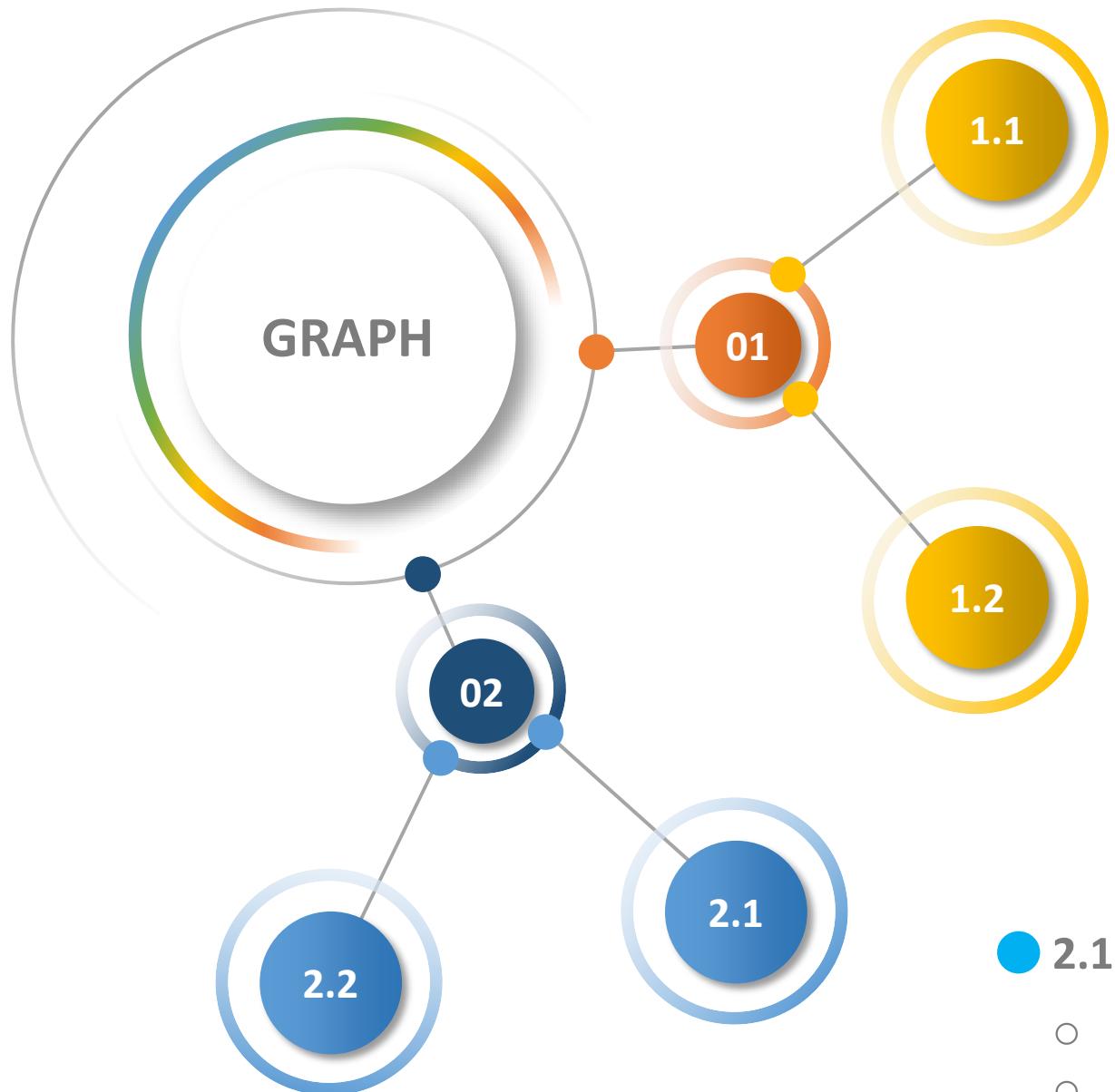




GRAPH ALGORITHMS GROUP 5

 Huy Hoàng

 Khanh Nam



● 01. Giới thiệu về đồ thị

● 1.1 Sơ lược về đồ thị

- Đồ thị là gì?
- Thế nào là bài toán đồ thị?
- ...

● 1.2 Một số ứng dụng

- Google map
- Mạng máy tính
- Social networking
- ...

● 02. Một vài thuật toán và ứng dụng của nó

● 2.1 Đường đi ngắn nhất

- Dijkstra
- Floyd
- A*

● 2.2 Luồng cực đại

Ford-Fullerson

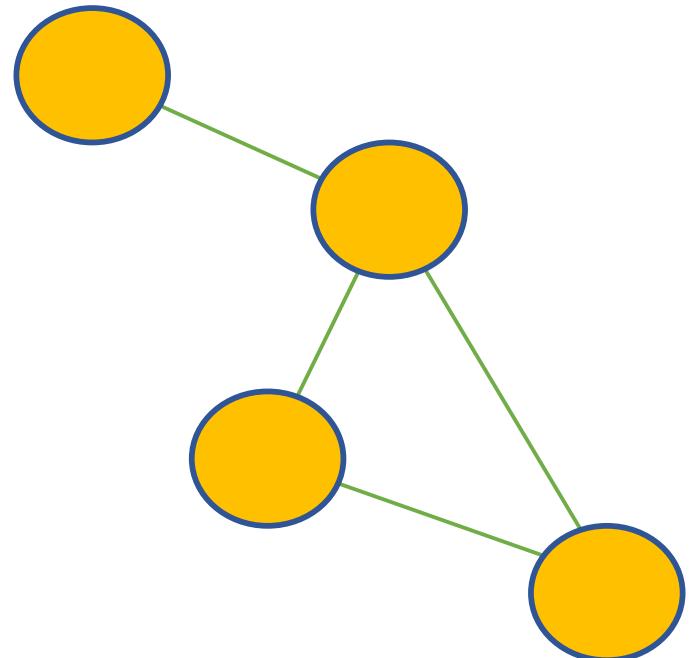
1.1 Sơ lược về đồ thị



What is graph?

In one restricted but very common sense of the term,^{[1][2]} a **graph** is an ordered pair $G = (V, E)$ comprising:

- V , a set of **vertices** (also called **nodes** or **points**);
- $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$, a set of **edges** (also called **links** or **lines**), which are **unordered pairs** of vertices (that is, an edge is associated with two distinct vertices).



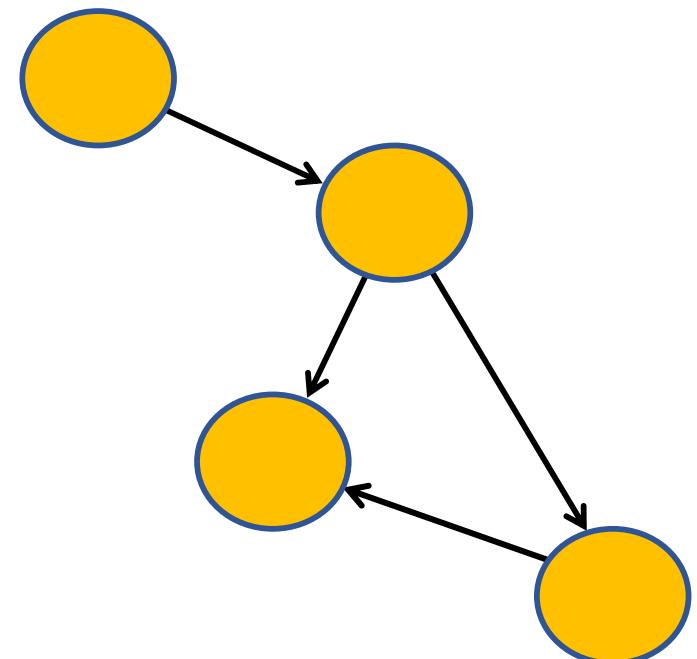
1.1 Sơ lược về đồ thị



What is graph?

In one restricted but very common sense of the term,^[5] a **directed graph** is an ordered pair $G = (V, E)$ comprising:

- V , a **set** of *vertices* (also called *nodes* or *points*);
- $E \subseteq \{(x, y) \mid (x, y) \in V^2 \text{ and } x \neq y\}$, a **set** of *edges* (also called *directed edges*, *directed links*, *directed lines*, *arrows* or *arcs*) which are **ordered pairs** of vertices (that is, an edge is associated with two distinct vertices).





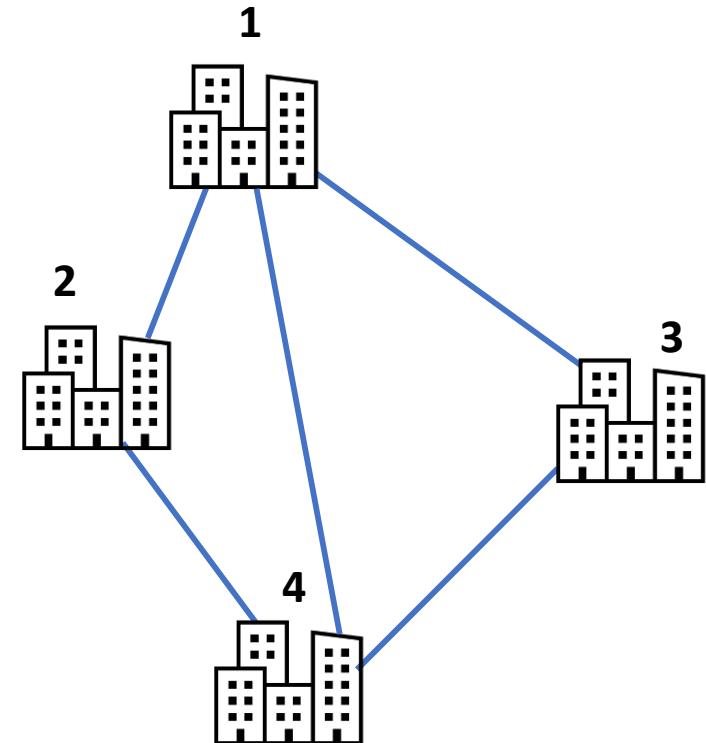
Bài toán thế nào được gọi là bài toán đồ thị?



Là bài toán có thể đưa về dạng $G = (V, E)$

VD1: Cho N thành phố, và M con đường hai chiều kết nối giữa 2 thành phố. Tìm đường đi từ thành phố 1 đến thành phố N sao cho đi qua ít thành phố nhất.

INPUT
4 5
1 2
2 4
3 4
3 1
4 1





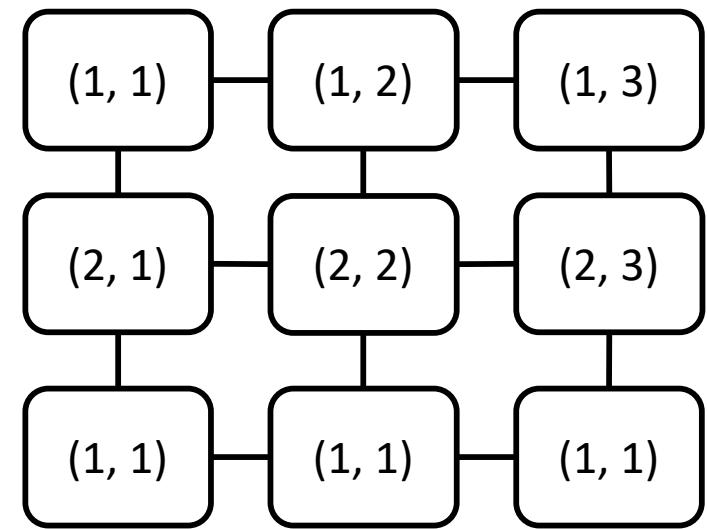
Bài toán thế nào được gọi là bài toán đồ thị?



Là bài toán có thể đưa về dạng $G = (V, E)$

VD2: Ma trận a kích thước $N * M$,
từ mỗi ô có thể đi 4 hướng là
xuống dưới, lên trên, qua trái, qua
phải. Chi phí để đi vào ô (u, v) là
 $a[u][v]$. Tìm chi phí nhỏ nhất đi từ
ô $(1, 1)$ đến ô (N, M)

INPUT
3 3
2 3 5
3 6 4
9 2 1





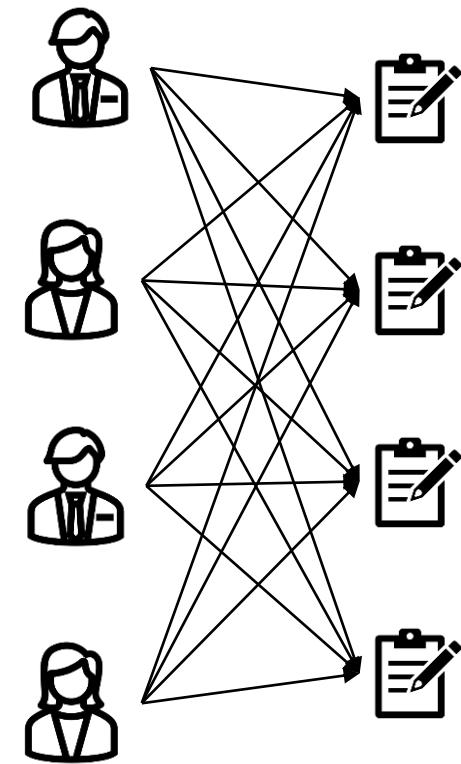
Bài toán thế nào được gọi là bài toán đồ thị?



Là bài toán có thể đưa về dạng $G = (V, E)$

VD3: Có N người, N việc. Người thứ i thực hiện công việc j mất $C[i, j]$ đơn vị thời gian. Giả sử tất cả bắt đầu vào thời điểm 0, hãy tìm cách bố trí mỗi công việc cho mỗi người sao cho thời điểm hoàn thành công việc là sớm nhất có thể.

INPUT
4
10 10 10 2
10 10 3 10
4 10 10 10
10 5 10 10





Bài toán thế nào được gọi là bài toán đồ thị?



Là bài toán có thể đưa về dạng $G = (V, E)$

VD4: Cho 2 số nguyên dương X và Y,
tìm số nguyên dương N sao cho:

$$N \bmod X = Y \bmod N$$

INPUT
69420 42068





Template của đồ thị

```
struct Graph{
    int n; // Số đỉnh của đồ thị n = |V|
    int m; // Số cạnh của đồ thị m = |E|
    vector<vector<int>> a; // Lưu đồ thị theo danh sách kề

    // Gán số đỉnh của đồ thị
    Graph(int n) {
        ...
    }

    // Hàm thêm cạnh vào đồ thị
    void addEdge(...){
        // Mỗi một đồ thị thì có thể có tính chất khác nhau
        // nên cách thêm sẽ khác nhau
        ...
    }

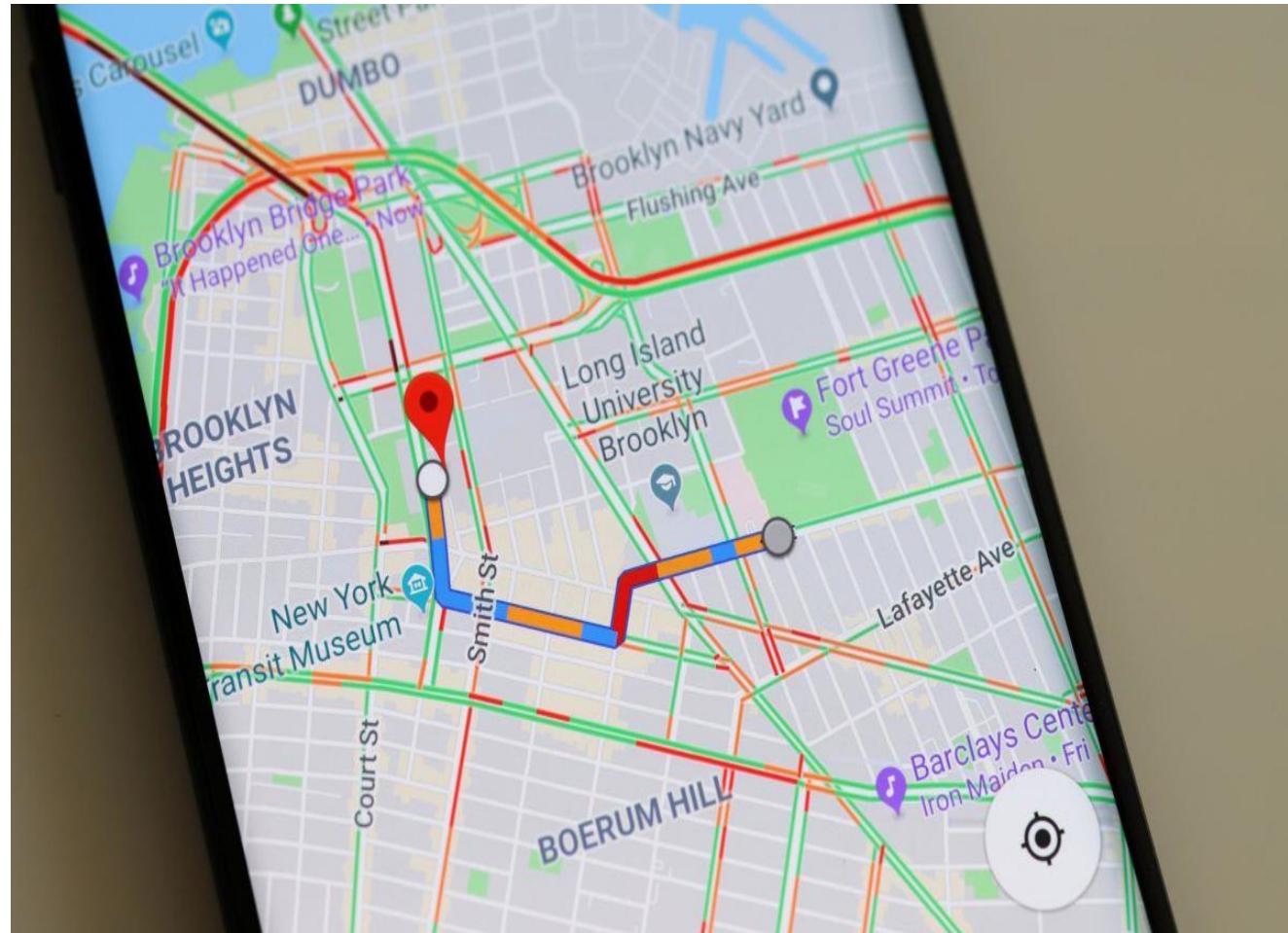
    // Thuật toán đồ thị nào đó
    void algoA(...){
        ...
    }
    void algoB(...){...}
    ...
};
```

1.2 Một số ứng dụng

Shortest paths in Google Maps

V: Các thành phố

E: Các tuyến đường



Social Networking (Facebook, LinkedIn, Instagram...)

V: People

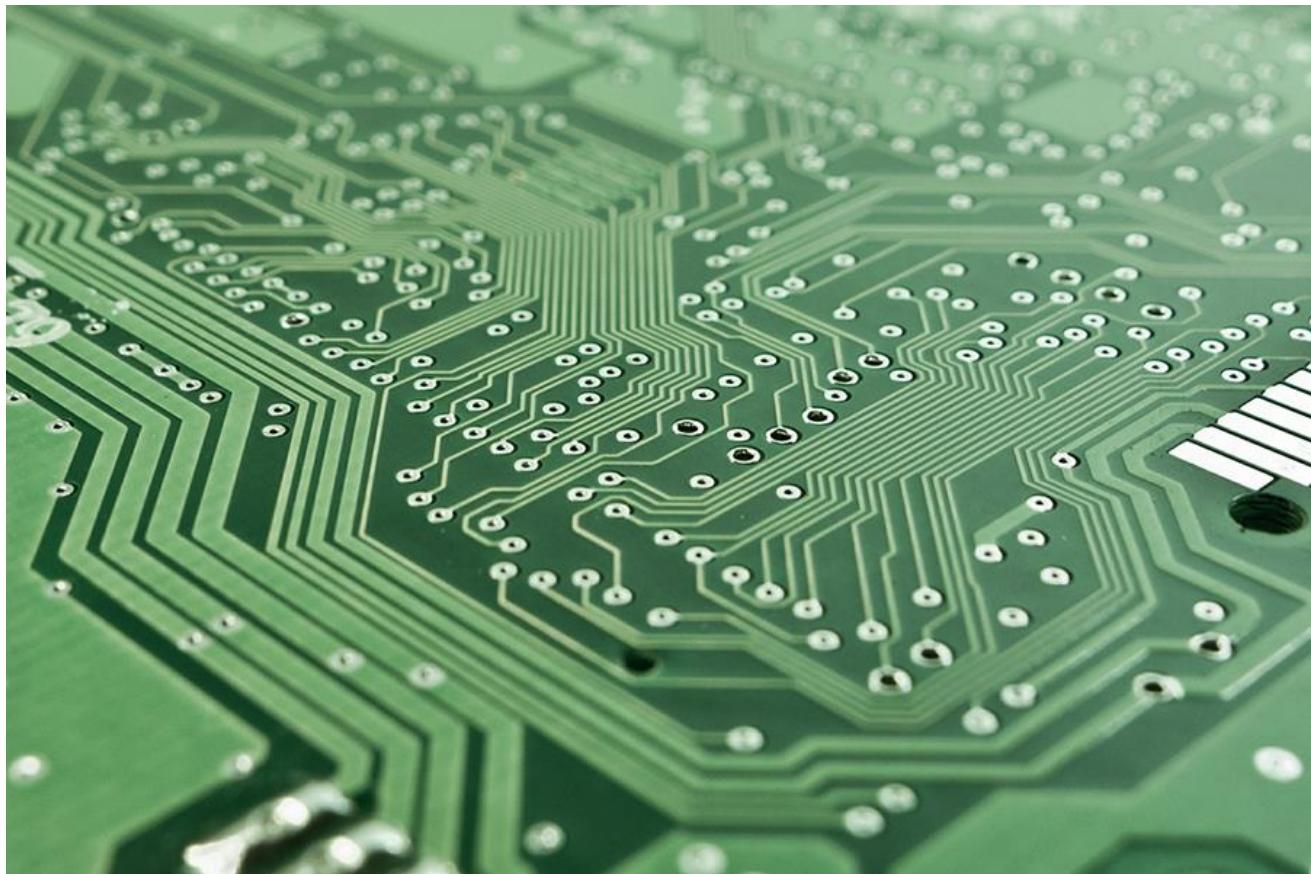
E: New feed, find mutual friend, ...



Circuit Design

V: Resistor

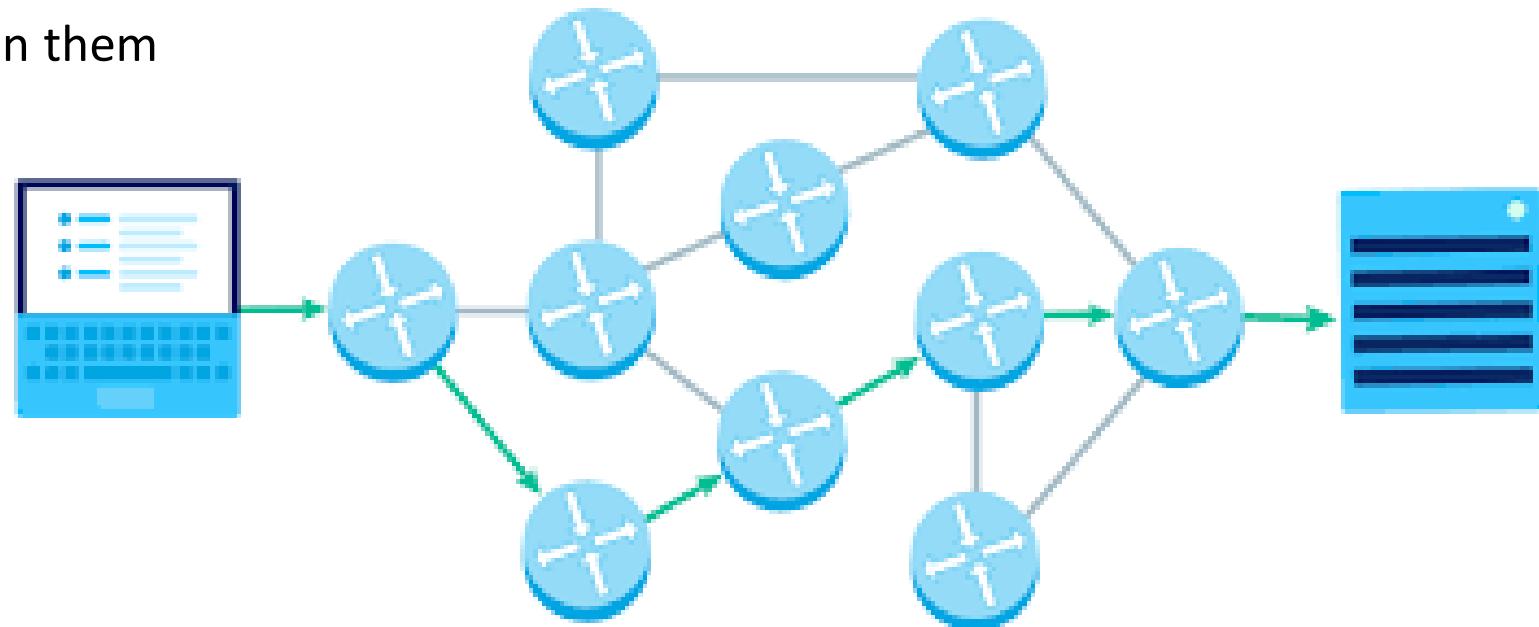
E: Wire



Routing Algorithms

V: Router

E: Connection between them



Graph in deep learning

V: Neurons

E: Connection

between them

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

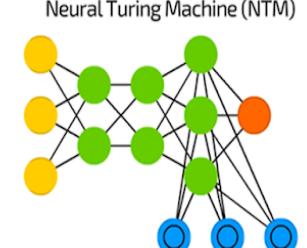
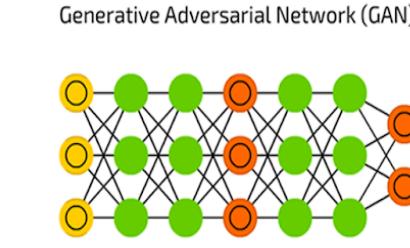
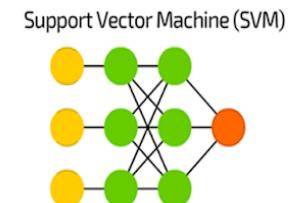
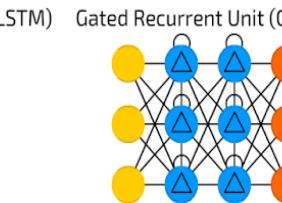
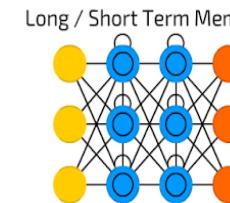
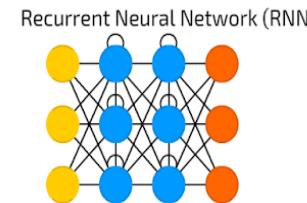
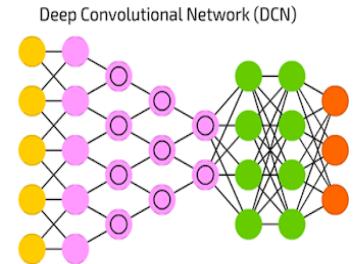
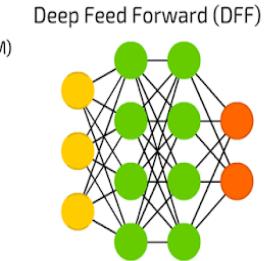
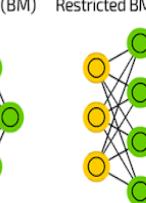
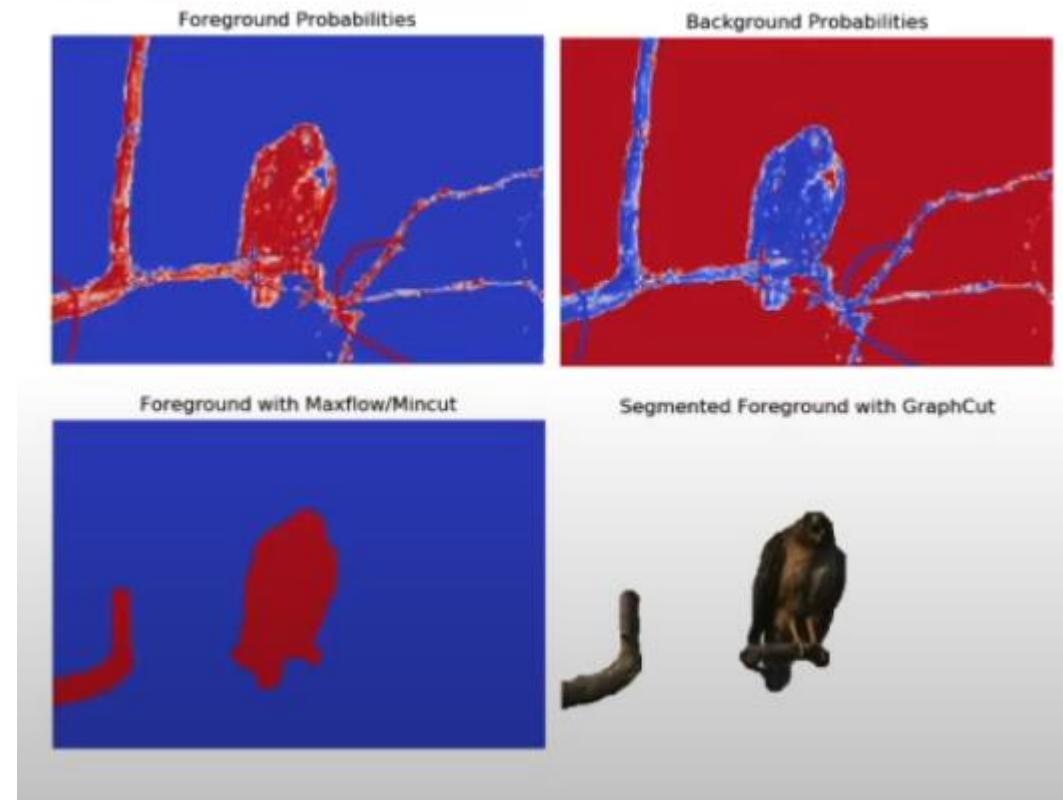


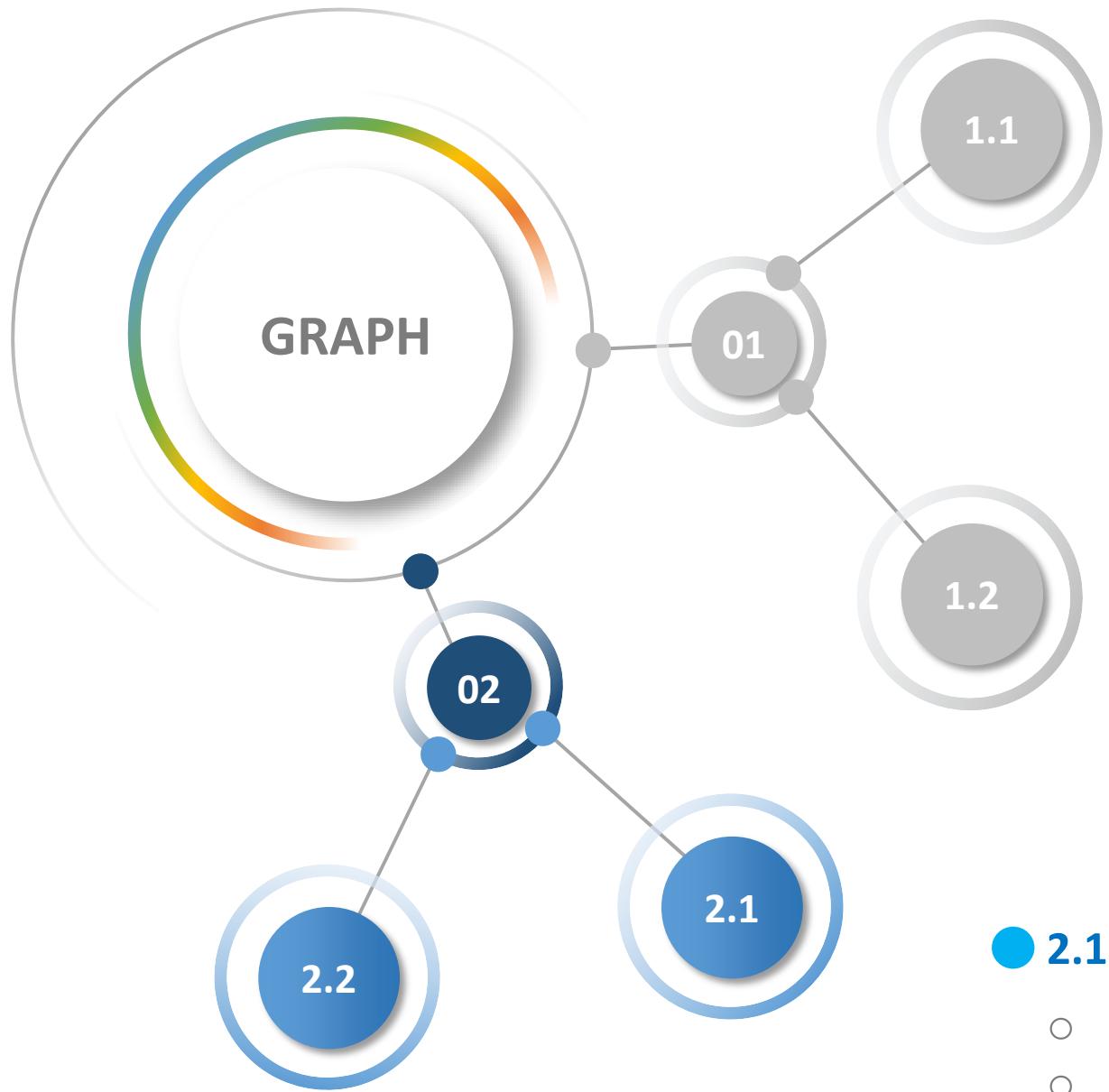
Image processing & segmentation

V: Pixels

E: Connections between

Neighbouring pixels





● 01. Giới thiệu về đồ thị

● 1.1 Sơ lược về đồ thị

- Đồ thị là gì?
- Thế nào là bài toán đồ thị?
- ...

● 1.2 Một số ứng dụng

- Google map
- Mạng máy tính
- Social networking
- ...

● 02. Một vài thuật toán và ứng dụng của nó

● 2.1 Đường đi ngắn nhất

- Dijkstra
- Floyd
- A*

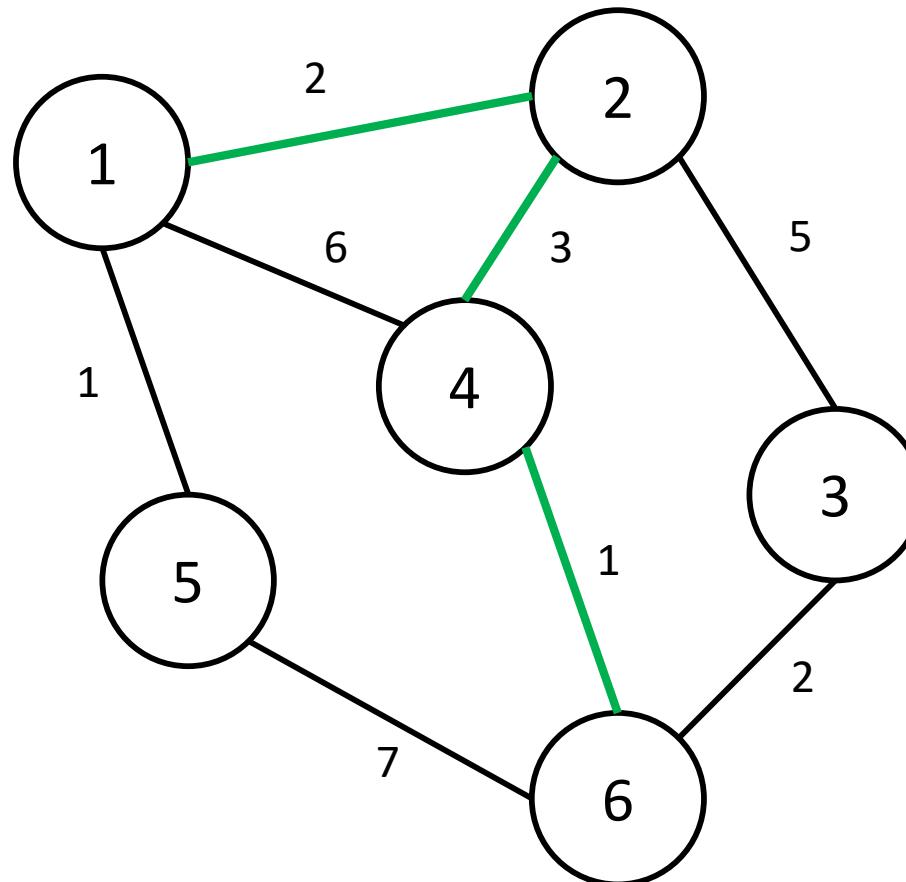
● 2.2 Luồng cực đại

Ford-Fullerson

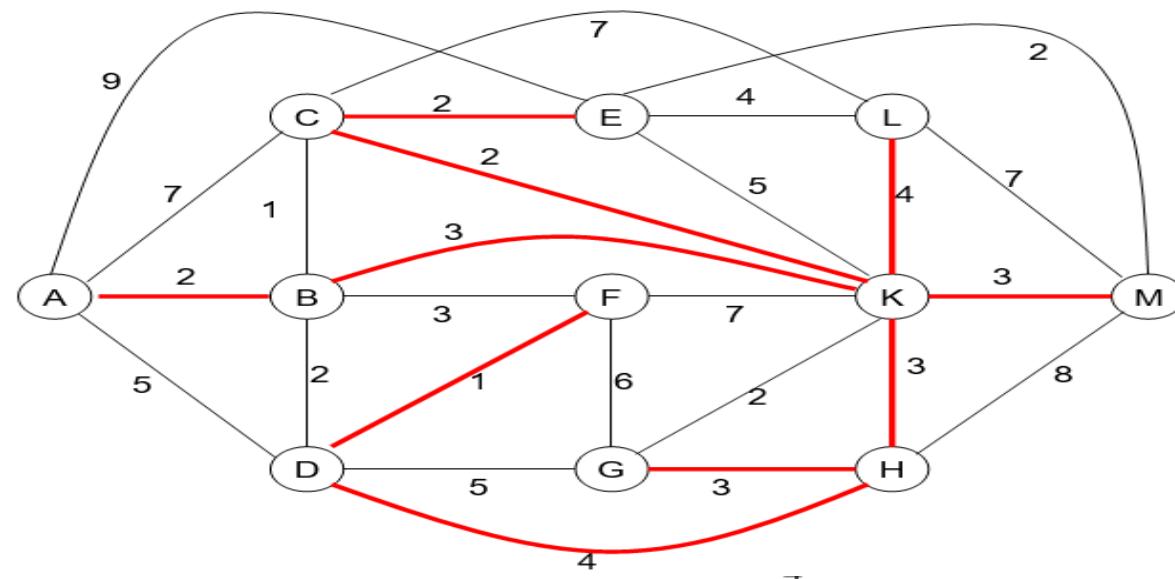
02. Một vài thuật toán và ứng dụng của nó

2.1 Đường đi ngắn nhất

- Dijkstra
- Bellman – Ford
- Floyd-Warshall
- A*
- BFS
- ...



Thuật toán Dijkstra



```
struct Graph{
    int n;
    vector<int> d;
    vector<vector<pair<int, int>>> a;
};

Graph(int n){
    this->n = n; a = vector<vector<pair<int, int>>>(n + 1);
}

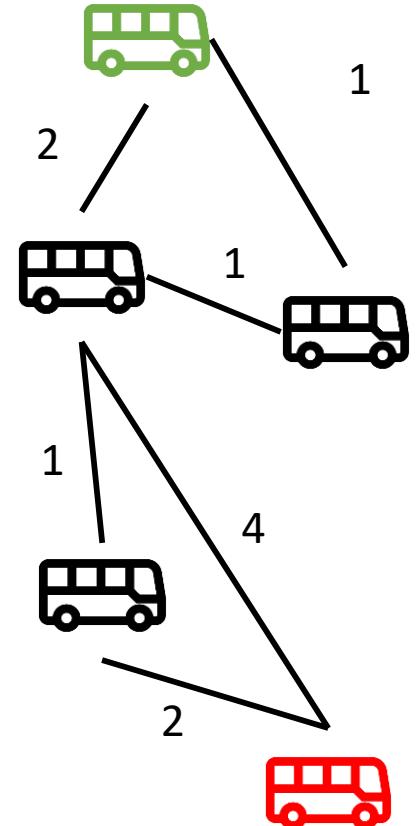
void addEdge(int u, int v, int w){
    a[u].push_back(make_pair(w, v));
    a[v].push_back(make_pair(w, u));
}

void dijkstra(int s, int t){
    d = vector<int>(n + 1, 1e9); d[s] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> heap;
    heap.push(make_pair(0, s));
    while(heap.size()) {
        int du = heap.top().first, u = heap.top().second; heap.pop();
        for(auto [uv, v]: a[u]) {
            if(d[v] > du + uv) {
                d[v] = du + uv;
                heap.push(make_pair(d[v], v));
            }
        }
    }
    return d[t];
};
```

Dijkstra (Code C++)
ĐPT: $O(M \log N)$

Một vài bài tập về Dijkstra

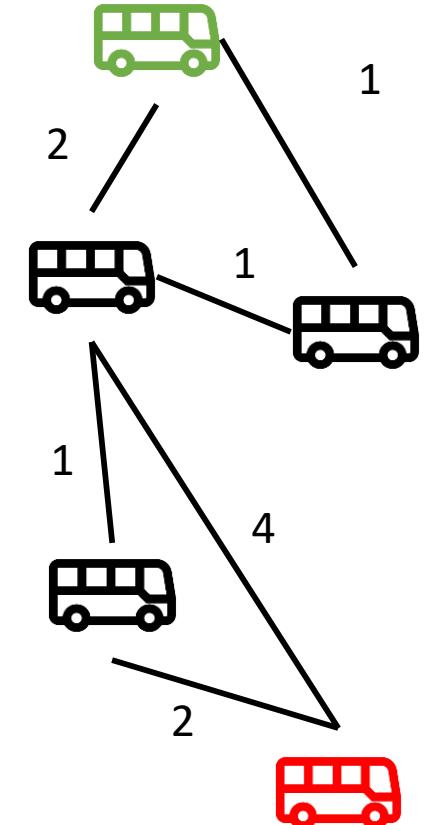
BT1: Thành phố của Nam sống có N trạm xe bus, trường của Nam thì ở ngay trạm thứ N. Nam thường đi học bắt đầu ở trạm thứ 1. Hôm nay Nam và Hoàng cùng nhau lộ trình của các xe bus và biết được các trạm xe bus đi đến được với nhau và giá tiền để đi lại. Sau một hồi hì hục thì cả hai đã tìm ra được một lộ trình rẻ nhất giữa nhà Nam đi từ trạm xe bus 1 đến trường ở trạm xe bus N, nhưng cả hai vẫn còn một thắc mắc đó là có bao nhiêu lộ trình rẻ nhất như thế, mọi người hãy cùng giúp Nam và Hoàng nhé?



Một vài bài tập về Dijkstra

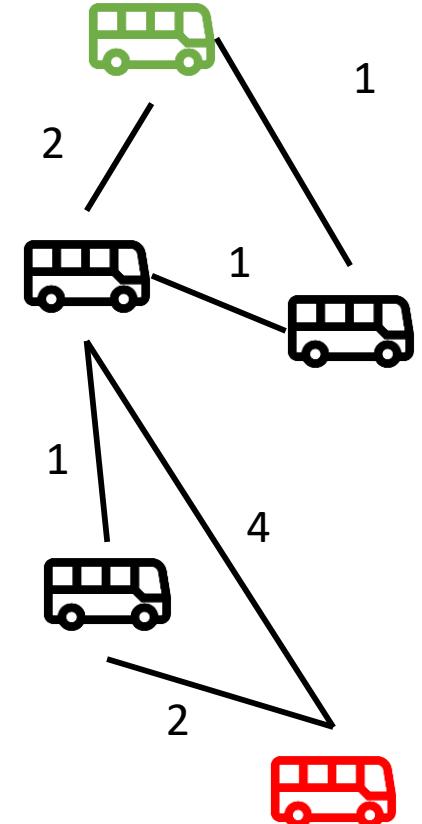
Giải bài tập 1:

- Gọi $d[u]$ là đường đi rẻ nhất từ 1 tới u , ban đầu $d[1] = 0$, $d[i] = \text{INF}$ ($i \neq 1$)
- Gọi $f[u]$ là số đường đi rẻ nhất từ 1 tới u , ban đầu $f[1] = 1$, $f[i] = 0$ ($i \neq 1$)
- Nếu $d[v] = d[u] + uv$ thì $f[v] += f[u]$
- Ngược lại if($d[v] > d[u] + uv$) thì $d[v] = d[u] + uv$; $f[v] = f[u]$



Một vài bài tập về Dijkstra

BT2: Vẫn là bài toán xoanh quanh các trạm xe bus ở thành phố mà Nam sinh sống. Sau khi được các bạn giúp đếm số đường đi ngắn nhất thì Nam rất vui và đi nói cho các bạn của mình nghe, nhiều bạn thấy thú vị nên một số bạn đã tặng cho Nam vé xe bus miễn phí, Nam đếm tổng cộng thì được k vé. Giờ Nam có một thắc mắc mới là với k vé xe bus miễn phí này thì với lộ trình tối ưu nhất số tiền Nam bỏ ra ít nhất là bao nhiêu để đi từ nhà đến trường.



Một vài bài tập về Dijkstra

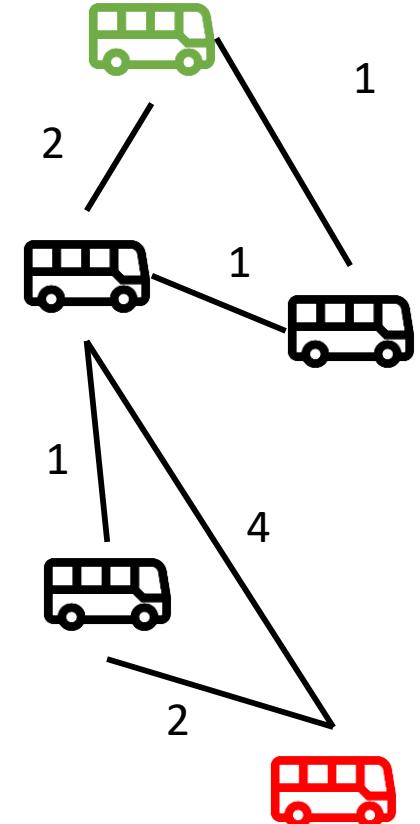
Giải bài tập 2:

Gọi $d[u][x]$ là đường đi rẻ nhất từ 1 tới u sử dụng x vé miễn phí

Ta có nếu $d[v][x] > d[u][x] + uv$ thì $d[v][x] = d[u][x] + uv$

Nếu $x > 0$ thì ta có thêm:

Nếu $d[v][x] > d[u][x - 1]$ thì $d[v][x] = d[u][x - 1]$

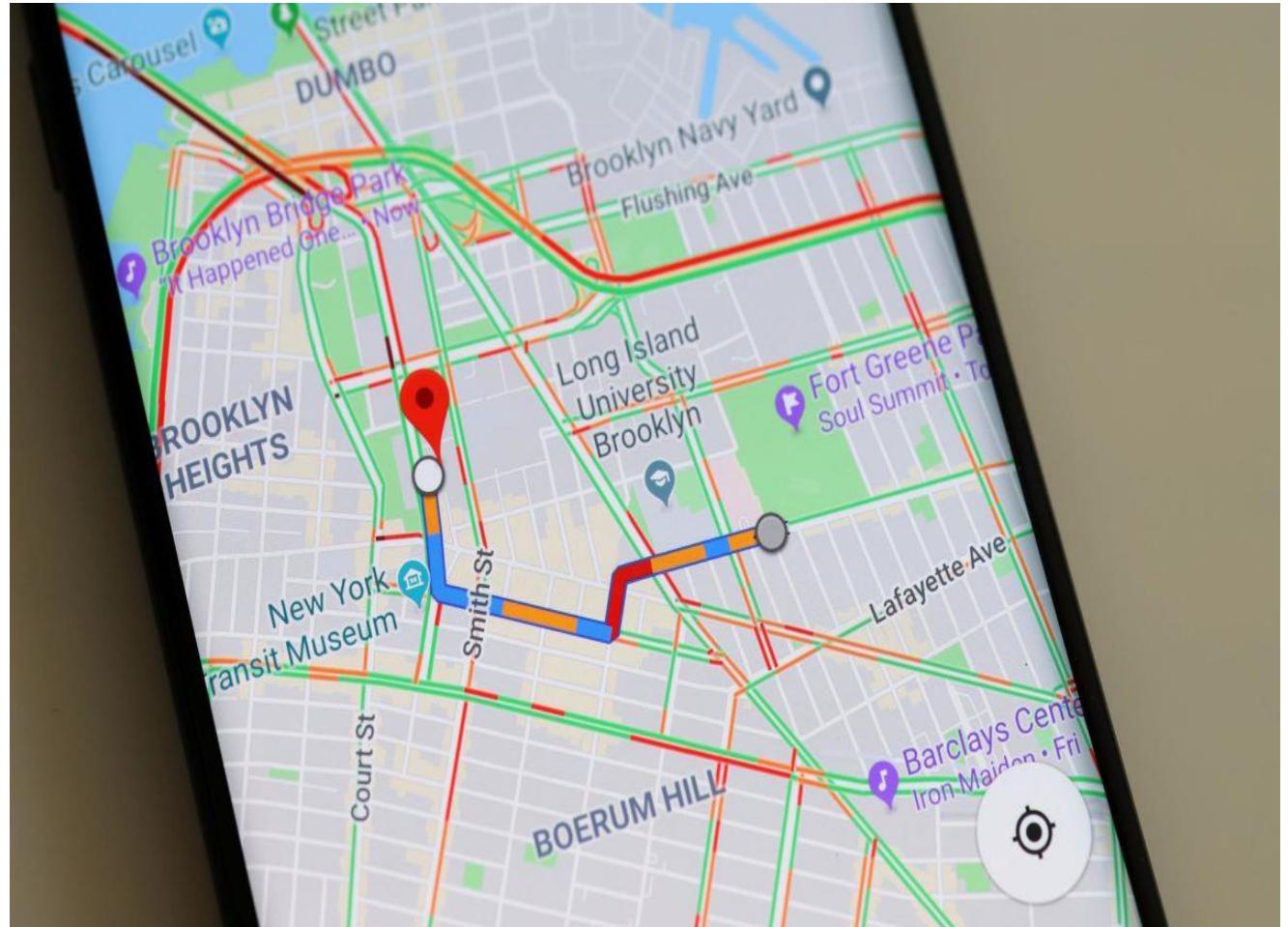


Một vài ứng dụng của Dijkstra trong thực tế

Một trong những thuật chính của Google Maps

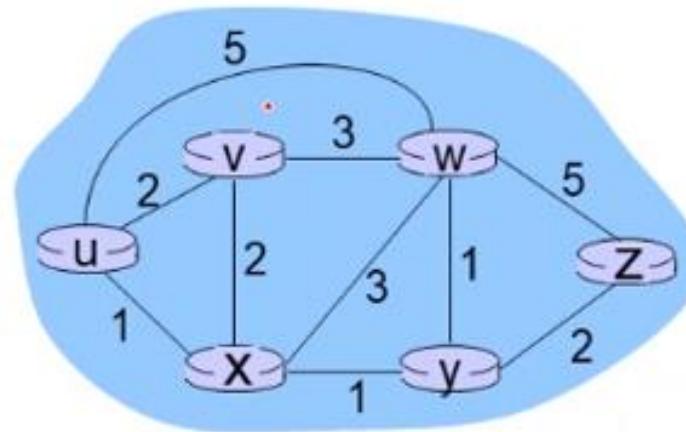
V: Các thành phố

E: Các tuyến đường



Một vài ứng dụng của Dijkstra trong thực tế

Tìm đường đi giữa các Router:



Đồ thị: $G = (N, E)$

N = tập hợp các router = { u, v, w, x, y, z }

E = tập hợp các kết nối ={ $(u,v), (u,x), (u,w), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$ }

Ghi chú: mô hình đồ thị cũng hữu ích trong các ngũ cảnh khác.

Ví dụ, P2P, trong đó N là tập các peer và E là tập các kết nối TCP

2.2 Luồng cực đại

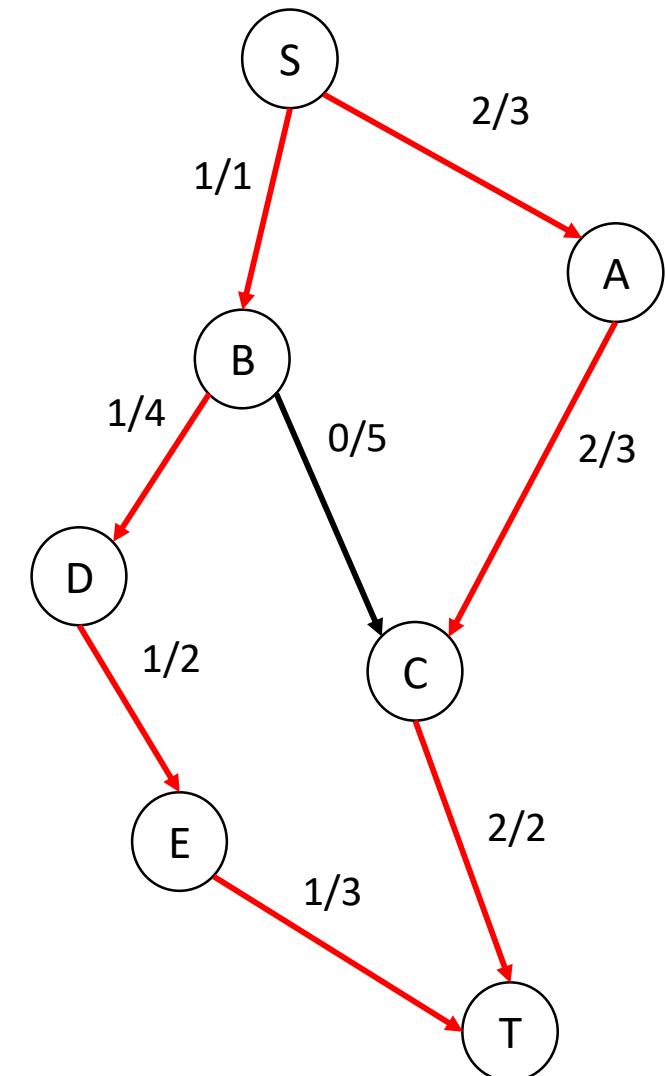
- ❖ Mạng luồng (Flow network)
- ❖ Luồng hợp lệ (Admissible flows)
- ❖ Mạng thặng dư (Residual network)
- ❖ Đường tăng luồng (Augment path)
- ❖ Bài toán luồng cực đại trên mạng (Maximum flow)
- ❖ Thuật toán tìm luồng cực đại
- ❖ Một vài ứng dụng trong thực tế (Real world applications)

Mạng luồng (Flow network)

Mạng luồng là một đồ thị có hướng thỏa mãn:

- Có 2 đỉnh đặc biệt là **đỉnh phát** (source-ký hiệu s) và **đỉnh thu** (sink-ký hiệu t)
- Không có cung đi vào đỉnh phát hay ra từ đỉnh thu
- Với mỗi cung (u, v) gọi một số thực dương $c(u, v)$ là sức chứa

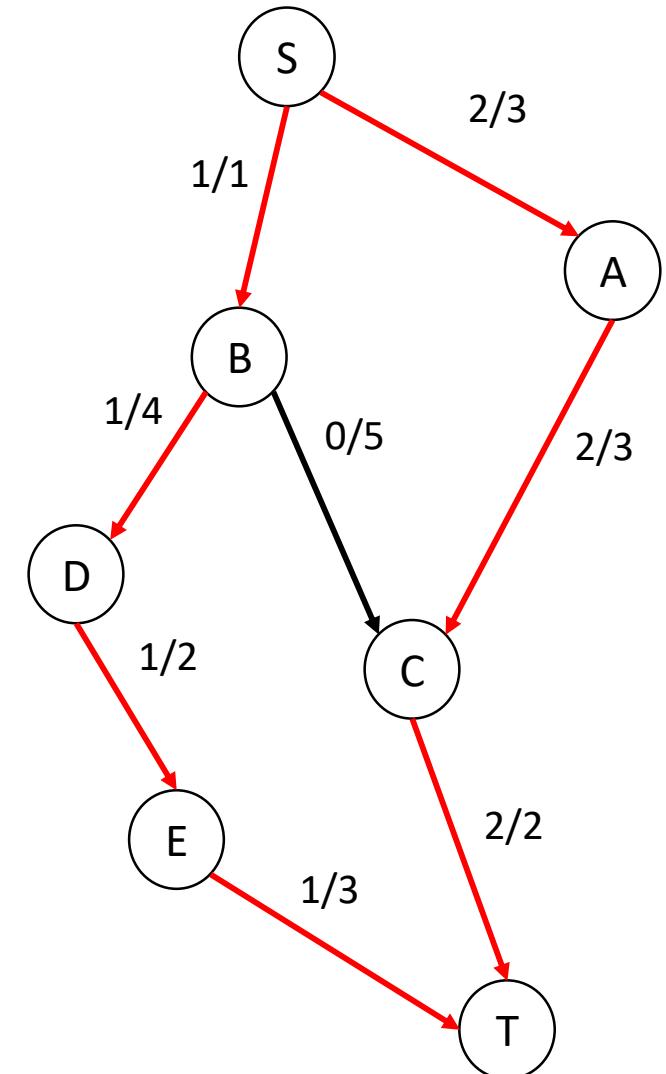
Ta biểu diễn mạng luồng bằng $G = (V, s, t, c)$ với $s, t \in V$,
 $c : V \times V \rightarrow [0, \infty)$ và $\forall v, c(v, s) = c(t, v) = 0$. Tồn tại cung (u, v) khi và chỉ khi $c(u, v) > 0$.



Luồng hợp lệ (Admissible flows)

Một luồng hợp lệ trên mạng G là một hàm $f : V \times V \rightarrow \mathbb{R}$ thỏa:

- Luồng trên mỗi cung không vượt quá sức chứa của nó:
 $\forall u, v \in V, f(u, v) \leq c(u, v)$
- Đối xứng: $\forall u, v \in V, f(u, v) = -f(v, u)$
- Bảo toàn luồng qua mỗi đỉnh: $\forall u \in V, \sum_{v \in V} f(u, v) = 0$

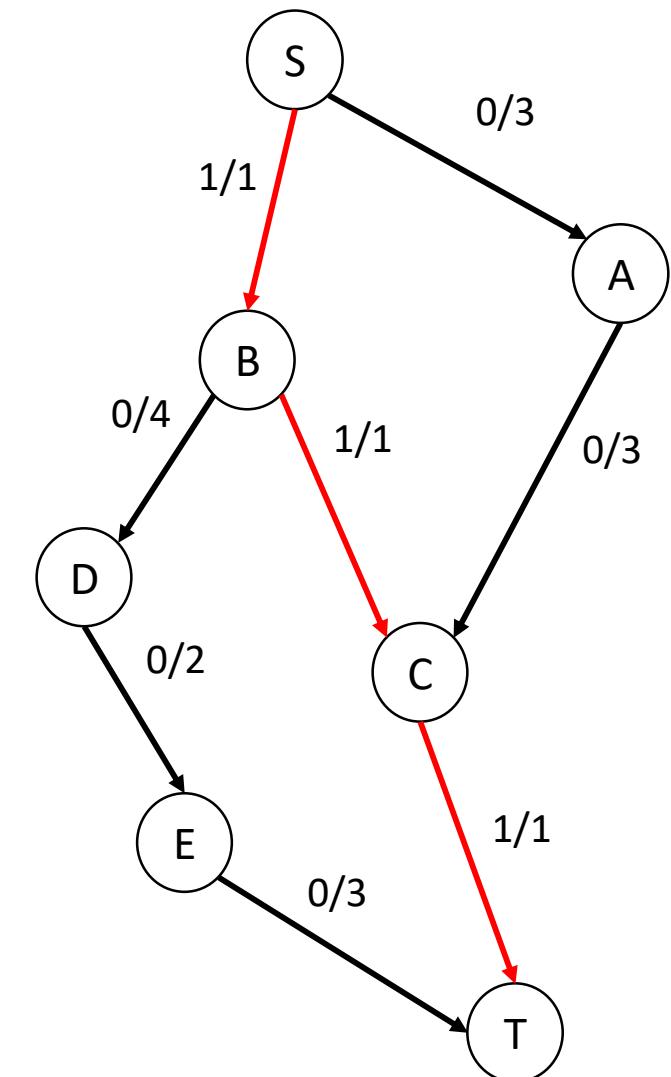


Mạng thặng dư (Residual network)

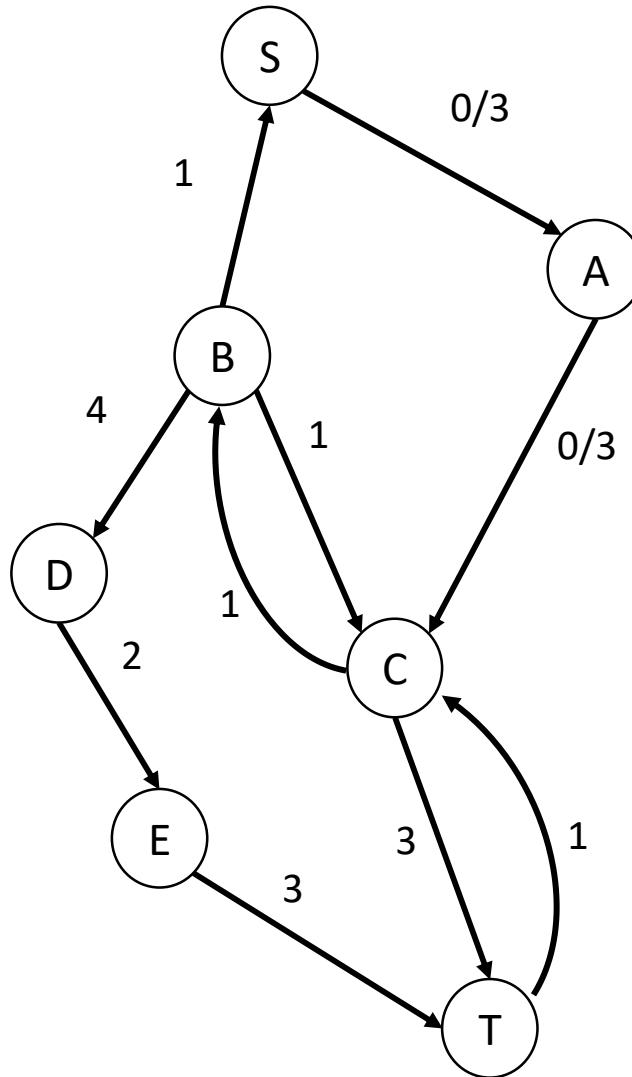
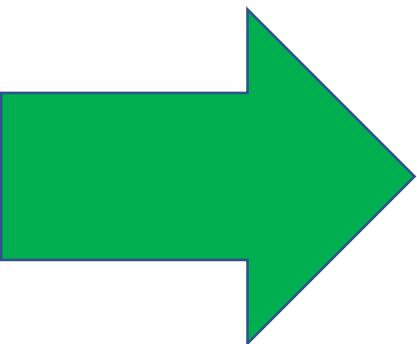
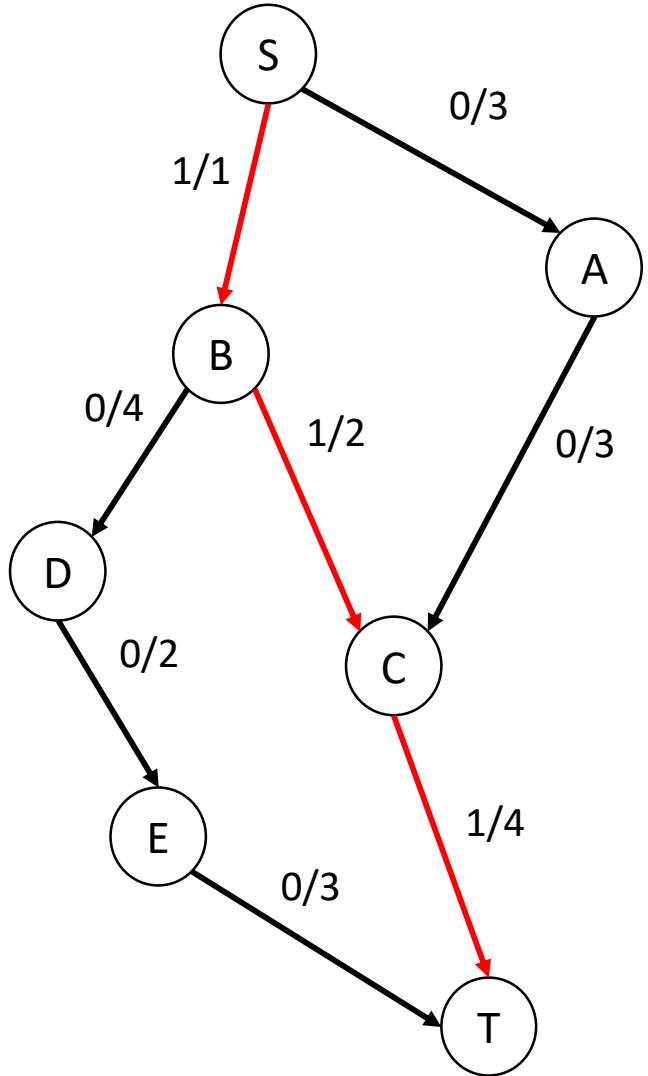
Mạng thặng dư cho chúng ta biết sức chứa còn lại trên mạng khi đã gửi một số luồng qua nó.

Từ một mạng (V, s, t, c) và một luồng f ta xây dựng một mạng thặng dư (V, s, t, r) với $r(u, v) = c(u, v) - f(u, v)$.

Mạng thặng dư có thể chứa một số cung mà mạng ban đầu không có. Cụ thể, khi gửi một luồng $f(u, v) > 0$ qua cung (u, v) thì ta cũng gửi một luồng $-f(u, v)$ qua cung (v, u) , khi đó $r(v, u) = c(v, u) + f(u, v) > 0$ có thể tạo ra cung (v, u) mà mạng ban đầu chưa có.

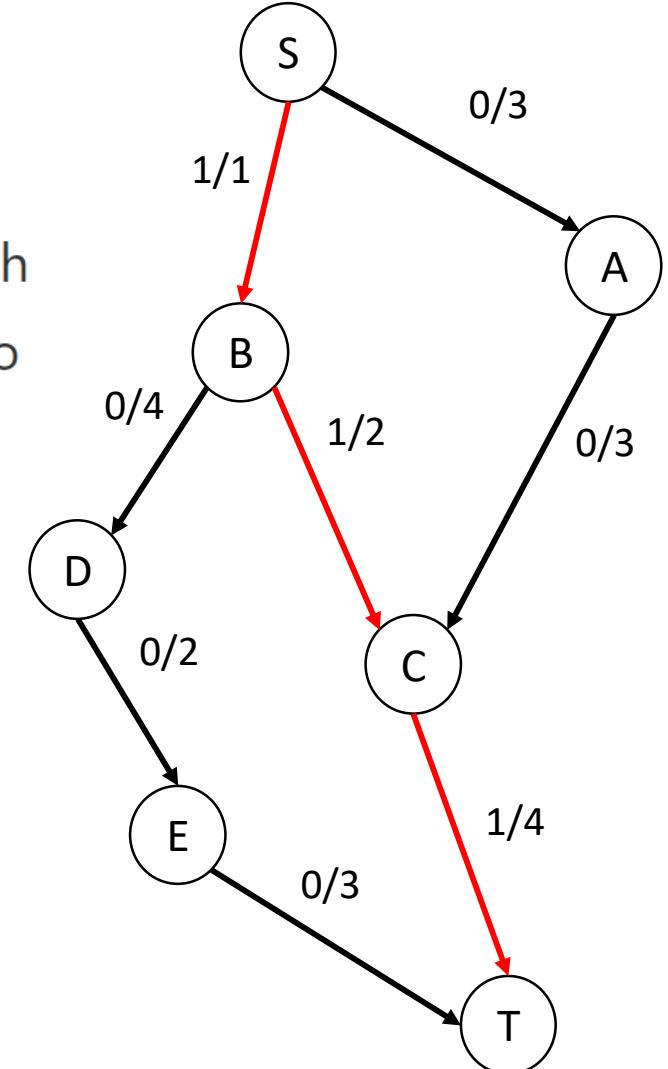


Mạng thặng dư (Residual network)



Đường tăng luồng (Augment path)

Đường tăng luồng là một đường đi đơn từ đỉnh phát s (source) đến đỉnh thu t (sink) trong mạng thặng dư G' mà kenh trên đường đi chưa bị bão hòa ($f'[u, v] < c'[u, v]$, một kenh $e'(u, v)$ được gọi là bão hòa nếu $f'(u, v) = c'(u, v)$).

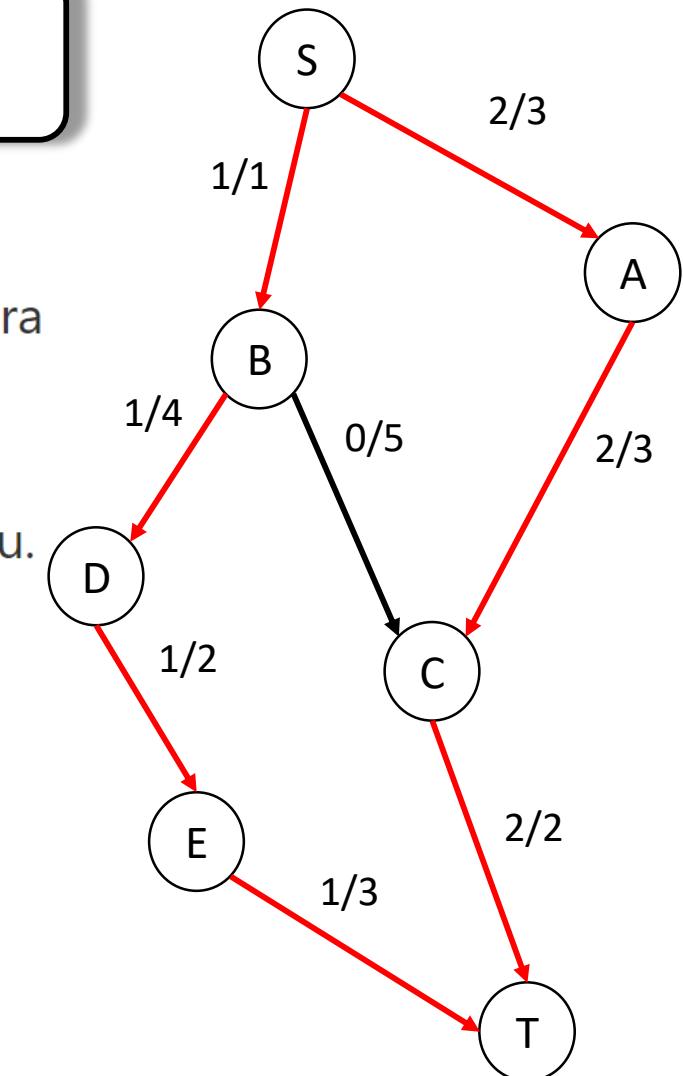


Bài toán luồng cực đại trên mạng (Maximum flow)

Cho một mạng, chúng ta cần tìm một luồng hợp lệ sao cho tổng luồng ra từ đỉnh phát cũng như đi vào đỉnh thu là cực đại.

Mệnh đề: Tổng luồng ra khỏi đỉnh phát bằng tổng luồng đi vào đỉnh thu.

Chứng minh: Xét luồng f trên mạng G



Bài toán luồng cực đại trên mạng (Maximum flow)

Ta có:

$$\begin{aligned} & \sum_{u \in V} \sum_{v \in V} f(u, v) \\ &= \sum_{u \in V} \sum_{v \in V} \frac{1}{2}(f(u, v) + f(v, u)) \\ &= \sum_{u \in V} \sum_{v \in V} \frac{1}{2}(f(u, v) - f(v, u)) \\ &= \frac{1}{2} \sum_{u \in V} \sum_{v \in V} f(u, v) - \frac{1}{2} \sum_{u \in V} \sum_{v \in V} f(v, u) \\ &= \frac{1}{2} \sum_{u \in V} \sum_{v \in V} f(u, v) - \frac{1}{2} \sum_{u \in V} \sum_{v \in V} f(u, v) \\ &= 0 \end{aligned}$$

Lại có:

$$\begin{aligned} & \sum_{u \in V} \sum_{v \in V} f(u, v) \\ &= \sum_{u \in V \setminus \{s, t\}} \sum_{v \in V} f(u, v) + \sum_{v \in V} f(s, v) + \sum_{v \in V} f(v, t) \\ &= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, t) \\ &\text{Vậy: } \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t) \end{aligned}$$

Thuật toán tìm luồng cực đại

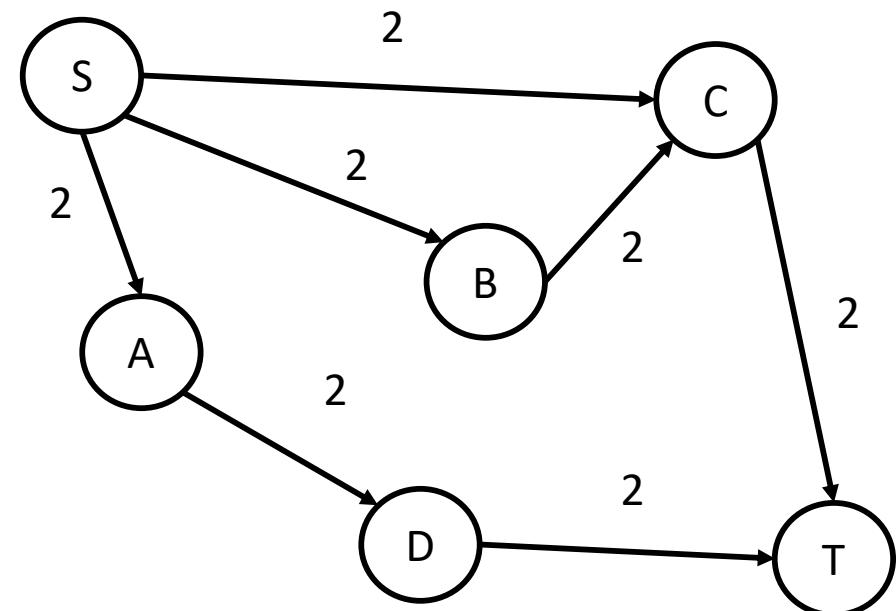
Thuật toán Ford-Fullerson:

bước(1): Tạo mạng thặng dư G' tương ứng cho mạng G ban đầu

bước(2): tìm một đường tăng luồng trên mạng thặng dư G'

- nếu không tồn tại đường tăng luồng \rightarrow kết thúc thuật toán
- nếu tồn một đường tăng luồng \rightarrow thực hiện tăng luồng trên mạng thặng dư và quay trở lại bước(2)
Khi thuật toán kết thúc $f(s, V')$ chính là giá trị luồng cực đại cần tìm.

Đến đây bạn đã có thể dùng thuật toán tìm kiếm trên đồ thị DFS (deep first search) hoặc BFS(breadth first search) để tìm đường tăng luồng và cập nhật mạng thặng dư thuật toán này có độ phức tạp bằng số lần tăng luồng (f^*) nhân với độ phức tạp của thuật toán tìm kiếm đồ thị- $O(E)$ và bằng $O(|f^*| \cdot E)$. Sau đây là code của thuật toán trên:



```
struct Ford_Fulkerson{
    struct Edge{
        int u, v, cap, flow;
        Edge() {}
        Edge(int u, int v, int cap, int flow): u(u), v(v), cap(cap), flow(flow) {}
    };

    int n;
    vector<int> trace, d;
    vector<vector<int>> a;
    vector<Edge> e;

    Ford_Fulkerson() {}

    Ford_Fulkerson(int n): n(n), d(n + 5), trace(n + 5), a(n + 5) {}

    void addEdge(int u, int v, int cap){
        a[u].push_back(e.size()); e.push_back(Edge(u, v, cap, 0));
        a[v].push_back(e.size()); e.push_back(Edge(v, u, 0, 0));
    }

    int maxFlow(int s, int t) {
        int flow = 0;
        while (true) {
            vector<int> parent(n + 5);
            parent[s] = -1;
            queue<int> q;
            q.push(s);
            while (!q.empty()) {
                int cur = q.front();
                q.pop();
                for (int i = 0; i < a[cur].size(); ++i) {
                    int next = e[a[cur][i]].v;
                    if (parent[next] == -1 && e[a[cur][i]].cap > e[a[cur][i]].flow) {
                        parent[next] = cur;
                        q.push(next);
                    }
                }
            }
            if (parent[t] == -1) break;
            int minCap = INT_MAX;
            for (int i = 0; i < a[t].size(); ++i) {
                if (parent[e[a[t][i]].v] == t && e[a[t][i]].cap > e[a[t][i]].flow) {
                    minCap = min(minCap, e[a[t][i]].cap - e[a[t][i]].flow);
                }
            }
            for (int i = 0; i < a[t].size(); ++i) {
                if (parent[e[a[t][i]].v] == t && e[a[t][i]].cap > e[a[t][i]].flow) {
                    e[a[t][i]].flow += minCap;
                    e[a[e[a[t][i]].v][a[t][i]]].flow -= minCap;
                }
            }
            flow += minCap;
        }
        return flow;
    }
};
```

```
bool bfs(int s, int t){  
    for(int i = 1; i <= n; i++) d[i] = 0;  
    queue<int> q;  
    q.push(s); d[s] = 1;  
    while(q.size() && !d[t]) {  
        int u = q.front();  
        q.pop();  
        for(auto i: a[u]) {  
            int v = e[i].v;  
            if(d[v]) continue;  
            if(e[i].cap - e[i].flow <= 0) continue;  
            d[v] = 1;  
            trace[v] = i;  
            q.push(v);  
        }  
    }  
    return d[t];  
}  
  
int maxFlow() {  
    int flow = 0;  
    while(bfs(s, t)) {  
        int u = t;  
        int minCap = INT_MAX;  
        while(u != s) {  
            int v = trace[u];  
            minCap = min(minCap, e[v].cap - e[v].flow);  
            u = v;  
        }  
        for(int i = 0; i < adj[s].size(); i++) {  
            int v = adj[s][i];  
            if(e[i].v == t) {  
                e[i].flow += minCap;  
                e[i ^ 1].flow -= minCap;  
            }  
        }  
        flow += minCap;  
    }  
    cout << g.maxFlow(s, t);  
}
```

```
int maxFlow(int s, int t){  
    int flow = 0;  
    while(bfs(s, t)){  
        int ans = 1e9;  
        for(int v = t; v != s; v = e[trace[v]].u){  
            int val = e[trace[v]].cap - e[trace[v]].flow;  
            ans = min(ans, val);  
        }  
        for(int v = t; v != s; v = e[trace[v]].u){  
            e[trace[v]].flow += ans;  
            e[trace[v] ^ 1].flow -= ans;  
        }  
        flow += ans;  
    }  
    return flow;  
}
```

```
se); cin.tie(0);
```

```
g.addEdge(u, v, c);  
g.addEdge(v, u, c);  
}  
cout << g.maxFlow(s, t);  
}
```

```
int32_t main()
{
    ios_base::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m >> s >> t;
    g = Ford_Fulkerson(n);
    for(int i = 1; i <= m; i++)
    {
        cin >> u >> v >> c;
        g.addEdge(u, v, c);
        g.addEdge(v, u, c);
    }
    cout << g.maxFlow(s, t);
}
```

Method	Complexity
Linear programming	
Ford–Fulkerson algorithm	$O(E f_{max})$
Edmonds–Karp algorithm	$O(VE^2)$
Dinic's algorithm	$O(V^2E)$
MKM (Malhotra, Kumar, Maheshwari) algorithm ^[10]	$O(V^3)$
Dinic's algorithm with dynamic trees	$O(VE \log V)$
General push–relabel algorithm ^[11]	$O(V^2E)$

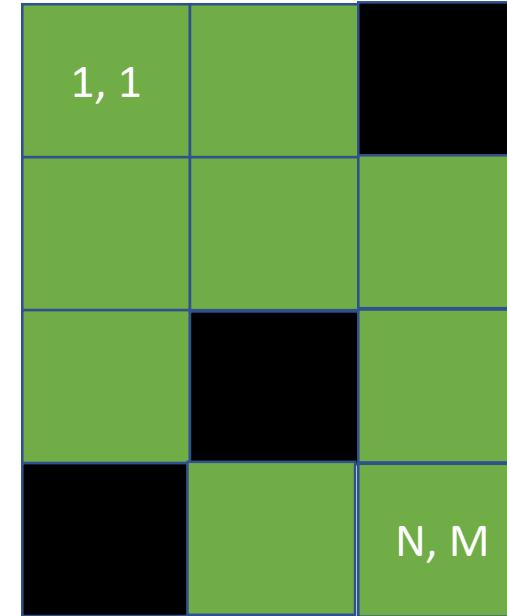
Method	Complexity
Push–relabel algorithm with maximum distance vertex selection rule ^[12]	$O(V^2\sqrt{E})$
Push–relabel algorithm with dynamic trees ^[11]	$O\left(VE \log \frac{V^2}{E}\right)$
KRT (King, Rao, Tarjan)'s algorithm ^[13]	$O\left(VE \log \frac{E}{V \log V} V\right)$
Binary blocking flow algorithm ^[14]	$O\left(E \cdot \min\{V^{2/3}, E^{1/2}\} \cdot \log \frac{V^2}{E} \cdot \log U\right)$
James B Orlin's + KRT (King, Rao, Tarjan)'s algorithm ^[9]	$O(VE)$
Kathuria-Liu-Sidford algorithm ^[15]	$E^{4/3+o(1)}U^{1/3}$
BLNPSSSW / BLLSSSW algorithm ^[17] ^[18]	$\tilde{O}((E + V^{3/2}) \log U)$
Gao-Liu-Peng algorithm ^[19]	$\tilde{O}(E^{\frac{3}{2}-\frac{1}{328}} \log U)$

Bài tập vận dụng

BT1: Có N thành phố được kết nối với nhau bằng M con đường 2 chiều, con đường thứ i ($1 \leq i \leq M$) kết nối thành phố u và thành phố v có lưu lượng xe có thể lưu thông nhiều nhất tại một thời điểm là w. Lưu lượng giao thông nhiều nhất mà đi từ thành phố 1 đến thành phố N nhiều nhất là bao nhiêu?

Bài tập vận dụng

BT2: Cho ma trận A kích thước $N * M$ ($N, M \leq 1000$), $A[i][j]$ là giá trị của hàng thứ i và cột thứ j . Nếu vị trí (i, j) rỗng thì $A[i][j] = 0$, ngược lại $A[i][j] = 1$ thì vị trí (i, j) có đá. Được quyền đi sang phải hoặc xuống dưới nếu ô đi đến là ô rỗng, khi đi qua ô rỗng thì ô đó bị biến thành đá. Đếm xem có bao nhiêu đường đi từ ô $(1, 1)$ đến ô (N, M)

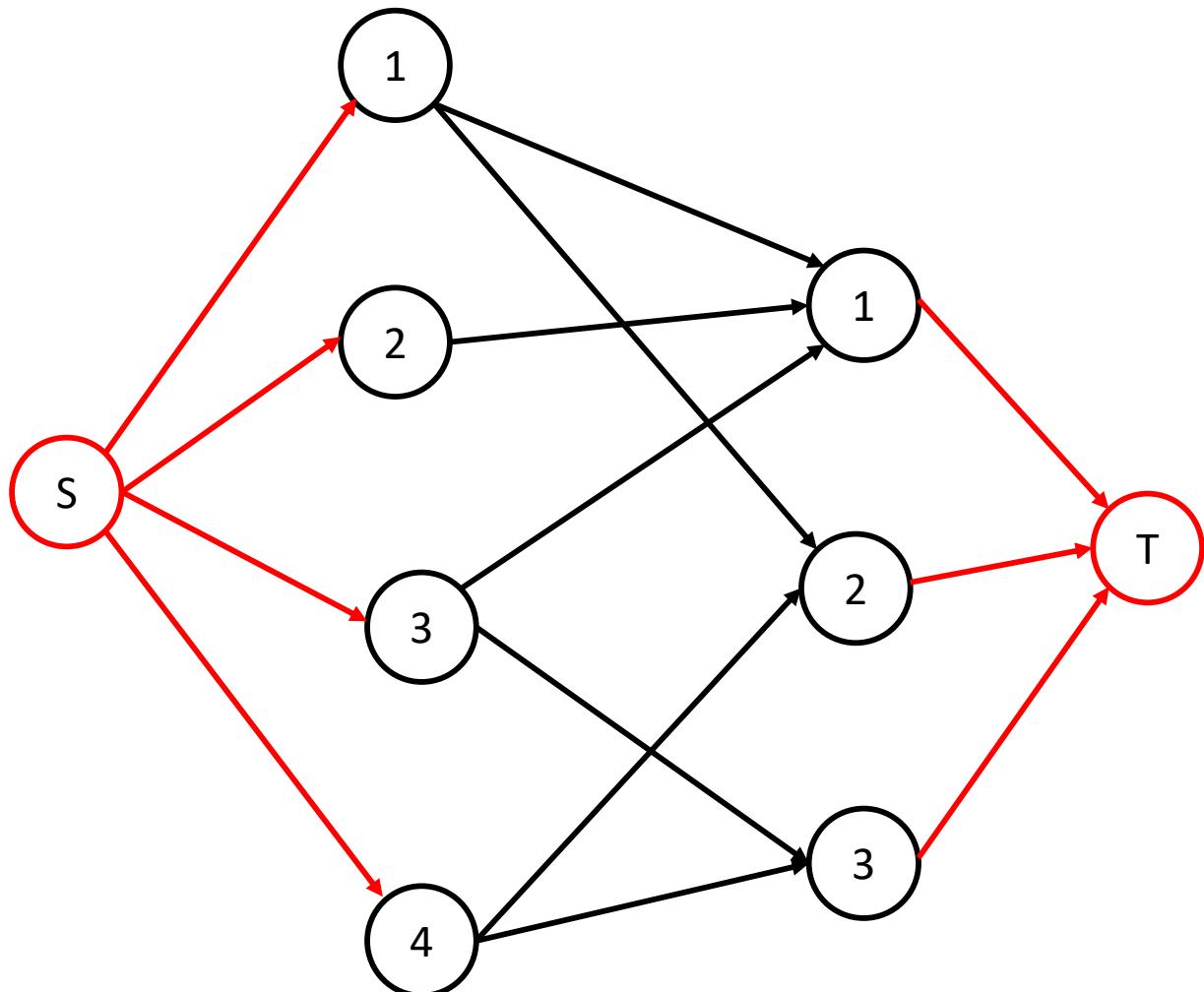


=> Kết nối các ô có thể đi đến được với nhau bằng cách số có chứa là 1. Ta có đỉnh phát là $(1, 1)$, đỉnh thu là (N, M) . Đáp án chính là luồng cực đại

Bài tập vận dụng

BT3: Có N cậu bé và M chai nước, biết danh sách các chai nước mà cậu bé thứ i ($1 \leq i \leq N$) thích uống. Hãy tìm cách chia các chai nước cho các cậu bé sau cho được nhiều cậu bé uống được loại nước yêu thích nhất càng tốt

=> Giải bài toán cặp ghép cực đại trên đồ thị 2 phía bằng luồng cực đại



Real world applications

Baseball elimination

Ở một thời điểm nào đó của một mùa giải bóng chày, tìm kiếm câu lập bộ nào không thể dành được top 1 trong mùa giải đó nữa

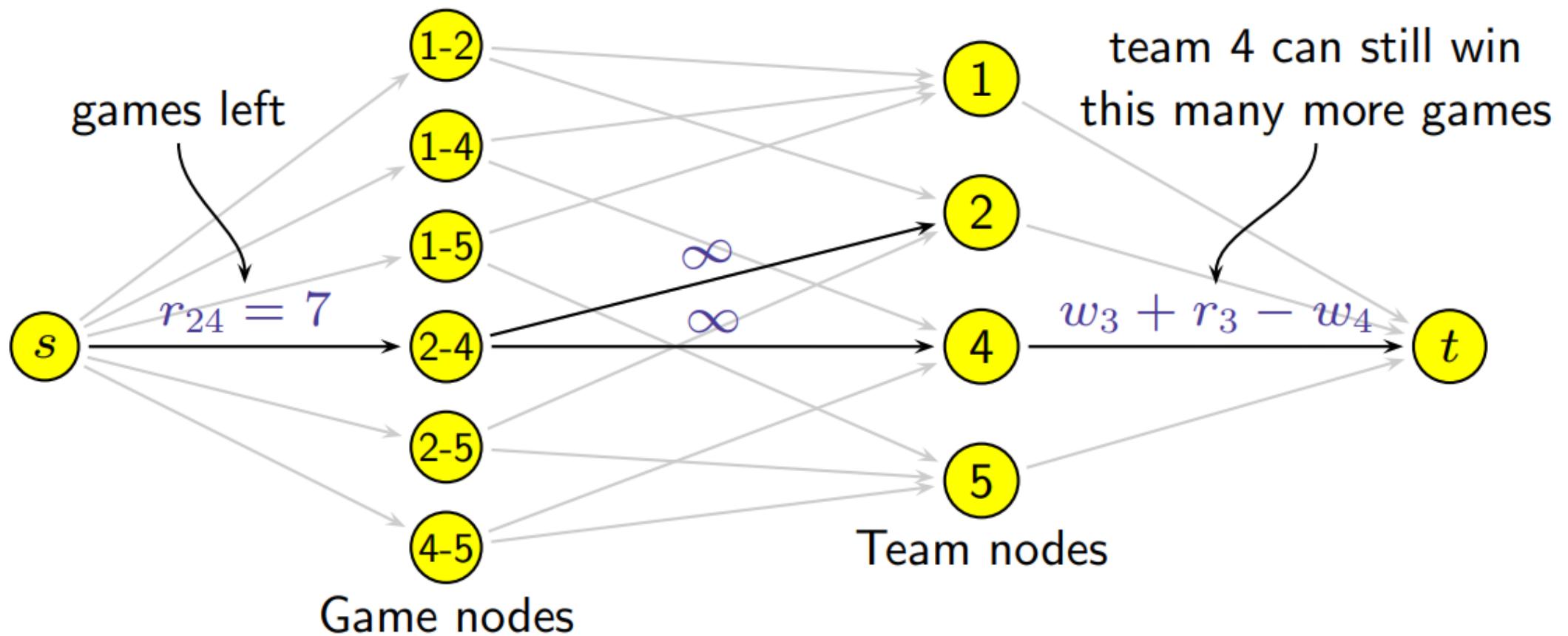


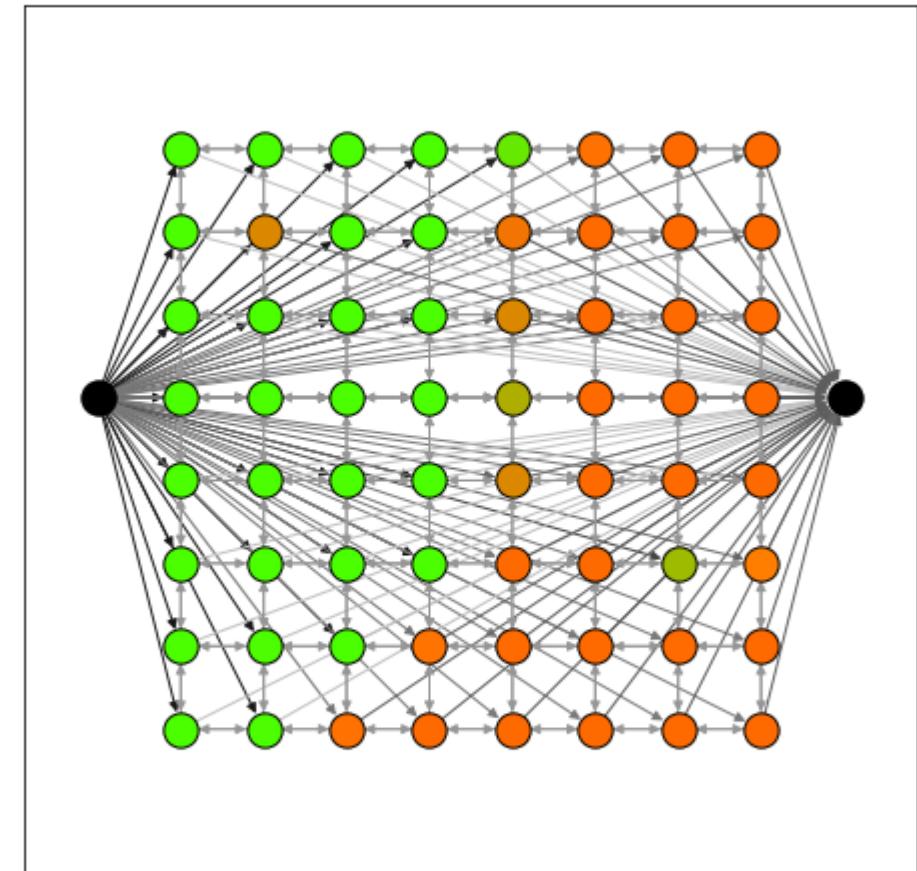
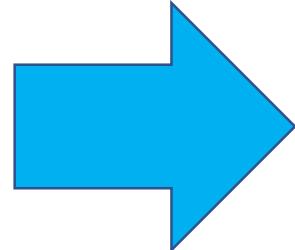
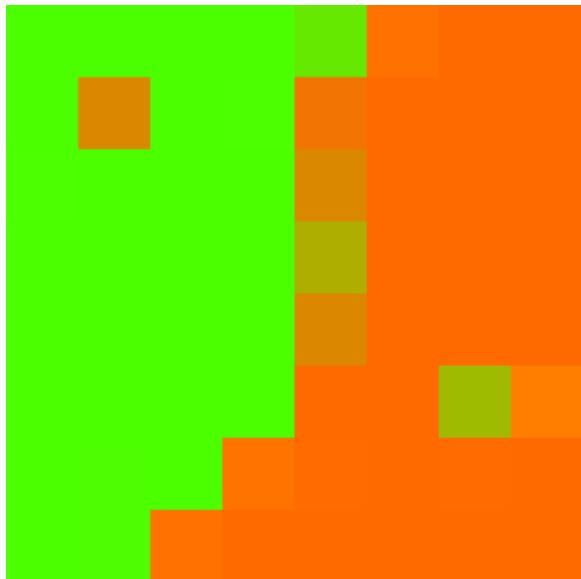
Image segmentation

V: Pixels

E: Connections between Neighbouring pixels

Giữa các pixels gần nhau thì có thể được kết nối dựa các đặc tính của chúng
(màu sắc, độ tương phản, ...)

Dùng min-cut để tìm ra background và foreground



Một số ứng dụng khác:

Airline scheduling

Circulation–demand problem

Census tabulation (matrix rounding)

Project selection (max weight closure).

THANK
YOU