



## Lecture 16

# More problems on the Greedy Method

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

# Greedy method

The **greedy method** is a general algorithm design technique, in which given:

- **configurations**: different choices we need to make
- **objective function**: a score assigned to all configurations, which we want to either **maximize** or **minimize**

We should make choices **greedily**: We can find a **globally-optimal solution** by a series of **local improvements** from a starting configuration.

# Scheduling jobs/tasks

**Problem 1:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform all the tasks using a **minimum** number of **machines**.  
A machine can **serve one task at a given time**.

# Scheduling jobs/tasks

**Problem 1:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform all the tasks using a **minimum** number of **machines**.  
A machine can **serve one task at a given time**.

**Idea:** Sort tasks in **increasing order** of their **start** time. Assign **first** task to **machine 1** and set  $K = 1$ .

When considering a **new task**, if **all machines are busy**, create a **new machine**, set  $K = K + 1$  and assign the **new task to the new machine** otherwise assign the **new task to an available machine**.

# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

**Idea:** Sort tasks in **increasing order** of their **finish** time. Perform **first** task and **remove all overlapping** tasks with first task. **Repeat** the same process to the remaining tasks.

**Example:** 7 Tasks,  $[1,4]$ ,  $[1,3]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[6,9]$ ,  $[7,8]$

# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

**Idea:** Sort tasks in **increasing order** of their **finish** time. Perform **first** task and **remove all overlapping** tasks with first task. **Repeat** the same process to the remaining tasks.

**Example:** 7 Tasks,  $[1,4]$ ,  $[1,3]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[6,9]$ ,  $[7,8]$

**Sort:**  $[1,3]$ ,  $[1,4]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[7,8]$ ,  $[6,9]$

$[1,3]$

# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

**Idea:** Sort tasks in **increasing order** of their **finish** time. Perform **first** task and **remove all overlapping** tasks with first task. **Repeat** the same process to the remaining tasks.

**Example:** 7 Tasks,  $[1,4]$ ,  $[1,3]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[6,9]$ ,  $[7,8]$

**Sort:**  ~~$[1,3]$~~ ,  ~~$[1,4]$~~ ,  ~~$[2,5]$~~ ,  $[3,7]$ ,  $[4,7]$ ,  $[7,8]$ ,  $[6,9]$

$[1,3]$      $[3,7]$



# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

**Idea:** Sort tasks in **increasing order** of their **finish** time. Perform **first** task and **remove all overlapping** tasks with first task. **Repeat** the same process to the remaining tasks.

**Example:** 7 Tasks,  $[1,4]$ ,  $[1,3]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[6,9]$ ,  $[7,8]$

**Sort:**  ~~$[1,3]$~~ ,  ~~$[1,4]$~~ ,  ~~$[2,5]$~~ ,  ~~$[3,7]$~~ ,  ~~$[4,7]$~~ ,  $[7,8]$ ,  ~~$[6,9]$~~

$[1,3]$       $[3,7]$

# Scheduling jobs/tasks

**Problem 2:** Given: a set  $T$  of  $n$  tasks, each having a start time  $s_i$  and a finish time  $f_i$  (where  $s_i < f_i$ )

**Goal:** Perform **as many tasks as possible** using one machine.  
In other words, find the **maximum number of non-overlapping intervals**.

**Idea:** Sort tasks in **increasing order** of their **finish** time. Perform **first** task and **remove all overlapping** tasks with first task. **Repeat** the same process to the remaining tasks.

**Example:** 7 Tasks,  $[1,4]$ ,  $[1,3]$ ,  $[2,5]$ ,  $[3,7]$ ,  $[4,7]$ ,  $[6,9]$ ,  $[7,8]$

**Sort:**  ~~$[1,3]$~~ ,  ~~$[1,4]$~~ ,  ~~$[2,5]$~~ ,  ~~$[3,7]$~~ ,  ~~$[4,7]$~~ ,  $[7,8]$ ,  ~~$[6,9]$~~

$[1,3]$        $[3,7]$        $[7,8]$

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

(5\$, 3), (4\$, 4), (2\$, 2), (2\$, 2), (2\$, 4), (1\$, 2), (1\$, 4)

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

(4\$, 4), (2\$, 2), (2\$, 2), (2\$, 4), (1\$, 2), (1\$, 4)

Schedule task (5\$,3) between times 2-3.

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

(2\$, 2), (2\$, 2), (2\$, 4), (1\$, 2), (1\$, 4)

Schedule task (5\$,3) between times 2-3. Schedule task (4\$,4) between times 3-4.



# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

(2\$, 2), (2\$, 4), (1\$, 2), (1\$, 4)

Schedule task (5\$,3) between times 2-3. Schedule task (4\$,4) between times 3-4.  
Schedule task (2\$,2) between times 1-2.

# Scheduling jobs/tasks

**Problem 3:** You are given a set  $T$  of  $n$  tasks, each having a deadline time  $f_i$  and profit  $p_i$  if completed and needs one unit of time to be completed. You have only **one machine**.

**Goal:** Complete **non-overlapping** tasks to **maximize** your profit.

**Example:** 7 Tasks, (5\$, 3), (2\$, 2), (2\$, 2), (1\$, 2), (4\$, 4), (2\$, 4), (1\$, 4)

**Solution :** Task 2, Task 3, Task 1, Task 5 with profit 13\$

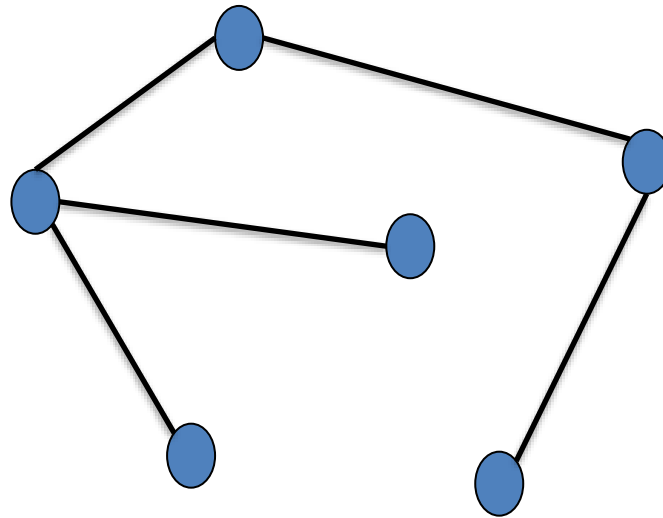
**Idea:** Sort tasks in **decreasing order** of their **profit**. **Repeat** the following until run out of tasks: Choose **first task** and schedule it at the **latest time possible without exceeding deadline**. If not possible, discard the task.

(2\$, 4), (1\$, 2), (1\$, 4)

Schedule task (5\$,3) between times 2-3. Schedule task (4\$,4) between times 3-4.  
Schedule task (2\$,2) between times 1-2. Schedule task (2\$,2) between times 0-1.

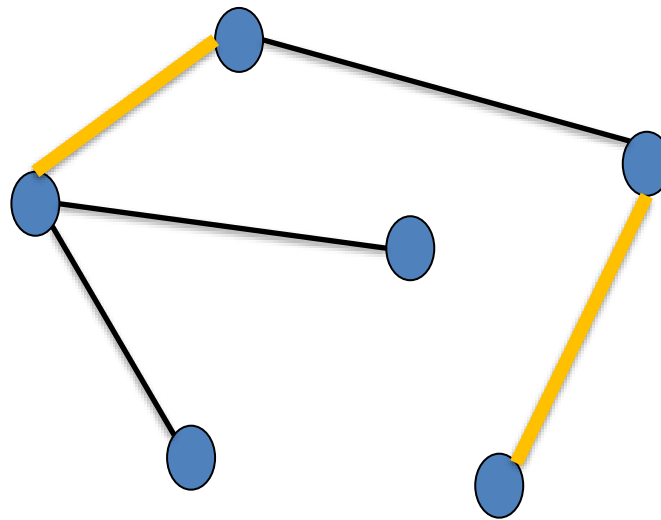
# Problems on trees using Greedy

**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do not share a vertex.



# Problems on trees using Greedy

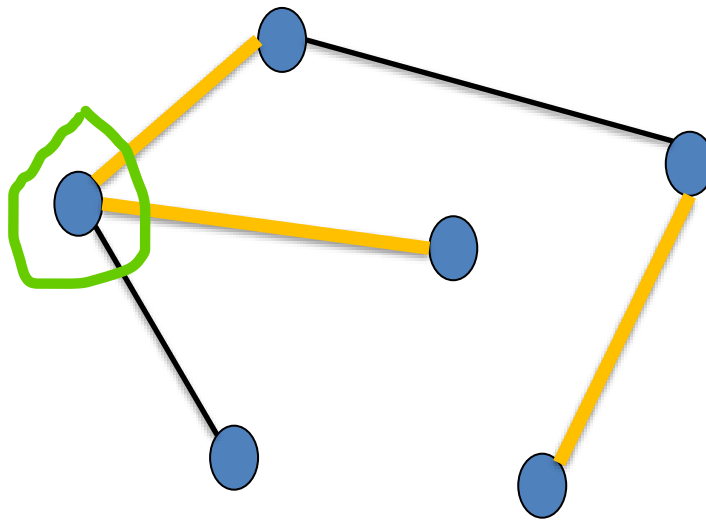
**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do not share a vertex.



**Matching**

# Problems on trees using Greedy

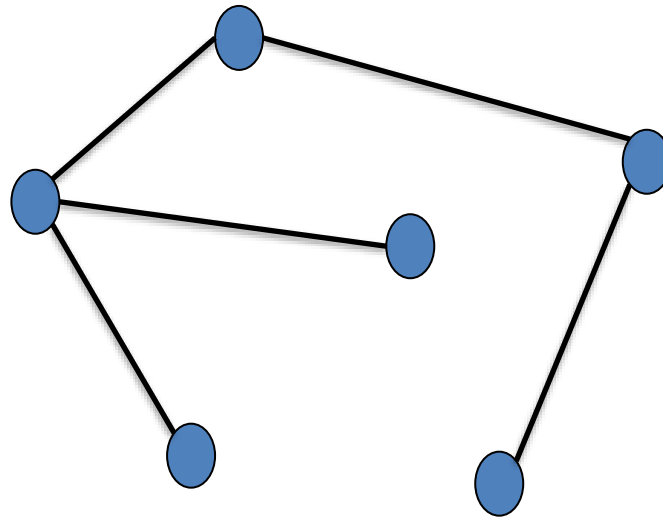
**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do **not share a vertex**.



**Not a Matching**

# Problems on trees using Greedy

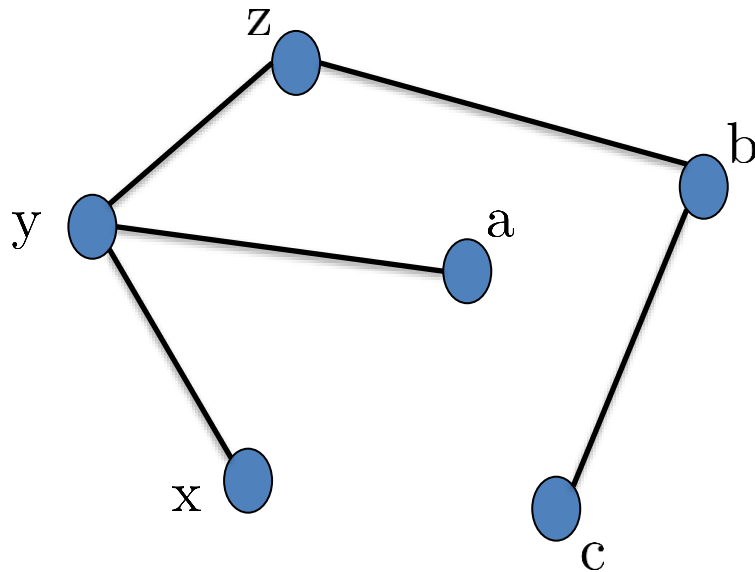
**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do not share a vertex.



**Problem:** Given a **tree** graph, compute/find a **maximum matching**.

# Problems on trees using Greedy

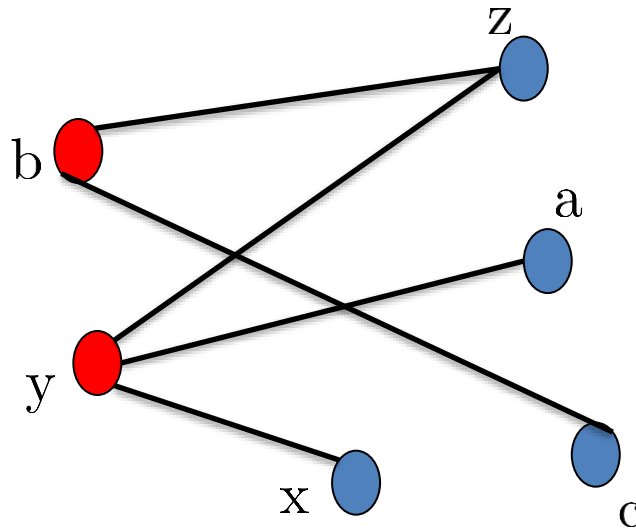
**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do **not share a vertex**.



**Problem:** Given a **tree** graph, compute/find a **maximum matching**. We know how to do it for **bipartite** graphs via maxflow!!

# Problems on trees using Greedy

**Definition:** Given a graph  $G$ , a **matching** is a collection of edges that do **not share a vertex**.



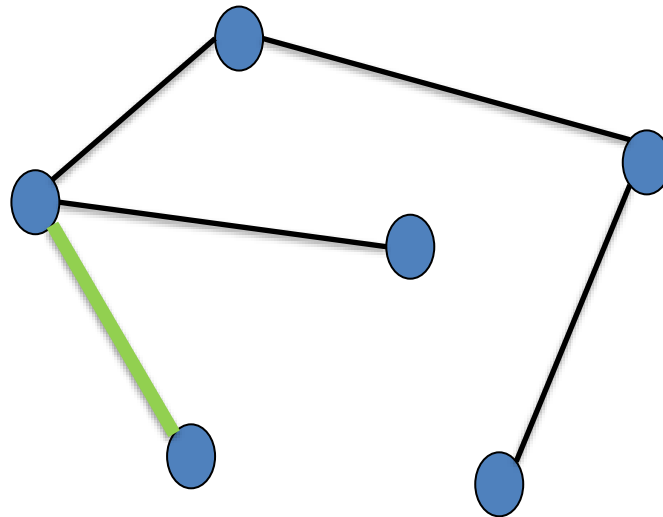
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
We know how to do it for **bipartite** graphs via maxflow!!

**Trees are bipartite!**



# Maximum Matching on trees

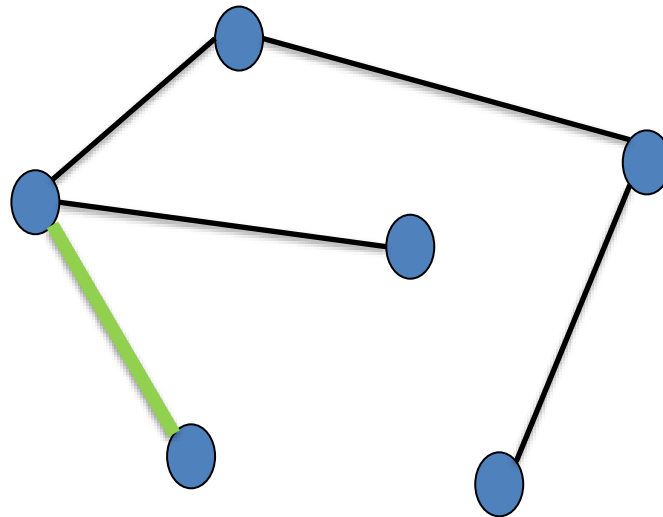
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Question:** The green edge has a **leaf** as an endpoint. Should it be **in** the matching?

# Maximum Matching on trees

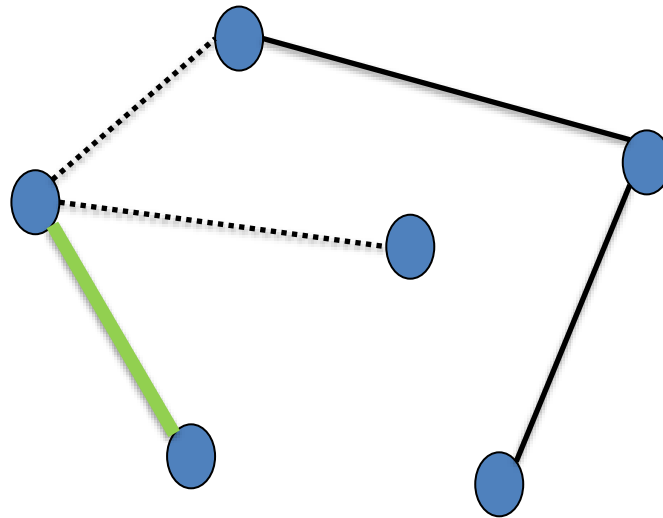
**Problem:** Given a **tree** graph, compute/find a **maximum matching**. Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **union of trees**. Repeat until run out of edges.

# Maximum Matching on trees

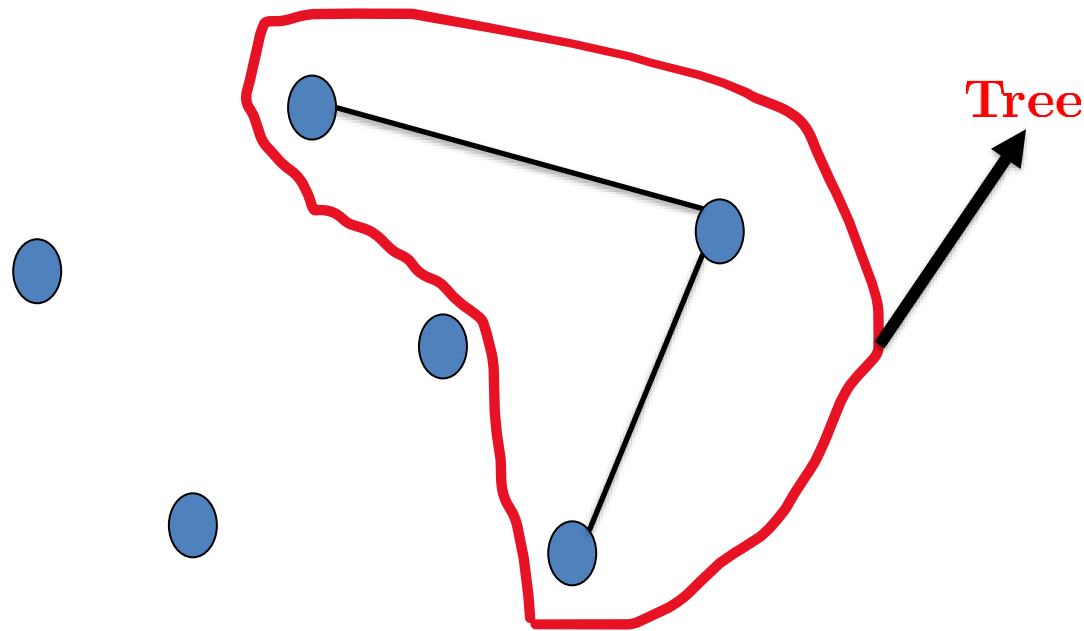
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **union of trees**. Repeat until run out of edges.

# Maximum Matching on trees

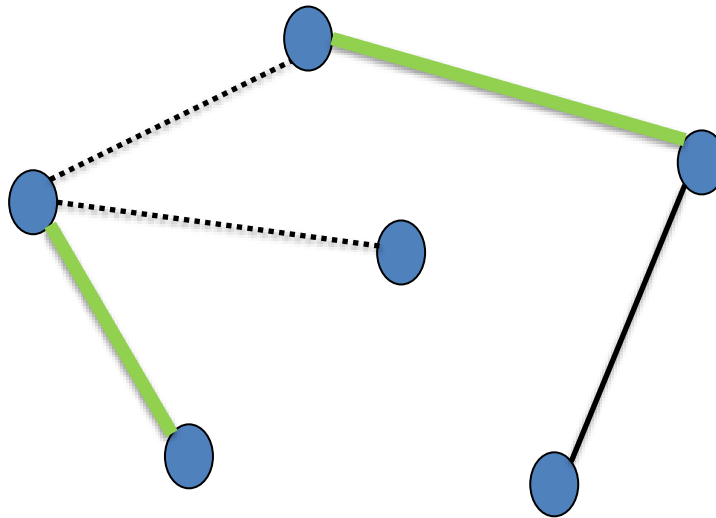
**Problem:** Given a **tree** graph, compute/find a **maximum matching**. Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **union of trees**. Repeat until run out of edges.

# Maximum Matching on trees

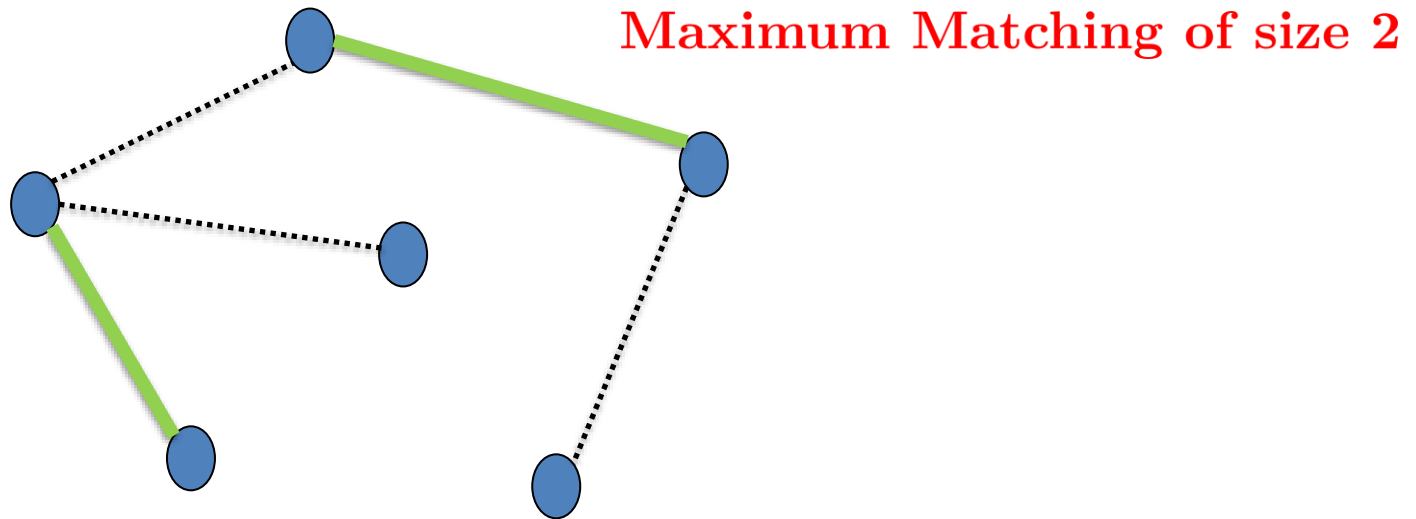
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **tree** as well. Repeat until run out of edges.

# Maximum Matching on trees

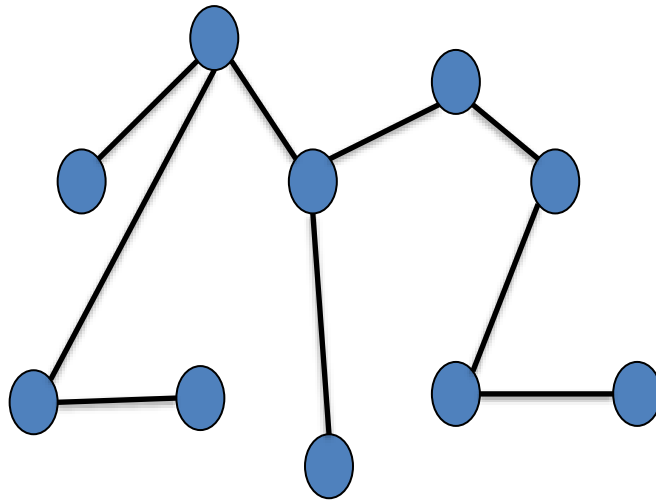
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **tree** as well. Repeat until run out of edges.

# Maximum Matching on trees

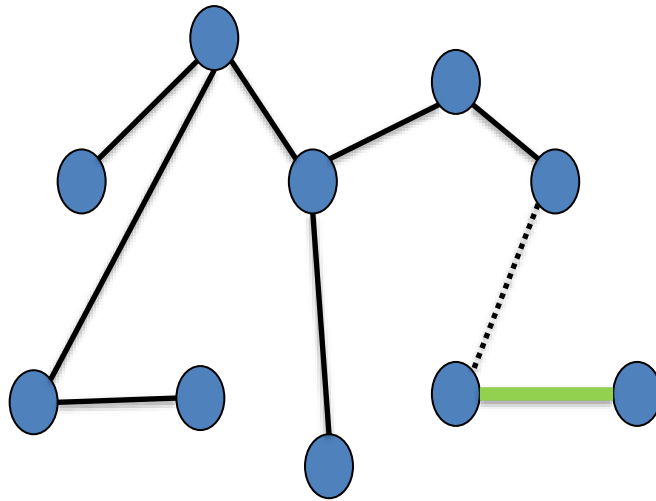
**Problem:** Given a **tree** graph, compute/find a **maximum matching**. Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **tree** as well. Repeat until run out of edges.

# Maximum Matching on trees

**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.

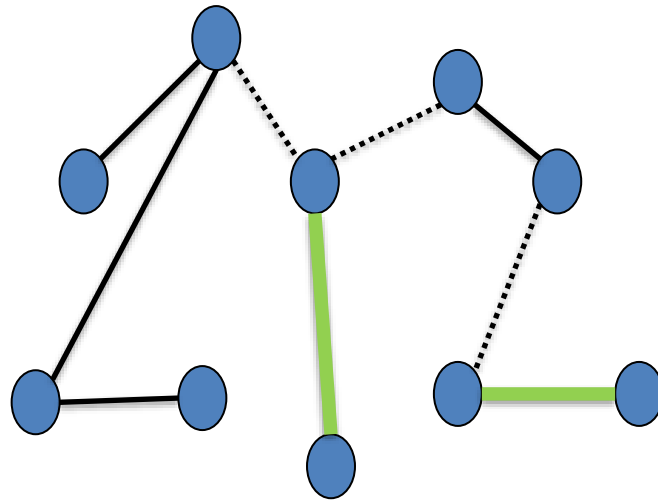


**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the **matching**. **Remove all other incident edges**. The new graph is a **tree** as well. **Repeat until run out of edges**.



# Maximum Matching on trees

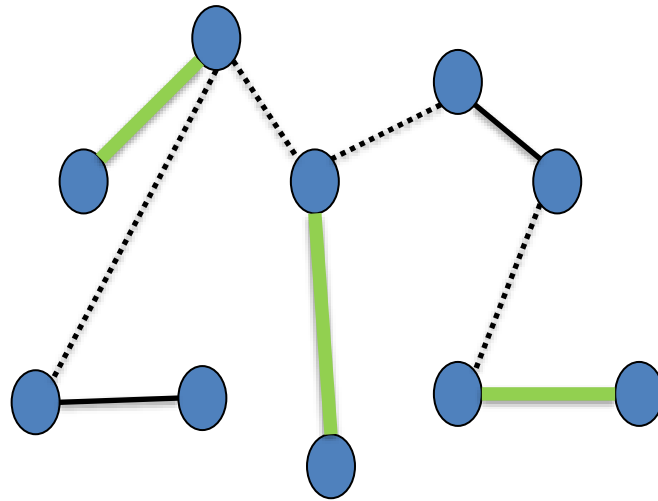
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the **matching**. **Remove all other incident edges**. The new graph is a **tree** as well. **Repeat until run out of edges**.

# Maximum Matching on trees

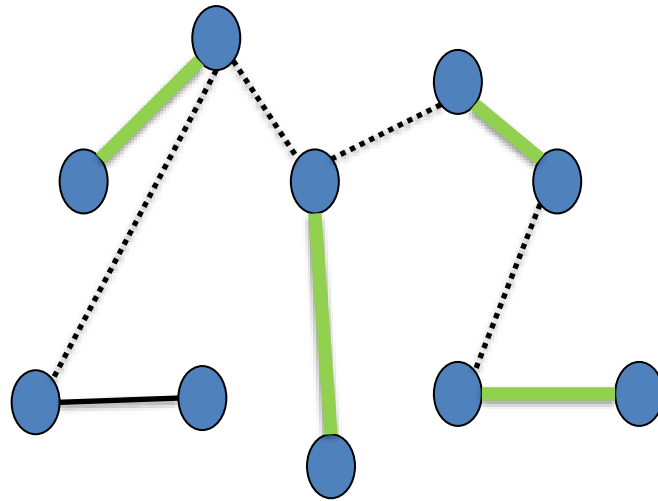
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the **matching**. **Remove all other incident edges**. The new graph is a **tree** as well. **Repeat until run out of edges**.

# Maximum Matching on trees

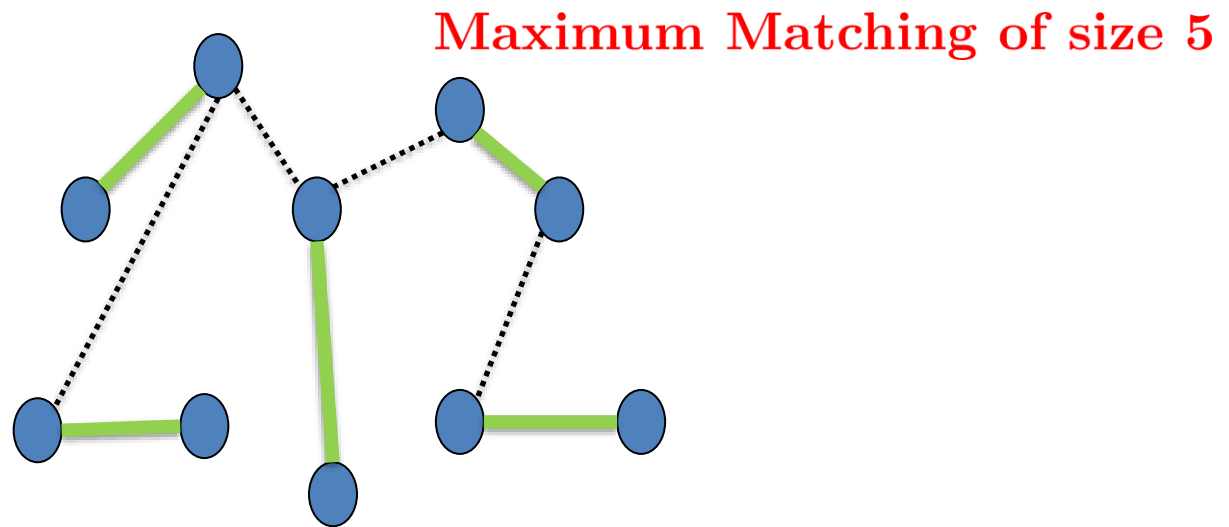
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the **matching**. **Remove all other incident edges**. The new graph is a **tree** as well. **Repeat until run out of edges**.

# Maximum Matching on trees

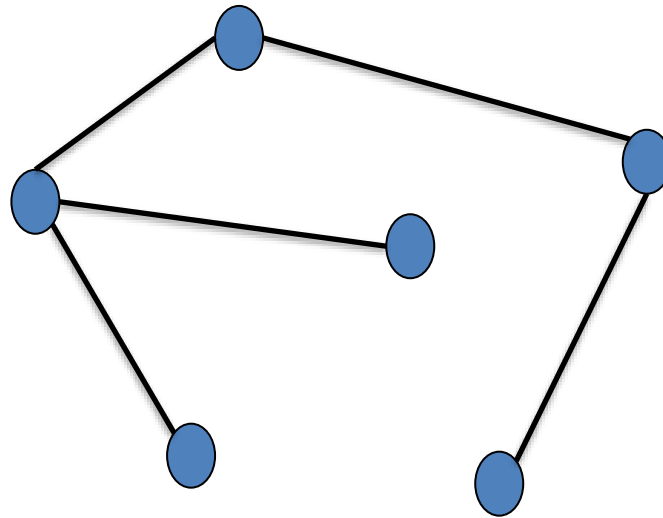
**Problem:** Given a **tree** graph, compute/find a **maximum matching**.  
Do not use Maxflow, but directly Greedy.



**Idea:** Choose an **edge** with one **endpoint** being a **leaf** and put it in the matching. Remove all other incident edges. The new graph is a **tree** as well. Repeat until run out of edges.

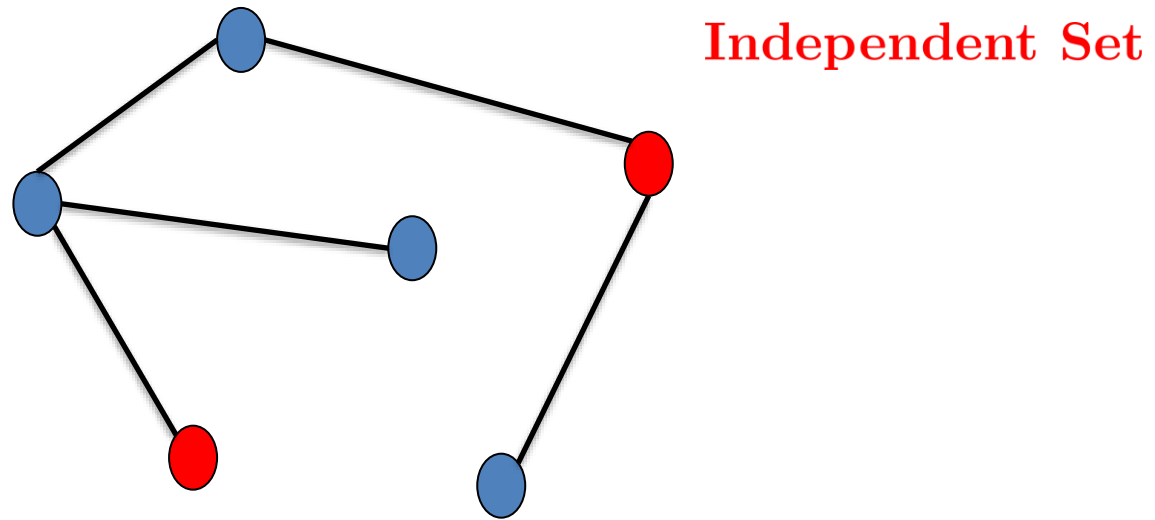
# Problems on trees using Greedy

**Definition:** Given a graph  $G$ , an **independent set** is a collection of vertices that do **not share an edge**.



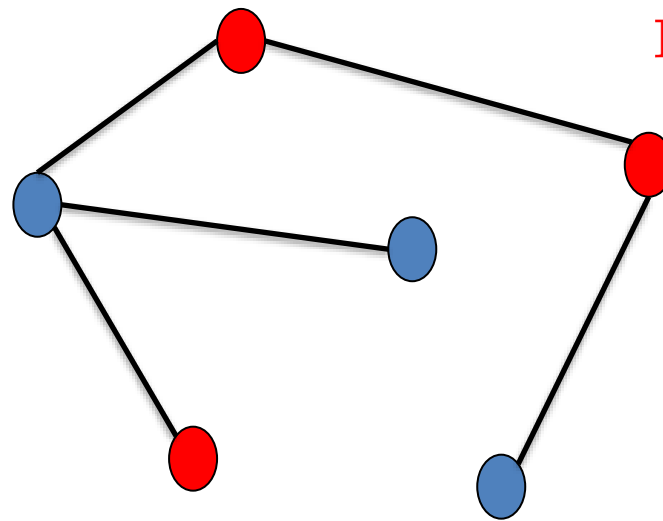
# Problems on trees using Greedy

**Definition:** Given a graph  $G$ , an **independent set** is a collection of vertices that do **not share an edge**.



# Problems on trees using Greedy

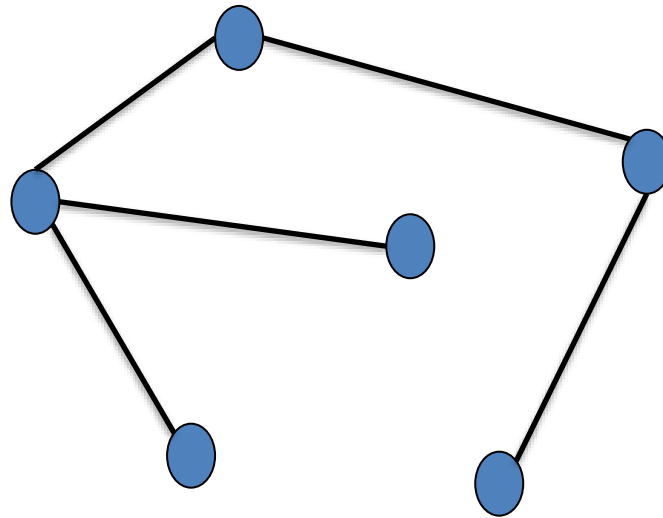
**Definition:** Given a graph  $G$ , an **independent set** is a collection of vertices that do **not share an edge**.



**Not an Independent Set**

# Problems on trees using Greedy

**Definition:** Given a graph  $G$ , an **independent set** is a collection of vertices that do **not share an edge**.

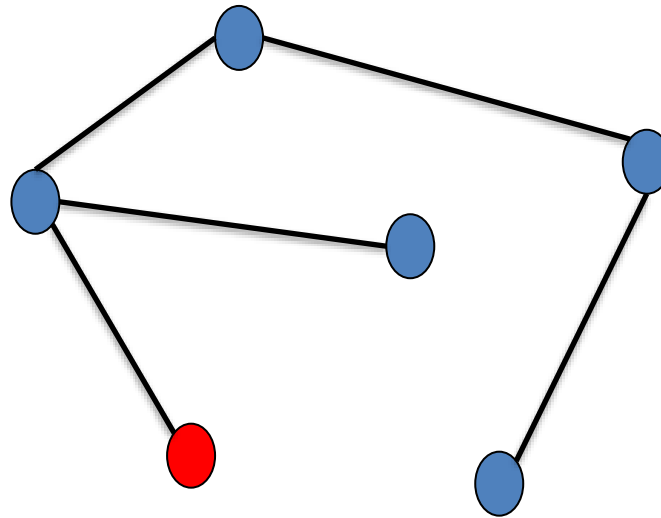


**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



# Maximum Independent set on trees

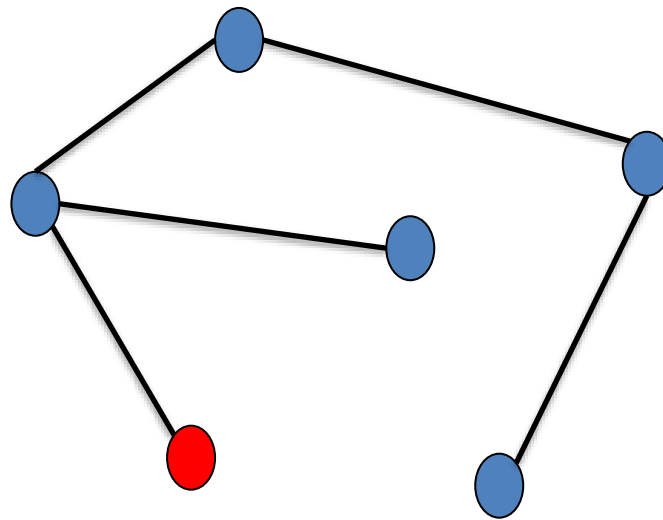
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Question:** Should a **leaf** be part of the independent set?

# Maximum Independent set on trees

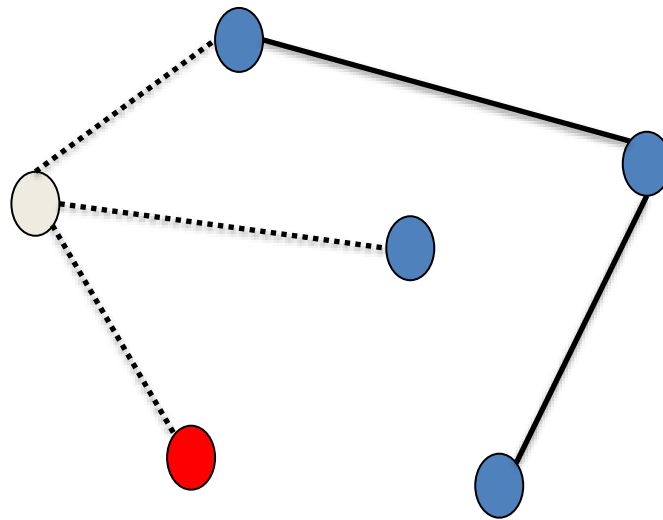
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

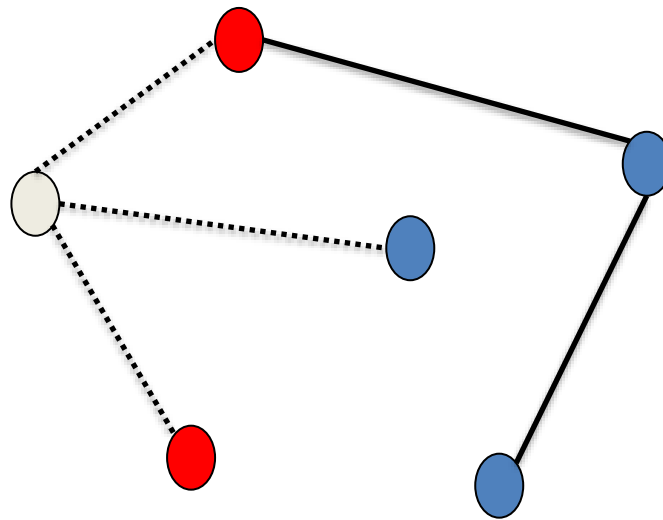
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

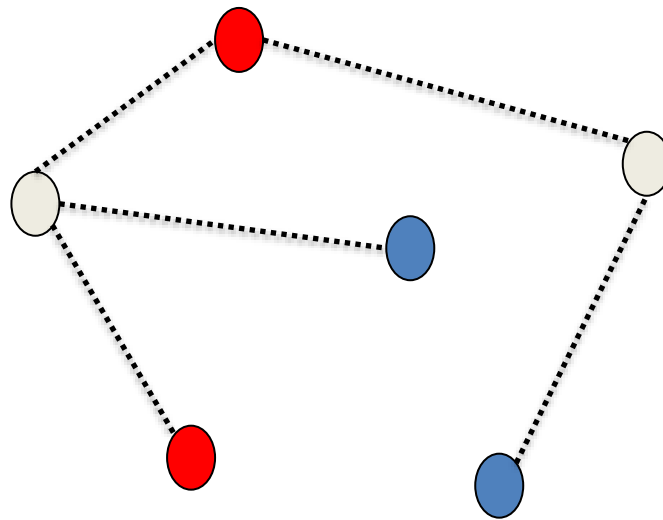
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

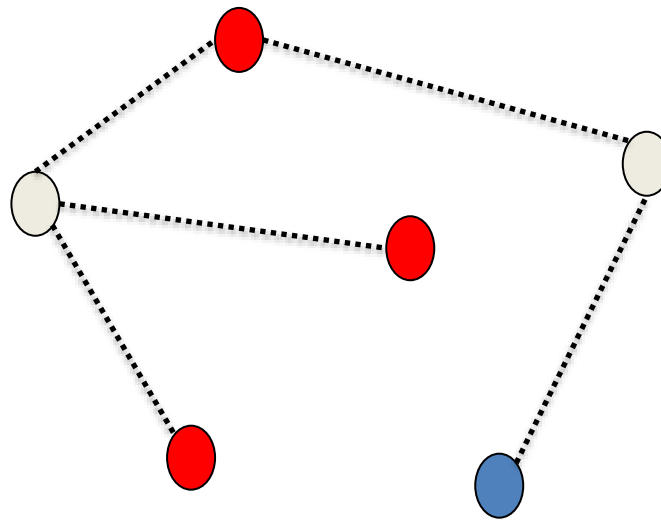
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex (and incident edges)**. The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

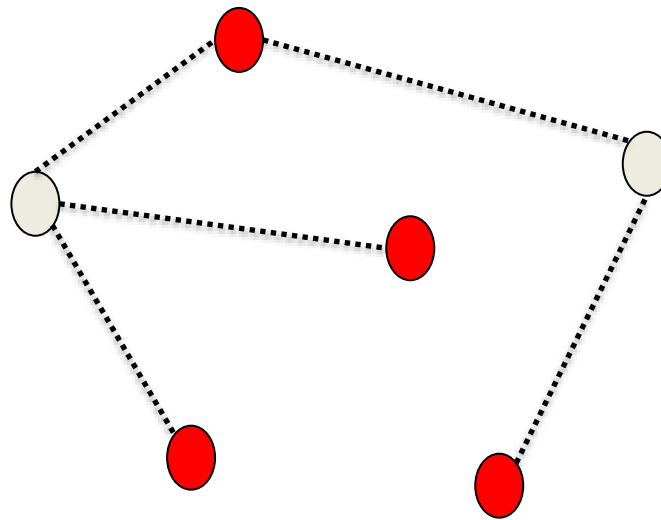
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

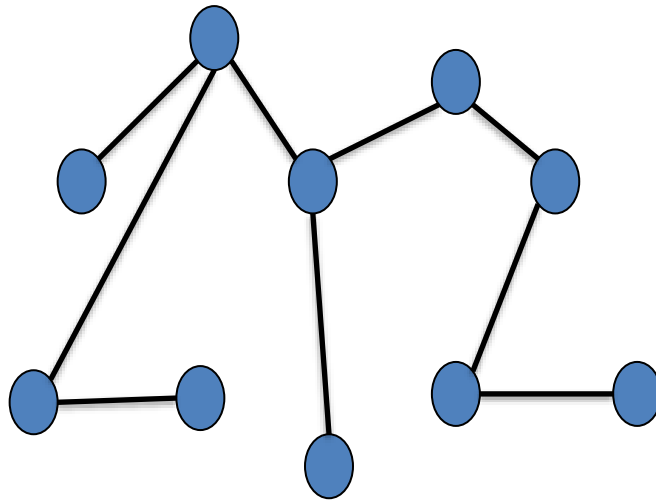
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex (and incident edges)**. The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.

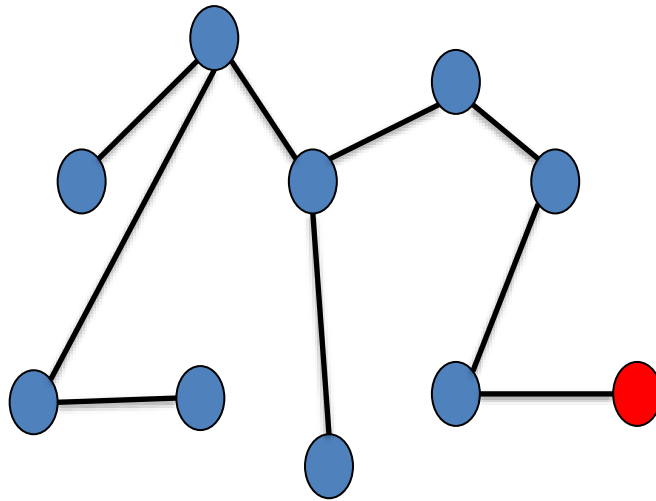


**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.



# Maximum Independent set on trees

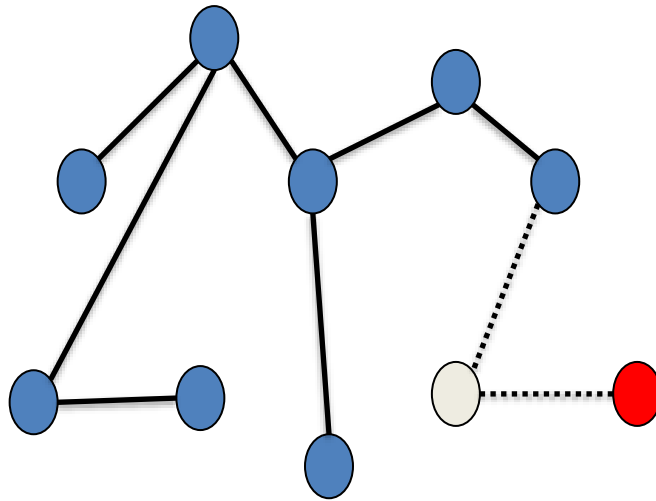
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

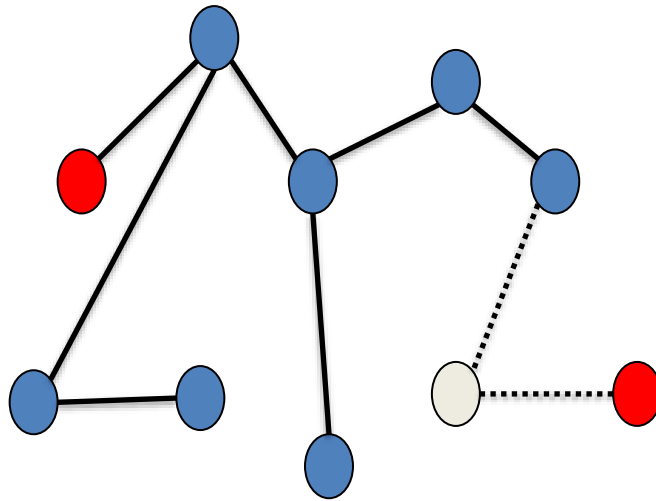
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

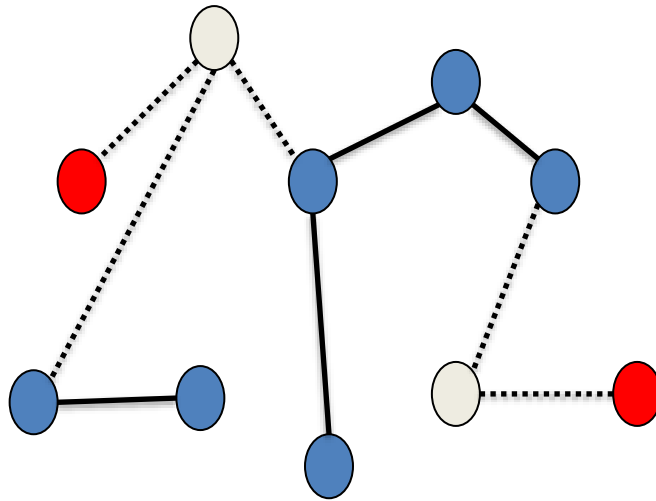
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

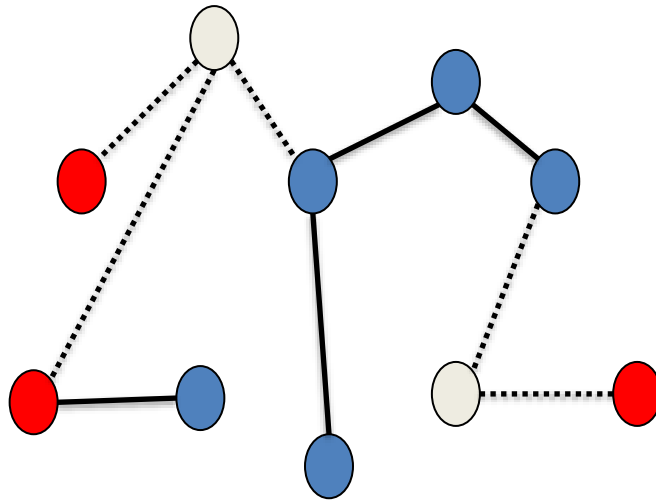
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

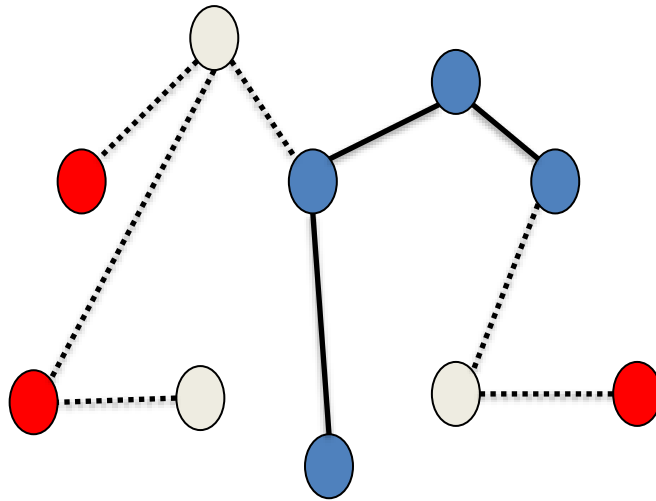
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex (and incident edges)**. The new graph is a union of **trees**. **Repeat until run out of vertices**.

# Maximum Independent set on trees

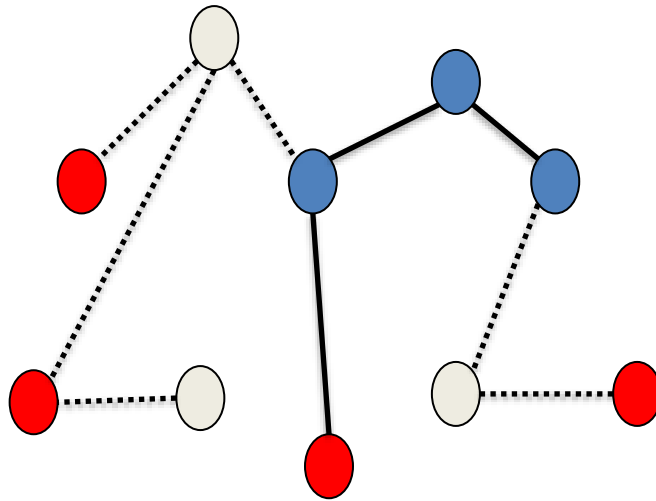
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a leaf (or isolated vertex) and put it in the independent set. Remove the neighboring vertex (and incident edges). The new graph is a union of trees. Repeat until run out of vertices.

# Maximum Independent set on trees

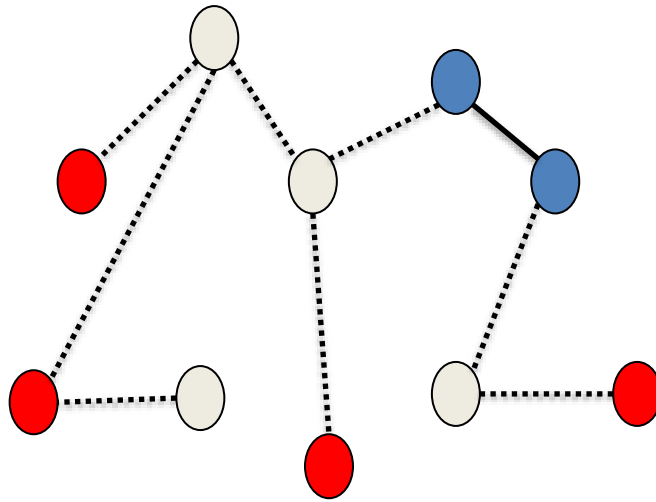
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a leaf (or isolated vertex) and put it in the independent set. Remove the neighboring vertex (and incident edges). The new graph is a union of trees. Repeat until run out of vertices.

# Maximum Independent set on trees

**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.

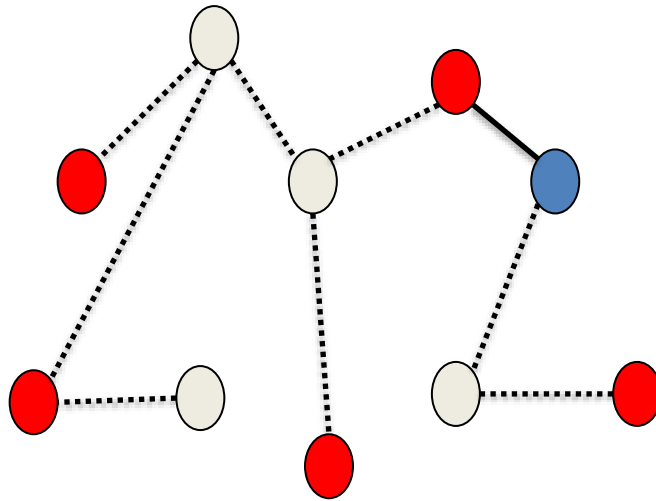


**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex** (and incident edges). The new graph is a union of **trees**. **Repeat until run out of vertices**.



# Maximum Independent set on trees

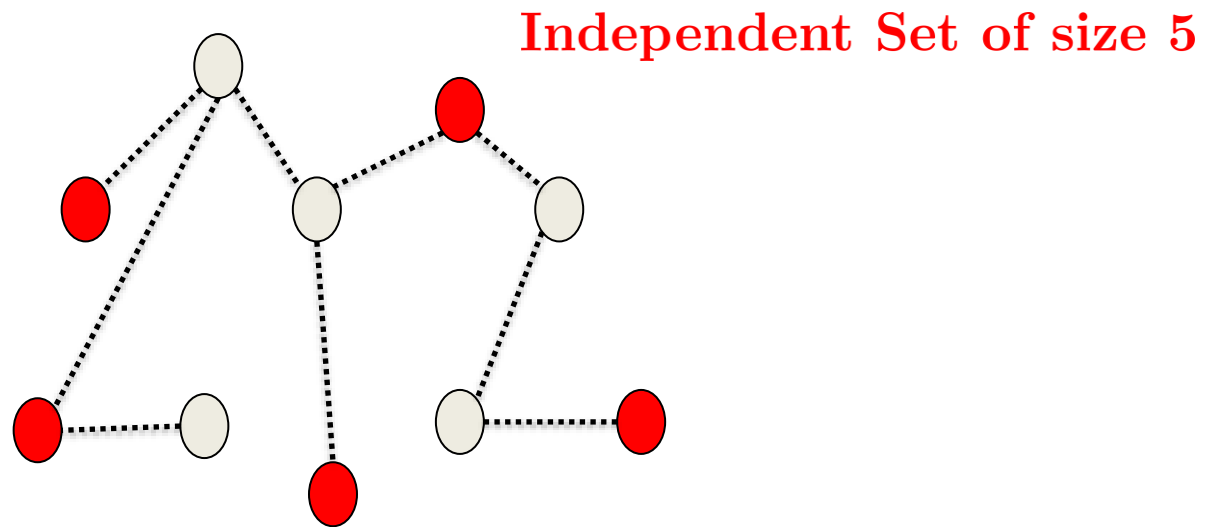
**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a leaf (or isolated vertex) and put it in the independent set. Remove the neighboring vertex (and incident edges). The new graph is a union of trees. Repeat until run out of vertices.

# Maximum Independent set on trees

**Problem:** Given a **tree** graph, compute/find a **maximum independent set**.



**Idea:** Choose a **leaf (or isolated vertex)** and put it in the **independent set**. **Remove the neighboring vertex (and incident edges)**. The new graph is a union of **trees**. **Repeat until run out of vertices**.