cặp

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một ghép cặp là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



Bài toán ghép cặp

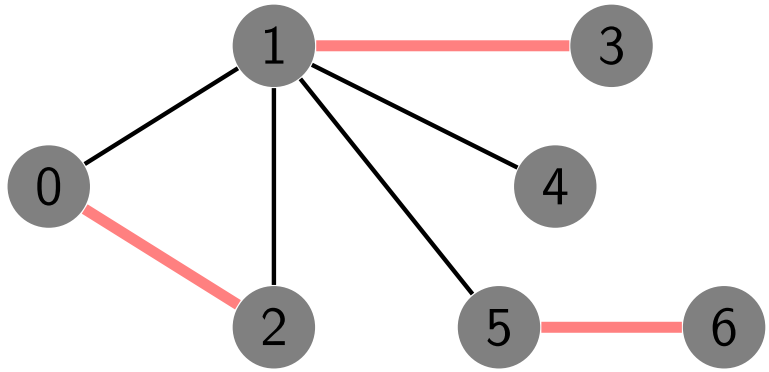
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một ghép cặp là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



- Hãy tìm một ghép cặp có lực lượng lớn nhất

Bài toán ghép cặp

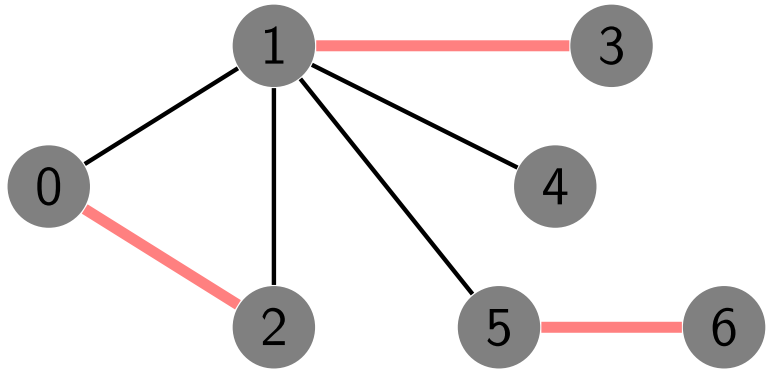
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một ghép cặp là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



- Hãy tìm một ghép cặp có lực lượng lớn nhất

Bài toán ghép cặp

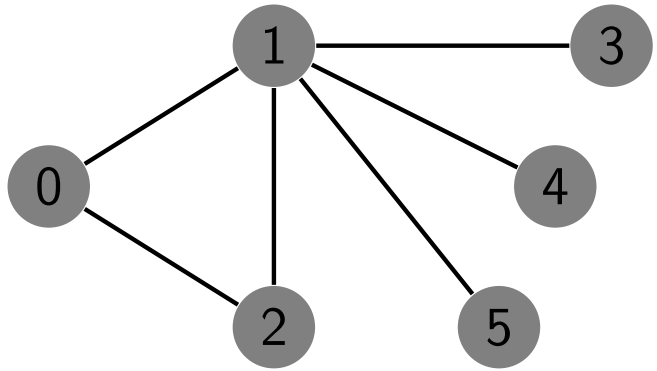
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một ghép cặp là một tập con các cạnh F sao cho mỗi đỉnh kề với tối đa một cạnh trong F



- Hãy tìm một ghép cặp có lực lượng lớn nhất
- Có thuật toán $O(|V|^4)$ cho đồ thị tổng quát, nhưng cài đặt khá phức tạp

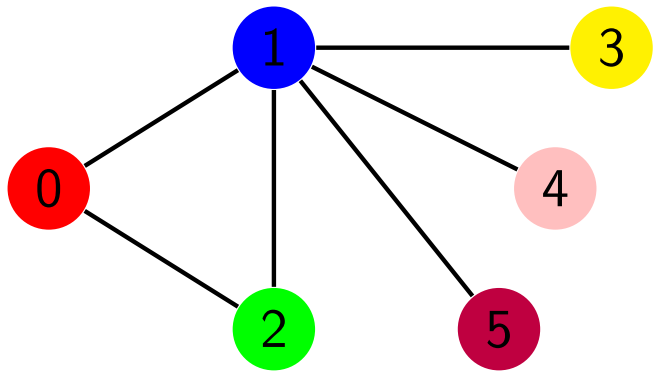
Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tô màu của đồ thị là một cách gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



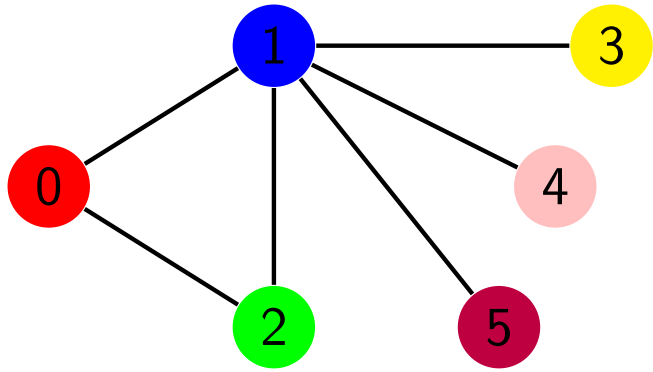
Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tô màu của đồ thị là một cách gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



Bài toán tô màu đồ thị

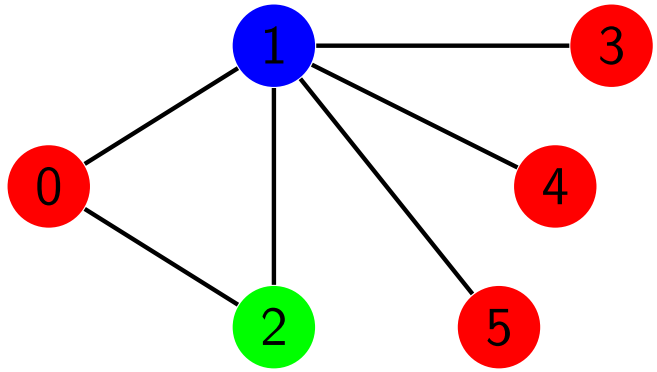
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tô màu của đồ thị là một cách gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất

Bài toán tô màu đồ thị

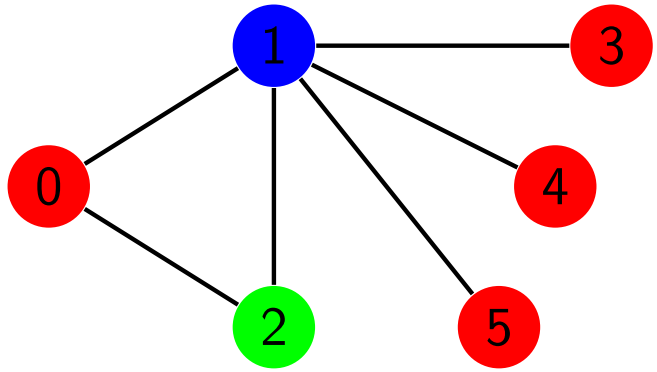
- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tô màu của đồ thị là một cách gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất

Bài toán tô màu đồ thị

- Cho đồ thị vô hướng không trọng số $G = (V, E)$
- Một tô màu của đồ thị là một cách gán các màu vào các đỉnh sao cho các đỉnh kề nhau không cùng màu



- Hãy tìm một cách tô màu sao cho sử dụng ít màu khác nhau nhất
- Đây là bài toán NP-khó trên đồ thị bất kỳ

Đồ Thị Nâng Cao

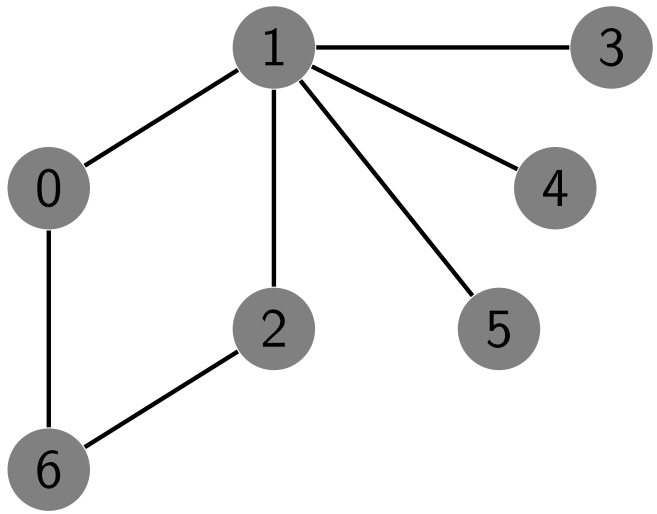
Một số đồ thị đặc biệt

- Tất cả các bài toán trên đều là NP-khó trong trường hợp đồ thị bất kỳ
- Còn trên một số loại đồ thị đặc biệt thì sao?

- Xét một số ví dụ

Đồ thị hai phía

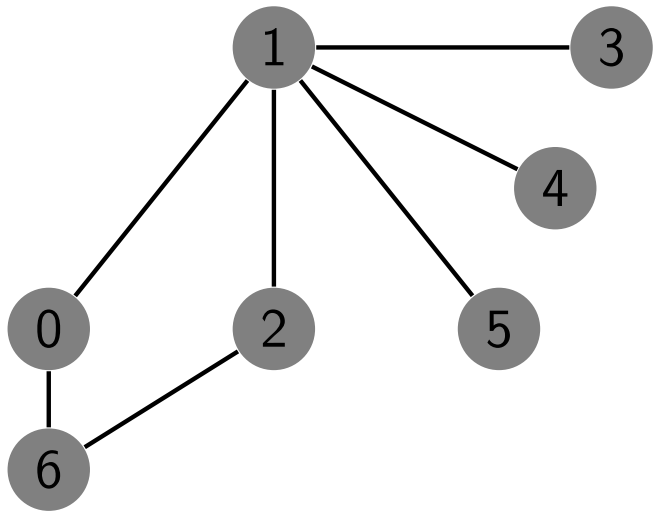
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

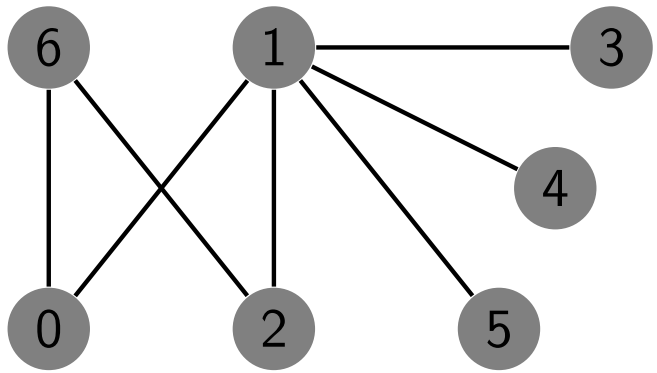
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

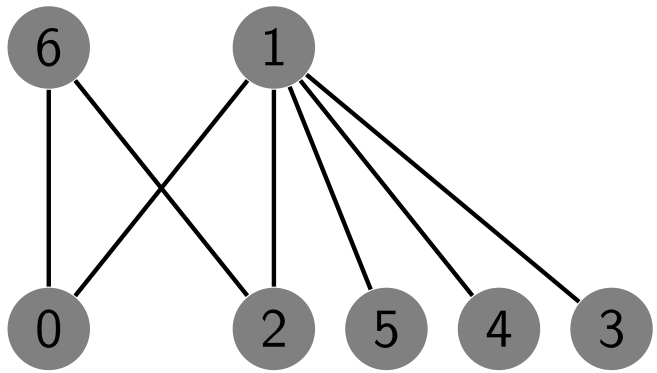
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

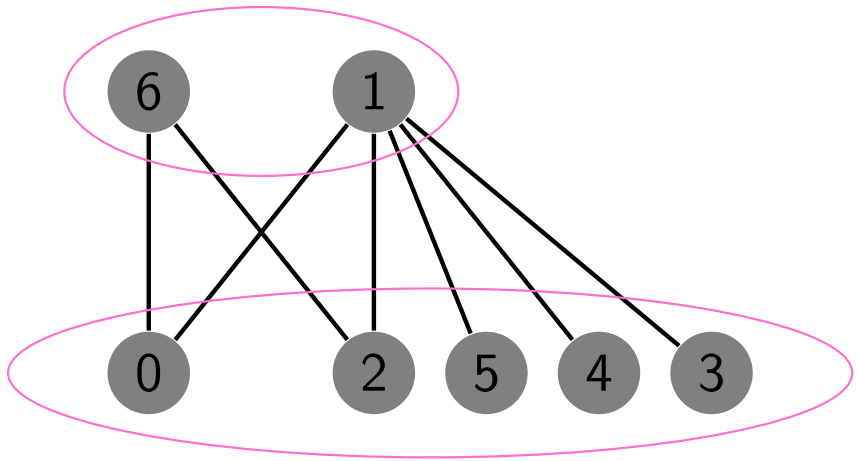
- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

- Một đồ thị là hai phía nếu như các đỉnh được phân chia thành hai tập sao cho với mỗi cạnh (u, v) thì u và v nằm ở hai tập khác nhau



- Làm thế nào để kiểm tra một đồ thị là hai phía?

Đồ thị hai phía

- Cần phải kiểm tra xem ta có thể chia tập đỉnh thành hai nhóm
- Lấy một đỉnh bất kỳ, giả sử nó thuộc nhóm 1
- Sau đó tất cả đỉnh kề với nó sẽ thuộc nhóm 2
- Tiếp theo tất cả đỉnh kề với các đỉnh nhóm 2 đó đều phải thuộc nhóm 1
- Và cứ kiểm tra như cậy...

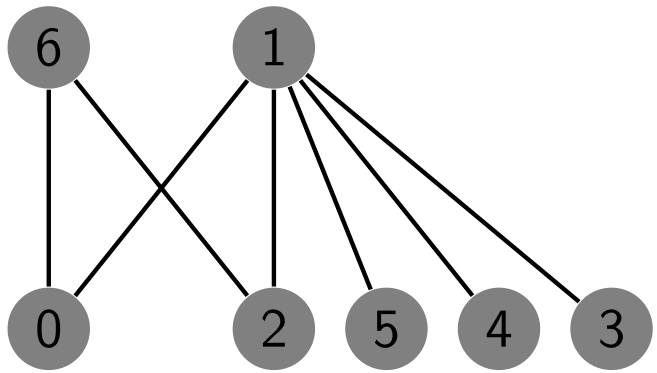
- Ta có thể thực hiện với thuật toán tìm kiếm theo chiều sâu DFS
- Nếu thấy điều vô lý xảy ra (nghĩa là một đỉnh phải nằm trong cả hai nhóm 1 và 2), thì đồ thị đó không phải là đồ thị hai phía

Đồ thị hai phía

```
vector<int> adj[1000];
vector<int> side(1000, -1);
bool is_bipartite = true;

void check_bipartite(int u) {
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (side[v] == -1) {
            side[v] = 1 - side[u];
            check_bipartite(v);
        } else if (side[u] == side[v]) {
            is_bipartite = false;
        }
    }
}

for (int u = 0; u < n; u++) {
    if (side[u] == -1) {
        side[u] = 0;
        check_bipartite(u);
    }
}
```

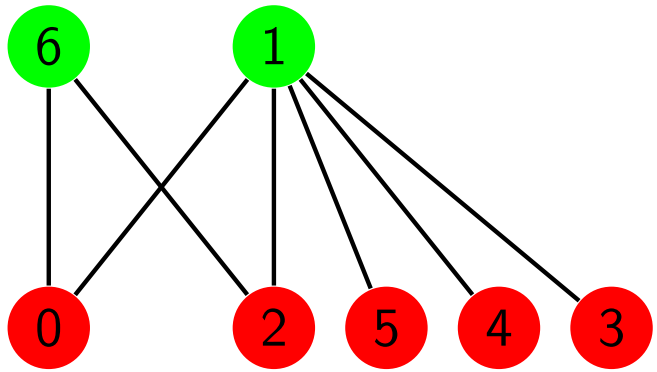
Bài toán tô màu trên đồ thị hai phía

- Làm thế nào để tô đỉnh bởi ít màu nhất trên đồ thị hai phía?



Bài toán tô màu trên đồ thị hai phía

- Làm thế nào để tô đỉnh bởi ít màu nhất trên đồ thị hai phía?



- Rất đơn giản, một phía tô bởi một màu, và phía kia tô bởi một màu khác

Ghép cặp trên đồ thị hai phía

- Bài toán tìm ghép cặp trên đồ thị hai phía rất quen thuộc
- *xem ví dụ*

- Nhó là thuật toán hiệu quả tìm ghép cặp lớn nhất trên đồ thị tổng quát là không quen thuộc

Định lý König

- Định lý König chỉ ra rằng lực lượng của một phủ đỉnh nhỏ nhất trên đồ thị hai phía bằng với lực lượng của ghép cặp lớn nhất trên đồ thị đó

- Do đó, để tìm một phủ đỉnh nhỏ nhất trên đồ thị hai phía, ta chỉ cần tìm ghép cặp lớn nhất với thuật toán hiệu quả quen thuộc
- Và do lực lượng của tập độc lập lớn nhaats chính là tổng số đỉnh của đồ thị trừ đi lực lượng của phủ đỉnh nhỏ nhất, nên ta có thể tính được tập độc lập lớn nhất một cách hiệu quả

Cây

- Đồ thị vô hướng là cây nếu nó không có chu trình
- Dễ dàng kiểm tra một đồ thị là cây bằng cách kiểm tra có tồn tại cạnh ngược hay không trên cây DFS
- Một cây với n đỉnh có chính xác $n - 1$ cạnh
- Giữa mỗi cặp đỉnh u, v trên cây tồn tại duy nhất một đường đi đơn, có thể sử dụng DFS hoặc BFS để tìm đường đi này

Cây

- Các bài toán trên áp dụng trên cây thế nào?
- Làm thế nào để tìm sắc số đỉnh cho cây? (sắc số số màu tối thiểu để tô các đỉnh đồ thị)

Cây

- Các bài toán trên áp dụng trên cây thế nào?
- Làm thế nào để tìm sắc số đỉnh cho cây? (sắc số số màu tối thiểu để tô các đỉnh đồ thị)
- Thực chất cây cũng giống như đồ thị hai phía...
- Vì sao? Lấy một đỉnh bất kỳ và coi đỉnh đó là gốc của cây. Tiếp theo các đỉnh có chiều cao chẵn thì cho thuộc về một phía, còn các đỉnh chiều cao lẻ thì cho thuộc vào phía còn lại
- Vì vậy tất cả các thuật toán hiệu quả trên đồ thị hai phía đều đúng cho cây

Cây

- Cây cũng rất thích hợp cho các thuật toán quy hoạch động, vì vậy rất nhiều bài toán trở nên dễ hơn nhiều trên cây vì lý do đó

Đồ thị có hướng không có chu trình - DAG

- Một đồ thị có hướng là một DAG nếu như nó không chứa chu trình
- Dễ dàng kiểm tra một đồ thị là DAG bằng cách kiểm tra xem có cạnh ngược nào không trên cây DFS
- Rất nhiều bài toán trở nên dễ dàng ở trên DAG, do có thể áp dụng quy hoạch động nhờ tính chất không có chu trình trên DAG
  - tính số lượng đường đi đơn từ u đến v
  - đường đi dài nhất từ u đến v