# Chapter 2
# Greedy Algorithms

## Algorithm Theory
## WS 2013/14


## Fabian Kuhn

# Traveling Salesperson Problem (TSP)

**Input:**

- Set $V$ of $n$ nodes (points, cities, locations, sites)
- Distance function $d: V \times V \to \mathbb{R}$, i.e., $d(u, v)$: dist. from $u$ to $v$
- Distances usually symmetric, asymm. distances → asymm. TSP

**Solution:**

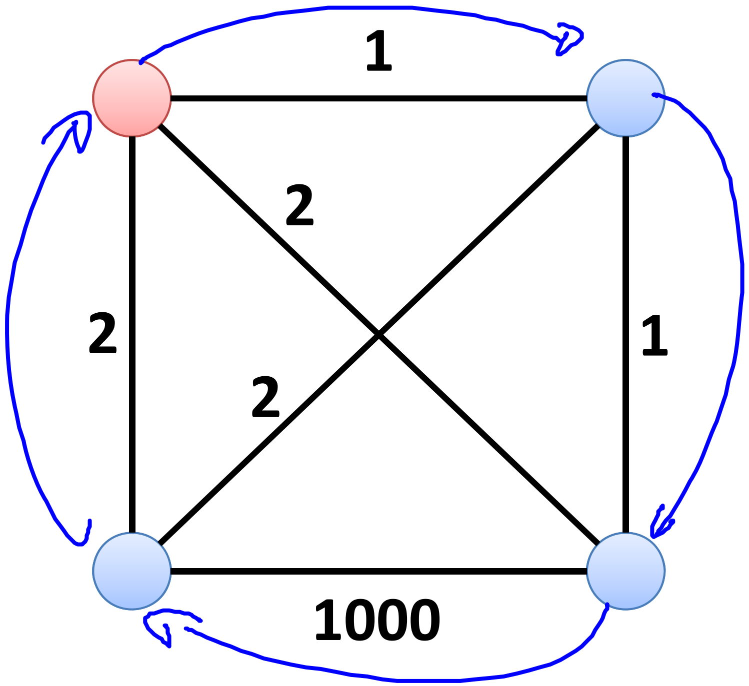
- Ordering/permutation $v_1, v_2, \dots, v_n$ of nodes
- Length of TSP path: $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$
- Length of TSP tour: $d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$

**Goal:**

- Minimize length of TSP path or TSP tour

# Nearest Neighbor (Greedy)

- Nearest neighbor can be arbitrarily bad, even for TSP paths



# TSP Variants

- Asymmetric TSP
  - arbitrary non-negative distance/cost function
  - most general, nearest neighbor arbitrarily bad
  - NP-hard to get within any bound of optimum

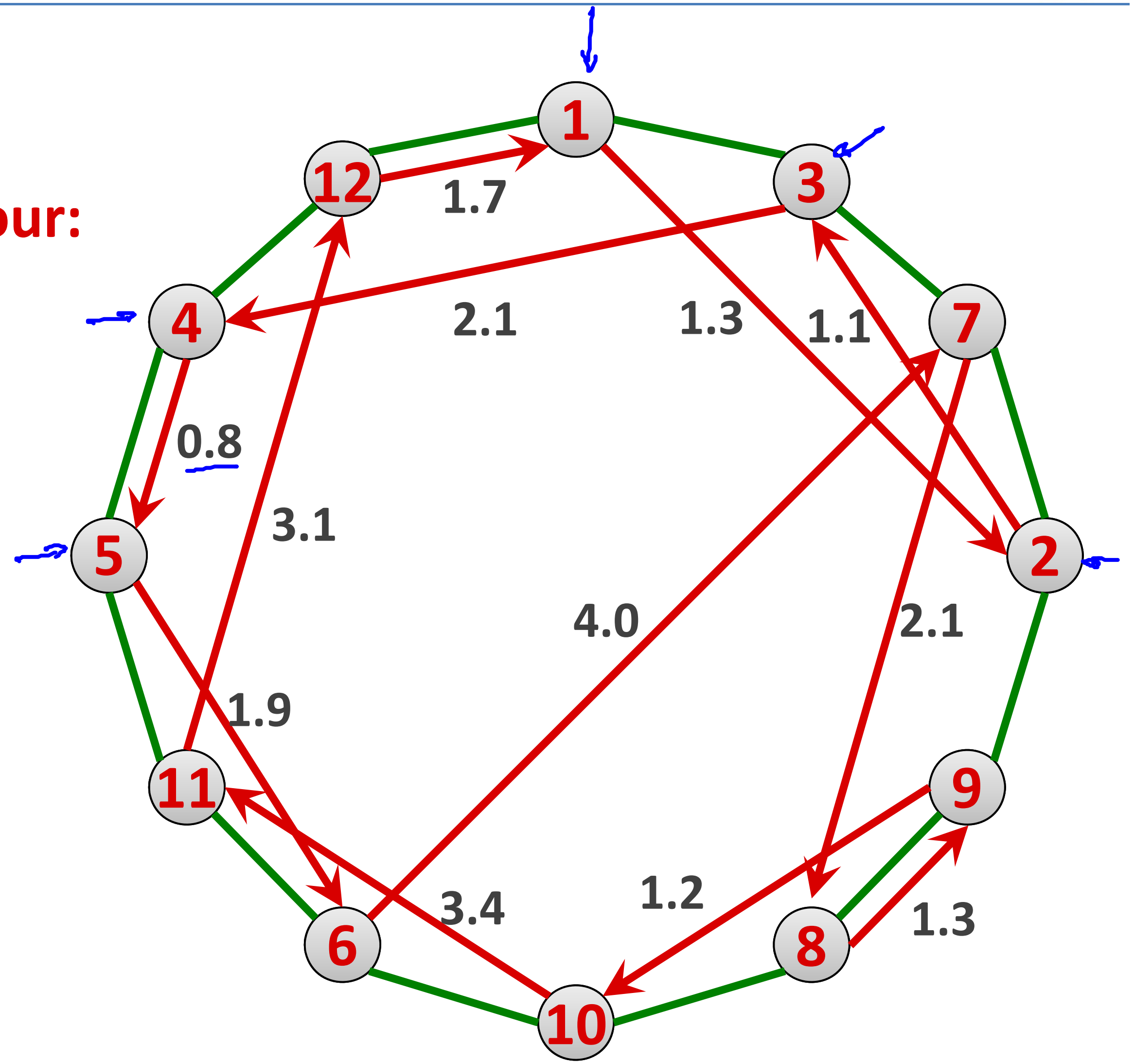
- Symmetric TSP
  - arbitrary non-negative distance/cost function
  - nearest neighbor arbitrarily bad
  - NP-hard to get within any bound of optimum

- $\boxed{\text{Metric TSP}}$
  - distance function defines metric space: symmetric, non-negative, triangle inequality: $d(u,v) \leq d(u,w) + d(w,v)$
  - possible to get close to optimum (we will later see factor $^{3}/_{2}$)
  - what about the nearest neighbor algorithm?

# Metric TSP, Nearest Neighbor

**Optimal TSP tour:**

**Nearest-Neighbor TSP tour:**



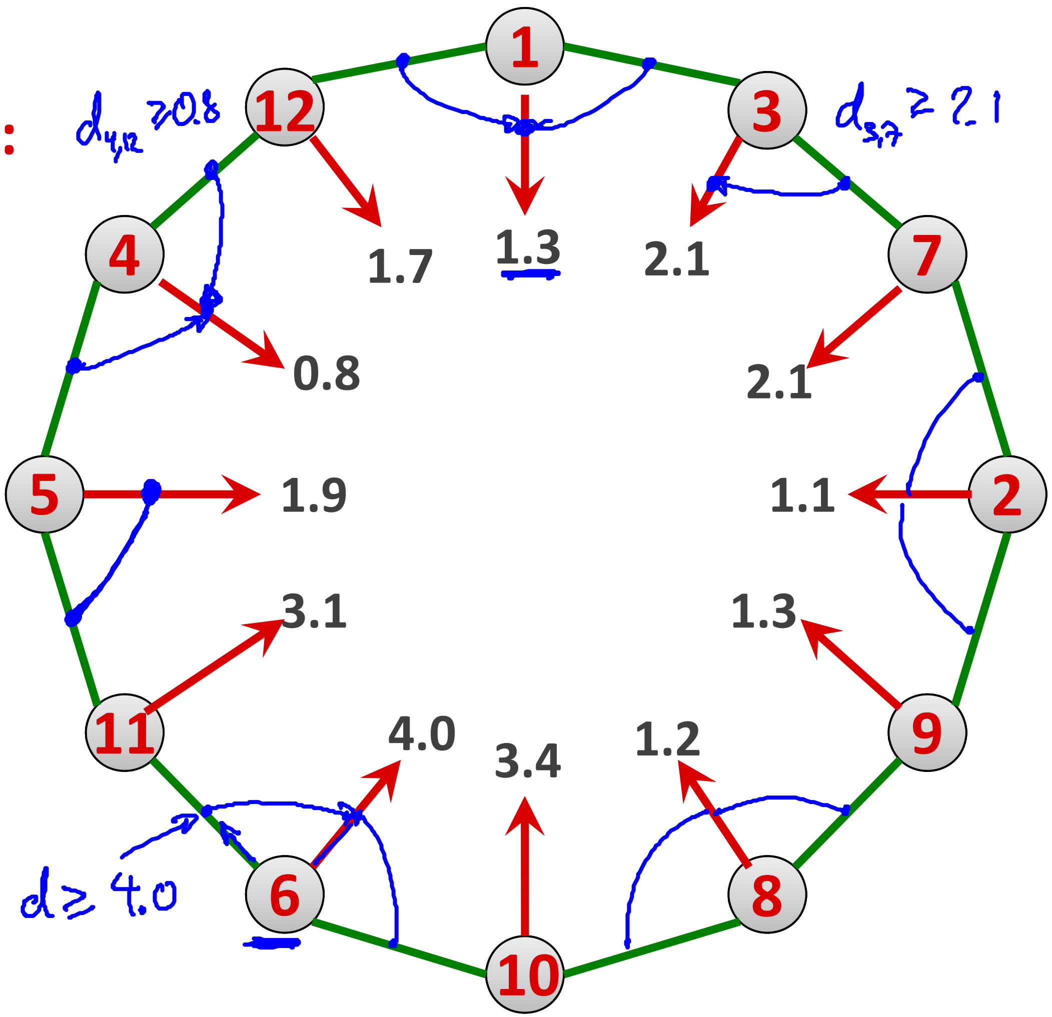
# Metric TSP, Nearest Neighbor

*n nodes*

**Optimal TSP tour:**

**Nearest-Neighbor TSP tour:**

cost = 24




marked red $\leq$ green

$\leq$ opt TSP length

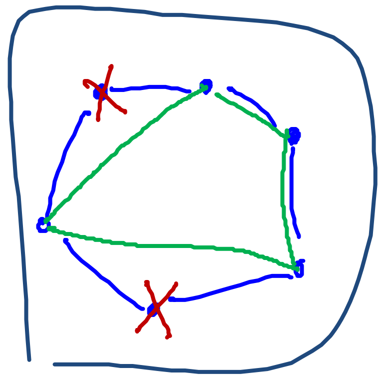
# Metric TSP, Nearest Neighbor

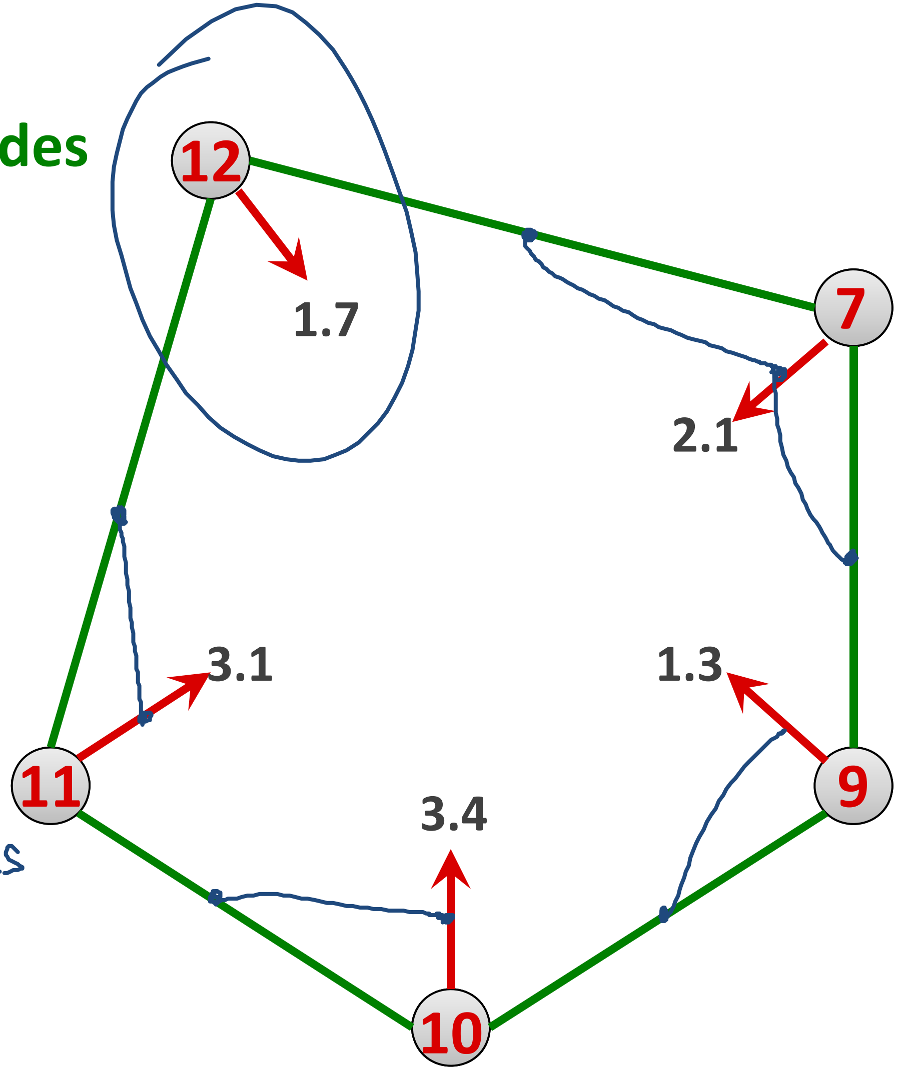
**Triangle Inequality:**

**optimal tour on remaining nodes**

$\leq$

**overall optimal tour**






mark $\geq 1/2$ remaining red edges

marked red $\leq$ green

# Metric TSP, Nearest Neighbor

Analysis works in phases:

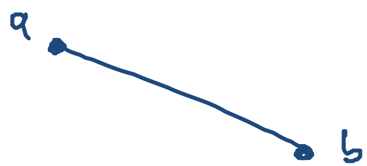
green red

- In each phase, assign each optimal edge to some greedy edge
  - Cost of greedy edge $\leq$ cost of optimal edge
- Each greedy edge gets assigned $\leq 2$ optimal edges
  - At least half of the greedy edges get assigned
- At end of phase:

  Remove points for which greedy edge is assigned

  Consider optimal solution for remaining points
- **Triangle inequality:** remaining opt. solution $\leq$ overall opt. sol.

- Cost of greedy edges assigned in **each phase $\leq$ opt. cost**
- **Number of phases $\leq \log_2 n$**
  - +1 for last greedy edge in tour



# Metric TSP, Nearest Neighbor

- Assume:

  NN: cost of greedy tour, OPT: cost of optimal tour

- We have shown:

$$\frac{\text{NN}}{\text{OPT}} \le \underbrace{1 + \log_2 n}_{\text{approximation ratio}}$$

- Example of an **approximation algorithm**

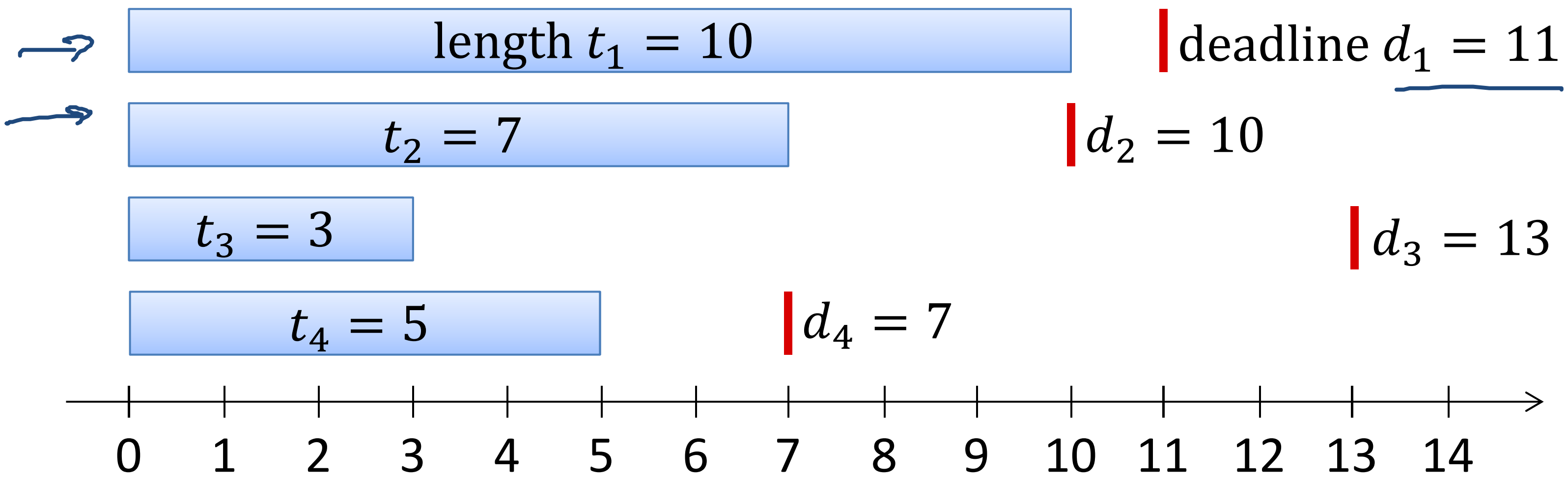
- We will later see a $^3/_2$-approximation algorithm for metric TSP

$\frac{S}{O} \le \frac{3}{2}$ approx. ratio Alg computes solution $S$

Opt. tour $O$

# Back to Scheduling

- Given: $n$ requests / jobs with deadlines:




- Goal: schedule all jobs with minimum lateness $L$
  - Schedule: $s(i)$, $f(i)$: start and finishing times of request $i$
    Note: $f(i) = s(i) + t_i$

$s(i) + t_i = f(i)$

Lateness of $i$

$\max\{0, f(i) - d_i\}$

- Lateness $L := \max\left\{0, \max_i \{f(i) - d_i\}\right\}$
  - largest amount of time by which some job finishes late

- Many other natural objective functions possible…

# Greedy Algorithm?

**Schedule jobs in order of increasing length?**

- Ignores deadlines: seems too simplistic…
- E.g.:

0

$t_1 = 10$ | deadline $d_1 = 10$

$t_2 = 2$ 98 2 $d_2 = 100$

Schedule: $t_2 = 2$ $t_1 = 10$

L=2

**Schedule by increasing slack time?**

- Should be concerned about slack time: $d_i - t_i$

$t_1 = 10$ | deadline $d_1 = 10$

$t_2 = 2$ | $d_2 = 3$

L=9

Schedule: $t_1 = 10$ $t_2 = 2$

# Greedy Algorithm

**Schedule by earliest deadline?**

- Schedule in increasing order of $d_i$
- Ignores lengths of jobs: too simplistic?

- Earliest deadline is optimal!

$s(1)=0,\ f(1)=t_1,\ s(2)=t_1,\ f(2)=t_1+t_2, \dots$

**Algorithm:**

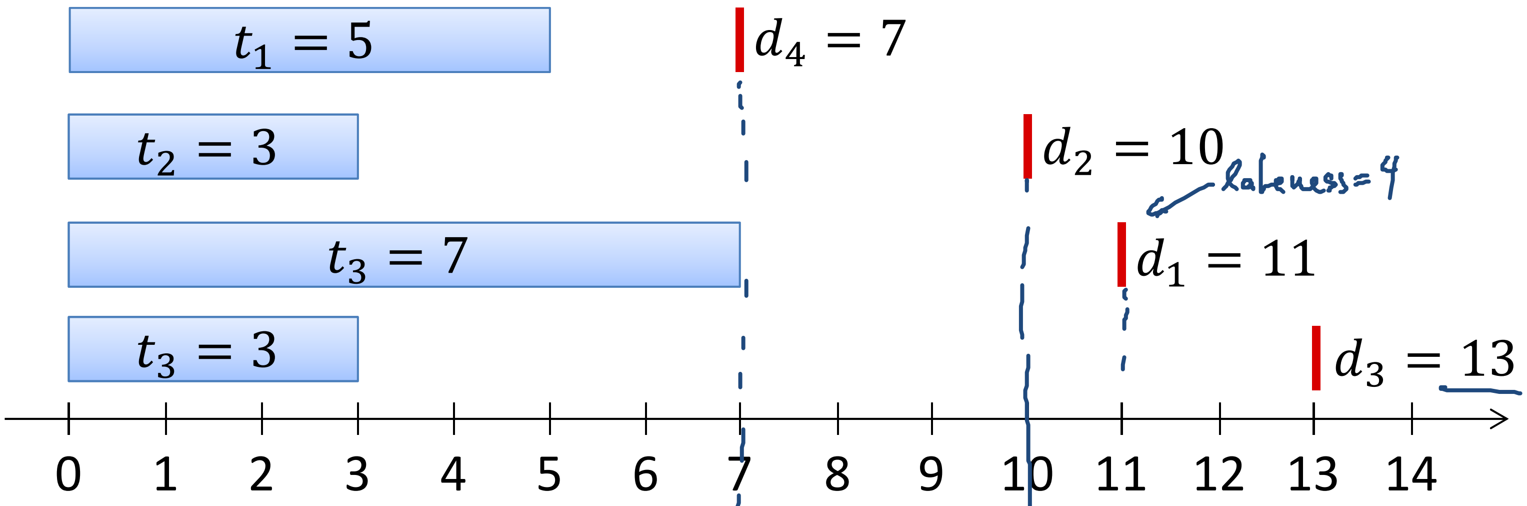
- Assume jobs are reordered such that $d_1 \leq d_2 \leq \cdots \leq d_n$
- Start/finishing times:


  - First job starts at time $s(1) = 0$
  - Duration of job $i$ is $t_i$: $f(i) = s(i) + t_i$
  - No gaps between jobs: $s(i+1) = f(i)$

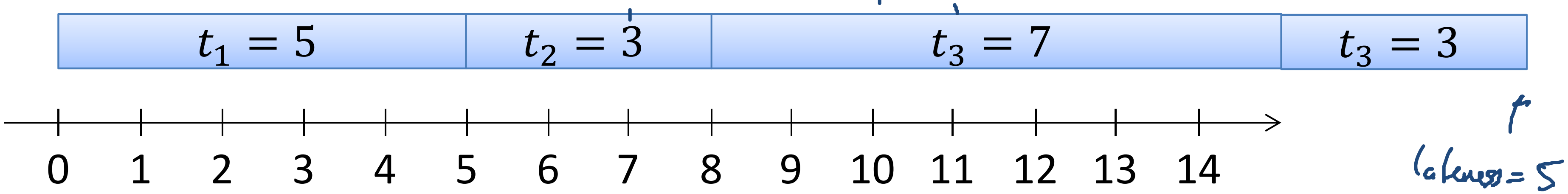
  (idle time: gaps in a schedule → alg. gives schedule with no idle time)

# Example

**Jobs ordered by deadline:**



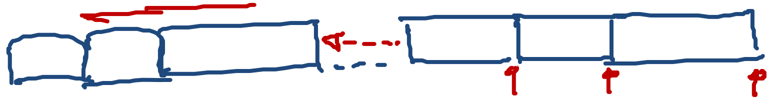

**Schedule:**



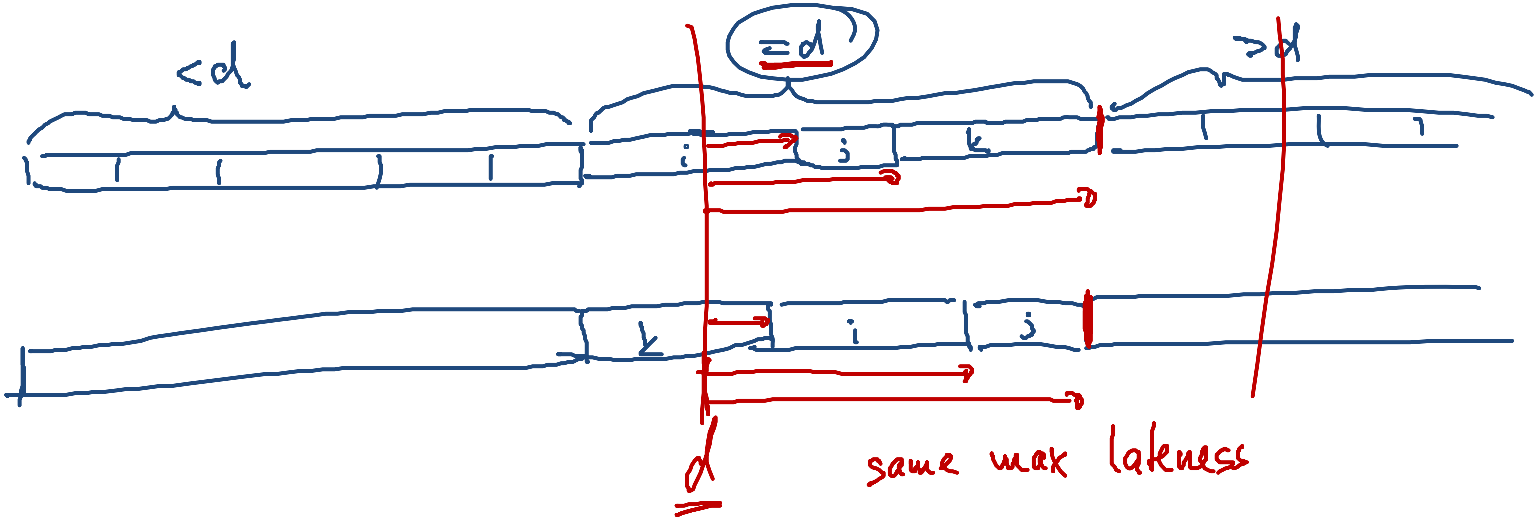

**Lateness:** job 1: 0, job 2: 0, job 3: 4, job 4: 5

# Basic Facts

1. There is an optimal schedule with no idle time
   – Can just schedule jobs earlier…



2. Inversion: Job $i$ scheduled before job $j$ if $d_i > d_j$
   Schedules with no inversions have the same maximum lateness



# Earliest Deadline is Optimal

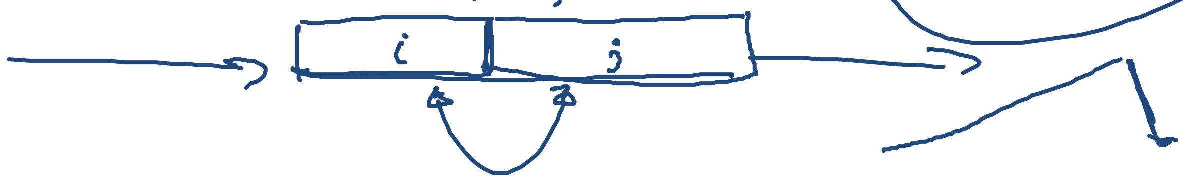
**Theorem:**
There is an optimal schedule $\mathcal{O}$ with no inversions and no idle time.

**Proof:**

- Consider optimal schedule $\mathcal{O}'$ with no idle time
- If $\mathcal{O}'$ has inversions, $\exists$ pair $(i, j)$, s.t. $i$ is scheduled immediately before $j$ and $d_j < d_i$

$d_1' \leq d_2' \leq d_3' \ldots$ $\quad d_i' < d_2'$

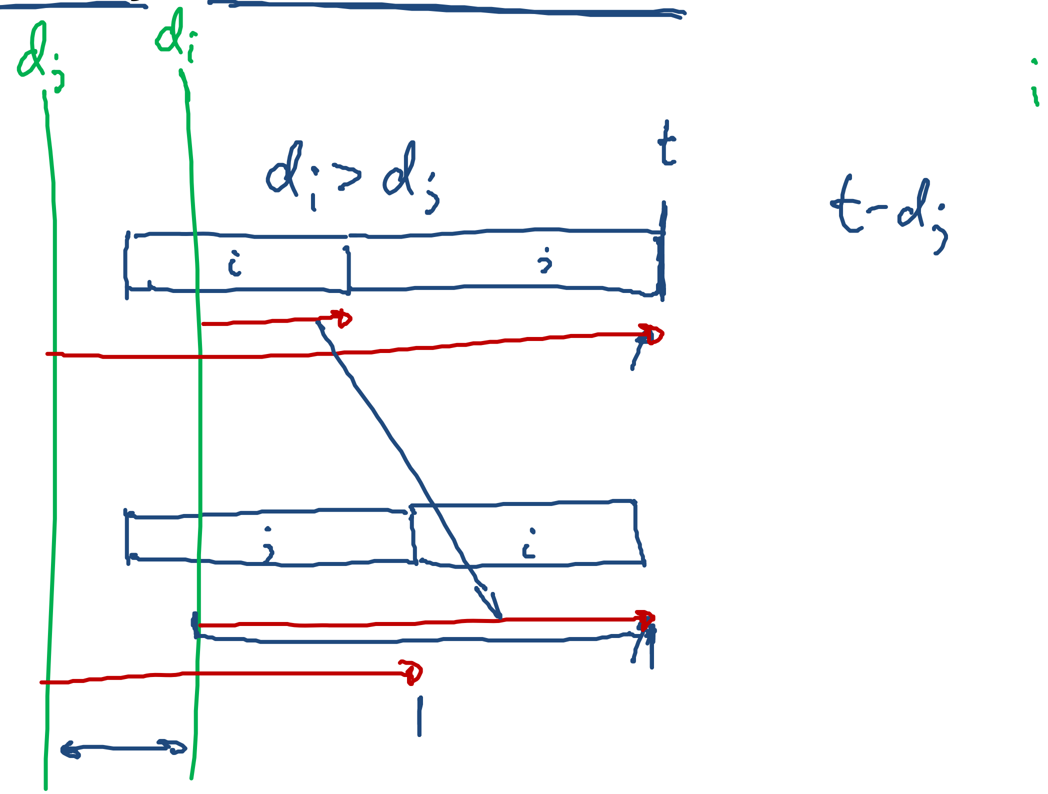
$d_i > d_j$




- Claim: Swapping $i$ and $j$ gives schedule with
  1. Less inversions
  2. Maximum lateness no larger than in $\mathcal{O}'$

# Earliest Deadline is Optimal

**Claim:** Swapping $i$ and $j$: maximum lateness no larger than in $\mathcal{O}'$



# Exchange Argument

- General approach that often works to analyze greedy algorithms

- Start with any solution
- Define basic exchange step that allows to transform solution into a new solution that is not worse
- Show that exchange step move solution closer to the solution produced by the greedy algorithm
- Number of exchange steps to reach greedy solution should be finite…

# Another Exchange Argument Example

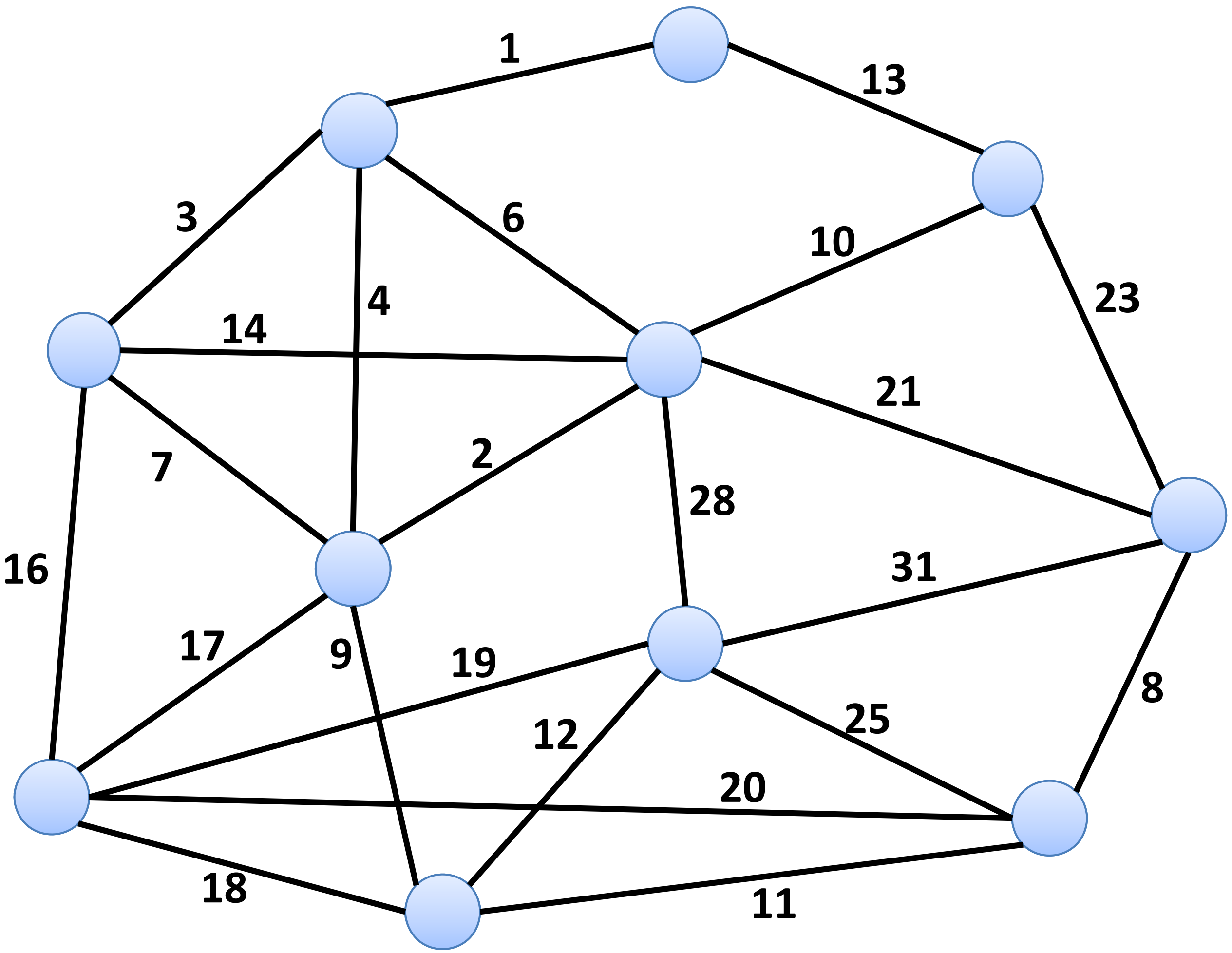
- **Minimum spanning tree (MST)** problem
  - Classic graph-theoretic optimization problem

- **Given**: weighted graph
- **Goal**: spanning tree with min. total weight

- Several greedy algorithms work

- Kruskal's algorithm:
  - Start with empty edge set
  - As long as we do not have a spanning tree:
    **add minimum weight edge that doesn't close a cycle**

# Kruskal Algorithm: Example

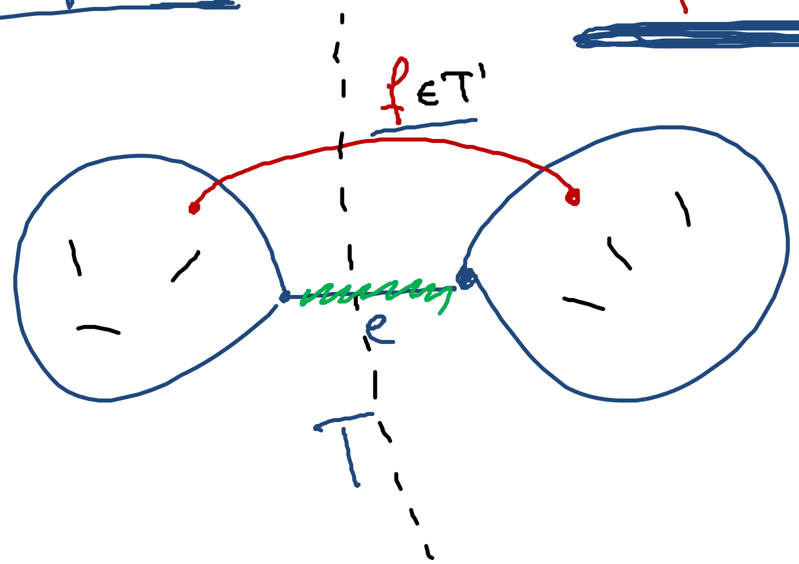


# Kruskal is Optimal

- Basic exchange step: swap to edges to get from tree $T$ to tree $T'$
  - Swap out edge not in Kruskal tree, swap in edge in Kruskal tree
  - Swapping does not increase total weight
- For simplicity, assume, weights are unique:

$T$ : any sp. tree $\quad T'$ : Kruskal tree

$e$ : edge in $T \setminus T'$ $\qquad w(f) < w(e)$

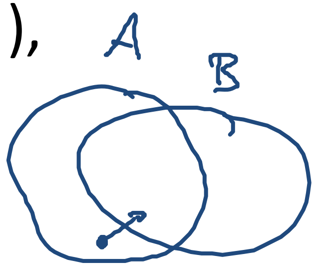


# Matroids

- Same, but more abstract…

**Matroid:** **pair $(\boldsymbol{E}, \boldsymbol{I})$**

- $\boldsymbol{E}$: set, called the **ground set**
- $\boldsymbol{I}$: finite family of finite subsets of $E$ (i.e., $I \subseteq 2^E$), called **independent sets**




$(E, I)$ needs to satisfy 3 properties:

1. Empty set is independent, i.e., $\emptyset \in \boldsymbol{I}$ (implies that $I \neq \emptyset$)
2. **Hereditary property**: For all $A \subseteq I$ and all $A' \subseteq A$,

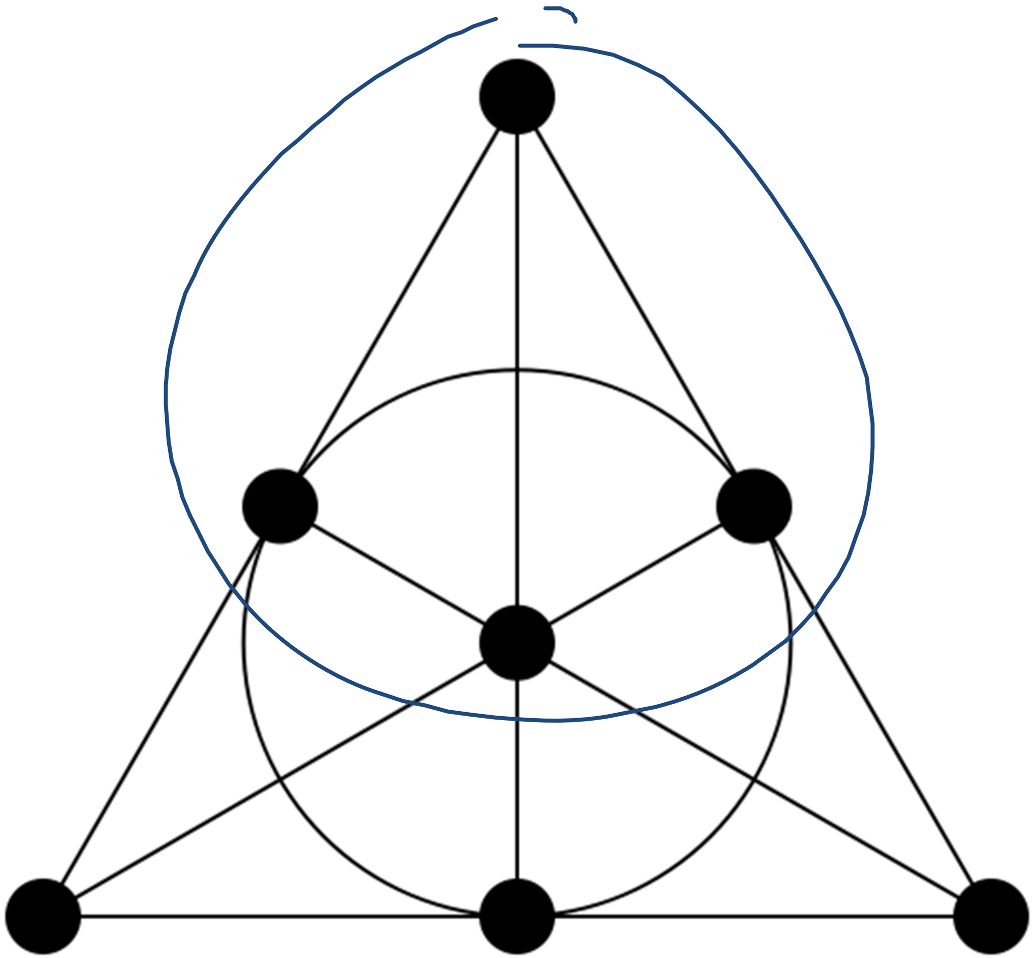
   **if $\boldsymbol{A} \in \boldsymbol{I}$, then also $\boldsymbol{A}' \in \boldsymbol{I}$**

3. **Augmentation / Independent set exchange property:**
   If $A, B \in I$ and $|A| > |B|$, there exists $x \in A \setminus B$ such that

   $$\mathbf{B}' := \boldsymbol{B} \cup \{\boldsymbol{x}\} \in \boldsymbol{I}$$

# Example

- Fano matroid:
  - Smallest finite projective plane of order 2…



# Matroids and Greedy Algorithms

**Weighted matroid**: each $e \in E$ has a weight $w(e) > 0$

**Goal:** find maximum weight independent set

**Greedy algorithm:**

1. Start with $S = \emptyset$
2. Add max. weight $e \in E \setminus S$ to $S$ such that $S \cup \{e\} \in I$

**Claim:** greedy algorithm computes optimal solution

# Greedy is Optimal

any independent set

- $S$: greedy solution $\qquad$ $A$: any other solution

$\underline{|S| \geq |A|}$ $\qquad$ if $|S| < |A| \rightarrow \exists x \in A \setminus S$ :
$S \cup \{x\} \in I$

$S \neq A$ $\qquad$ $w(x_1) \geq w(x_2) \geq \ldots$

$\underline{w(x_k) \geq w(x_k')}$

$S$: $\quad \overbrace{x_1, x_2, x_3, \ldots, x_{k-1}, x_k}, \ldots$

$A$: $\quad \underbrace{x_1, x_2, x_3, \ldots, x_{k-1}, x_k'}, \ldots$