CMPSC 465
Data Structures and Algorithms
Spring 2022

Instructor: Chunhao Wang

Greedy algorithms

Greedy algorithms

Matroid, Task Scheduling
(Cormen et al. 16.4, 16.5)

Warning

Very abstract!

"Computer Science is a science of **abstraction** — creating the right model for a problem and devising the appropriate mechanizable techniques to solve it."

— Alfred Aho

Matroid

Matroid is a combinatorial structure

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$)

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
    - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)
    - if $A, B \in \mathcal{I}$ and $|A| < |B|$

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)
   - if $A, B \in \mathcal{I}$ and $|A| < |B|$ then $\exists x \in B - A$ s.t.

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)
   - if $A, B \in \mathcal{I}$ and $|A| < |B|$ then $\exists x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)
   - if $A, B \in \mathcal{I}$ and $|A| < |B|$ then $\exists x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$ (Exchange property)

Matroid

Matroid is a combinatorial structure

Many problems for which a greedy approach provides optimal solution can be formulated as some problems involve matroids

A more abstract view of graph vs. matroid

Graph $G = (V, E)$

1. V: finite nonempty set
2. E: a collection of subsets of V (or $E \subseteq \mathcal{P}(V)$) each $e \in E$ has two elements of V called an *edge*

Matroid $M = (S, \mathcal{I})$

1. S: finite nonempty set
2. $\mathcal{I} \subseteq \mathcal{P}(S)$ s.t.
   - if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$ (Hereditary property)
   - if $A, B \in \mathcal{I}$ and $|A| < |B|$ then $\exists x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$ (Exchange property)

For a matroid $M = (S, \mathcal{I})$, each $A \in \mathcal{I}$ is called an **independent subset**

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary:

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest

Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example

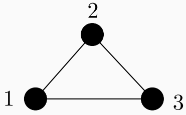
Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example



Graphic Matroid

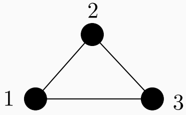
Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example

$$S = \{(1,2), (2,3), (3,1)\}$$



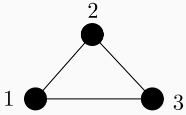
Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example



$$S = \{(1,2),(2,3),(3,1)\}$$
$$\mathcal{I} = \{\emptyset, \{(1,2)\}, \{(2,3)\}, \{(1,3)\}, \{(1,2),(2,3)\}, \{(1,3),(1,2)\}, \{(1,3),(2,3)\}\}$$

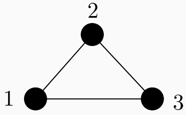
Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example



$S = \{(1,2),(2,3),(3,1)\}$
$\mathcal{I} = \{\emptyset, \{(1,2)\}, \{(2,3)\}, \{(1,3)\}, \{(1,2),(2,3)\}, \{(1,3),(1,2)\}, \{(1,3),(2,3)\}\}$
Say $A = \{(2,3)\}, B = \{(1,3),(1,2)\}$

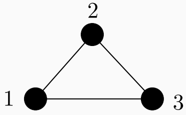
Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example



$S = \{(1,2), (2,3), (3,1)\}$
$\mathcal{I} = \{\emptyset, \{(1,2)\}, \{(2,3)\}, \{(1,3)\}, \{(1,2),(2,3)\},$
$\{(1,3),(1,2)\}, \{(1,3),(2,3)\}\}$
Say $A = \{(2,3)\}, B = \{(1,3),(1,2)\}$
$x \in B - A$, for example, $x = (1,3)$

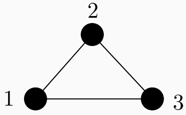
Graphic Matroid

Given undirected $G = (V, E)$, construct **graphic matroid** $M_G = (S, \mathcal{I})$ via

- $S = E$
- $\mathcal{I} = \{A \subseteq E : A \text{ is acyclic}\}$
  A is a forest

Why M_G is a matroid?

- Hereditary: a subset of forest is still a forest
- Exchange: demonstrate by example



$S = \{(1,2),(2,3),(3,1)\}$
$\mathcal{I} = \{\emptyset, \{(1,2)\}, \{(2,3)\}, \{(1,3)\}, \{(1,2),(2,3)\},$
$\{(1,3),(1,2)\}, \{(1,3),(2,3)\}\}$
Say $A = \{(2,3)\}, B = \{(1,3),(1,2)\}$
$x \in B - A$, for example, $x = (1,3)$
then $A \cup \{x\} = \{(2,3),(1,3)\} \subseteq \mathcal{I}$

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Proof.

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Proof.

Suppose $A, B \in \mathcal{I}$ are both maximal, but $|B| > |A|$.

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Proof.

Suppose $A, B \in \mathcal{I}$ are both maximal, but $|B| > |A|$. Then by exchange property, there exists an $x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$,

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Proof.

Suppose $A, B \in \mathcal{I}$ are both maximal, but $|B| > |A|$. Then by exchange property, there exists an $x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$, which is a contradiction of A being maximal $\square$

Connection to spanning tree

Definition

For all $A \in \mathcal{I}$, $x \in S$ is an **extension** of A if $A \cup \{x\} \in \mathcal{I}$

Definition

$A \in \mathcal{I}$ is **maximal** if it has no extension

Theorem

All maximal $A \in \mathcal{I}$ have the same size

Proof.

Suppose $A, B \in \mathcal{I}$ are both maximal, but $|B| > |A|$. Then by exchange property, there exists an $x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$, which is a contradiction of A being maximal $\square$

For connected undirected G, every maximal independent subset of M_G must be a tree with $|V| - 1$ edges. Hence it is a spanning tree

Weighted matroid

Definition

A **weighted matroid** $M = (S, \mathcal{I})$ is one that has a strictly positive weight $w(x)$ for all $x \in S$.

Weighted matroid

Definition

A **weighted matroid** $M = (S, \mathcal{I})$ is one that has a strictly positive weight $w(x)$ for all $x \in S$. The weight function w extends to $\mathcal{I}$ as for all $A \in \mathcal{I}$:

$$w(A) = \sum_{a \in A} w(a)$$

Weighted matroid

Definition

A **weighted matroid** $M = (S, \mathcal{I})$ is one that has a strictly positive weight $w(x)$ for all $x \in S$. The weight function w extends to $\mathcal{I}$ as for all $A \in \mathcal{I}$:

$$w(A) = \sum_{a \in A} w(a)$$

Note: for graphic matroids, weight of M_G is corresponding to edge weights

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Application: MST of $G \to$ max-weighted independent subset of M_G via

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Application: MST of $G \to$ max-weighted independent subset of M_G via

- G with $w(e) \to M_G$ with $w'(e) = c - w(e)$ where c is a constant larger than the largest $w(e)$

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Application: MST of $G \to$ max-weighted independent subset of M_G via

- G with $w(e) \to M_G$ with $w'(e) = c - w(e)$ where c is a constant larger than the largest $w(e)$
- For M_G, $w'(e)$ are positive

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Application: MST of $G \to$ max-weighted independent subset of M_G via

- G with $w(e) \to M_G$ with $w'(e) = c - w(e)$ where c is a constant larger than the largest $w(e)$
- For M_G, $w'(e)$ are positive
- For max-weighted independent subset A
  $w'(A) = (|V| - 1)c - w(A)$, so $w(A)$ is minimized

Optimization problem for matroids

Problem (Maximum-weighted Independent Subset)

Given a weighted matroid M, the goal is to find the maximum-weighted independent subset of M

Remark: because weights are positive, it always helps to find a subset as large as possible

Application: MST of $G \to$ max-weighted independent subset of M_G via

- G with $w(e) \to M_G$ with $w'(e) = c - w(e)$ where c is a constant larger than the largest $w(e)$
- For M_G, $w'(e)$ are positive
- For max-weighted independent subset A $w'(A) = (|V| - 1)c - w(A)$, so $w(A)$ is minimized

Hence a max-weighted indep. subset of M_G corresponds to an MST of G

Pseudocode for finding max-weighted independent subset

def GREEDY($M = (S, \mathcal{I})$, *weights* w)**:**

Pseudocode for finding max-weighted independent subset

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
```

Pseudocode for finding max-weighted independent subset

def GREEDY($M = (S, \mathcal{I})$, *weights* w)**:**

    Set $A := \{\}$;

    Sort S in decreasing order of w ;   // $O(n \log n)$

Pseudocode for finding max-weighted independent subset

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;                // O(n log n)
    for x ∈ S:
```

Pseudocode for finding max-weighted independent subset

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;            // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ I:
```

Pseudocode for finding max-weighted independent subset

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;          // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ I:
            A := A ∪ {x};
```

Pseudocode for finding max-weighted independent subset

```
def Greedy(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;          // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ I:
            A := A ∪ {x};
    return A;
```

Pseudocode for finding max-weighted independent subset

def $\textsc{Greedy}(M = (S, \mathcal{I}),$ *weights* $w)$**:**
- Set $A := \{\,\}$;
- Sort S in decreasing order of w ; // $O(n \log n)$
- **for** $x \in S$**:**
  - **if** $A \cup \{x\} \in \mathcal{I}$**:**
    - $A := A \cup \{x\}$;
- **return** A;

Proof of correctness: Cormen et al. 16.4

Pseudocode for finding max-weighted independent subset

def $\textsc{Greedy}(M = (S, \mathcal{I}),\ \textit{weights } w)$**:**

    Set $A := \{\,\}$;
    Sort S in decreasing order of w ; // $O(n \log n)$
    for $x \in S$**:**
        if $A \cup \{x\} \in \mathcal{I}$**:**
            $A := A \cup \{x\}$;
    return A;

Proof of correctness: Cormen et al. 16.4

Running time: let $n = |S|$

```
def GREEDY(M = (S, 𝓘), weights w):
    Set A := { };
    Sort S in decreasing order of w ;                    // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ 𝓘:
            A := A ∪ {x};
    return A;
```

Proof of correctness: Cormen et al. 16.4

Running time: let $n = |S|$

Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$.

Pseudocode for finding max-weighted independent subset

```
1 def GREEDY(M = (S, I), weights w):
2     Set A := { };
3     Sort S in decreasing order of w ;        // O(n log n)
4     for x ∈ S:
5         if A ∪ {x} ∈ I:
6             A := A ∪ {x};
7     return A;
```

Proof of correctness: Cormen et al. 16.4

Running time: let $n = |S|$
Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$. Lines 5-6 takes $O(n \cdot f(n))$

Pseudocode for finding max-weighted independent subset

1 **def** $\textsc{Greedy}(M = (S, \mathcal{I}),\ \textit{weights}\ w)$**:**
2   Set $A := \{\,\}$;
3   Sort S in decreasing order of w ;   // $O(n \log n)$
4   **for** $x \in S$**:**
5   **if** $A \cup \{x\} \in \mathcal{I}$**:**
6             $A := A \cup \{x\}$;
7   **return** A;

Proof of correctness: Cormen et al. 16.4

Running time: let $n = |S|$
Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$. Lines 5-6 takes $O(n \cdot f(n))$

Total running time: $O(n \log n + n \cdot f(n))$

Application: task scheduling

Problem (Task scheduling)

Application: task scheduling

Problem (Task scheduling)

Setup:

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \dots, a_n$*

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \leq d_i \leq n$*

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \le d_i \le n$*
- *$w_1, \ldots, w_n > 0$ penalties if a_i is not completed by d_i*

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \leq d_i \leq n$*
- *$w_1, \ldots, w_n > 0$ penalties if a_i is not completed by d_i*

Goal: *Find a **schedule** (i.e., permutation of tasks) that minimizes the penalties incurred*

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \le d_i \le n$*
- *$w_1, \ldots, w_n > 0$ penalties if a_i is not completed by d_i*

Goal: *Find a **schedule** (i.e., permutation of tasks) that minimizes the penalties incurred*

Example:

task	a	b	c	d
deadline	1	1	4	2
penalty	5	10	1	3

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \dots, a_n$*
- *$d_1, \dots, d_n$ deadlines for each task, $1 \le d_i \le n$*
- *$w_1, \dots, w_n > 0$ penalties if a_i is not completed by d_i*

Goal: *Find a **schedule** (i.e., permutation of tasks) that minimizes the penalties incurred*

Example:

task	a	b	c	d
deadline	1	1	4	2
penalty	5	10	1	3

A (non-optimal) schedule b a c d

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \le d_i \le n$*
- *$w_1, \ldots, w_n > 0$ penalties if a_i is not completed by d_i*

Goal: *Find a **schedule** (i.e., permutation of tasks) that minimizes the penalties incurred*

Example:

task	a	b	c	d
deadline	1	1	4	2
penalty	5	10	1	3

A (non-optimal) schedule

b	a	c	d
✓	✗	✓	✗

Application: task scheduling

Problem (Task scheduling)

Setup:

- *n unit-time tasks $a_1, \ldots, a_n$*
- *$d_1, \ldots, d_n$ deadlines for each task, $1 \le d_i \le n$*
- *$w_1, \ldots, w_n > 0$ penalties if a_i is not completed by d_i*

Goal: *Find a **schedule** (i.e., permutation of tasks) that minimizes the penalties incurred*

Example:

task	a	b	c	d
deadline	1	1	4	2
penalty	5	10	1	3

A (non-optimal) schedule

b	a	c	d
✓	✗	✓	✗

penalty: 8

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones

$$\cdots \overset{\text{late}}{a_i} \cdots \overset{\text{early}}{a_j} \cdots$$

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones

$$\cdots \underset{}{\overset{\text{late}}{a_i}} \cdots \overset{\text{early}}{a_j} \cdots$$

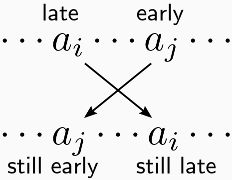
$$\cdots a_j \cdots a_i \cdots$$

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones

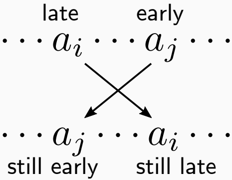


The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones



Definition

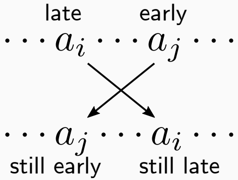
A schedule is in the **canonical form** if it's early-first and its early tasks are ordered by increasing deadlines

The canonical form of a schedule

Definition

In a schedule, a task is **early** if it finishes before its deadline; a task is **late** if it finishes after its deadline

We can transfer any schedule into the **early-first** form, i.e., early tasks before late ones



Definition

A schedule is in the **canonical form** if it's early-first and its early tasks are ordered by increasing deadlines

We can transfer any schedule into its canonical form

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early
    1.2 Sort the tasks of A in increasing deadlines
    1.3 Add late tasks in any order

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early
    1.2 Sort the tasks of A in increasing deadlines
    1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$$S = \{a_1, \ldots, a_n\}$$

$$\mathcal{I} = \{A \subseteq S : \exists \text{ a way to schedule the tasks in } A \text{ s.t. no task is late}\}$$

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

$\mathcal{I} = \{A \subseteq S : \exists$ a way to schedule the tasks in A s.t. no task is late$\}$

w : penalty

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

$\mathcal{I} = \{A \subseteq S : \exists \text{ a way to schedule the tasks in } A \text{ s.t. no task is late}\}$

w : penalty

Finding an optimal schedule $\equiv$ finding max-weighted indep. subset of M