# EECS 336: Lecture 14: Introduction to Algorithms

**Greedy Algorithms: Interval Scheduling**

**Reading:** 4.1

**Last Time:**

- CIRCUIT-SAT $\leq_p$ LE3-SAT $\leq_p$ 3-SAT

**Today:**

- Greedy Algorithms
- Interval Scheduling

---

**Exercise 14.2: Greedy Scheduling**

**Setup:** Consider scheduling jobs:

- job 1 needs to run from time 1 to 2
- job 2 needs to run from time 1 to 4
- job 3 needs to run from time 3 to 6
- job 4 needs to run from time 5 to 6

But only one job can run one job at a time, e.g., you can run both 1 and 3 for but you cannot run both 1 and 2 (since they overlap at times 1 and 2).

**Question:**

- What is the maximum number of jobs that can be simultaneously scheduled?

---

## Greedy Algorithms

- build solution in steps.
- each step myopically optimal
- hard part: prove final solution is optimal

**Question:** For what problems are greedy algorithms optimal?

## Interval Scheduling

"sharing a single resource"

- $n$ jobs
- one machine
- requests: job $i$ needs machine between $s(i)$ and $f(i)$.

**Goal:** schedule to maximize # of jobs scheduled.

**Examples:** Greedy by . . .

- "start time"

```
     ---  ---  ---  ---  ---
   --------------------------
```

- "smallest size"

```
  ---------  ---------
         ------
```

- "fewest incompatibilities"

```
  ---- ---- ---- -----
     ---  ---  ---
     ---       ---
     ---       ---
```

**Note:** important to be able to find counterexamples quickly.

### Greedy Algorithm for Interval Scheduling

**Idea:** scheduling the earliest finish time first, leave the least constraint on remaining schedule.

**Def:** jobs $i$ and $j$ are

- **incompatible** if $[s(i), f(i)] \vee [s(j), f(j)] \neq \emptyset$
- otherwise **compatible.**
- set $S$ is **compatible** if all $i, j \in S$ are compatible.

**Examples:** incompatible jobs

```
-----      or    -----   or    -----
  -----         -----        ---------
```

**Algorithm:** Greedy by Min. Finish Time

1. $S = \emptyset$
2. Sort jobs by increasing finish time.
3. For each job $j$ (in sorted order):
   - if $j$ if compatible with $S$
     - schedule: $j : S \leftarrow S \cup \{j\}$
   - else discard $j$

## Analysis

### Runtime

$$\begin{aligned} T(n) &\underbrace{\leq n \log n}_{\text{sort}} + \overbrace{\sum_j j}^{\text{check compatibility}} \\ &\approx n \log n + n^2 \\ &= O(n^2). \end{aligned}$$

**Idea:** Job $j$ in alg. is compatible if it is compatible with last scheduled job.

$$\begin{aligned} T(n) &= n \log n + n \\ &= \Theta(n \log n) \end{aligned}$$

---

### Exercise 14.1: Scheduling, Revisited

**Setup:** Consider running the greedy algorithm to schedule jobs:

- job 1 needs to run from time 1 to 2
- job 2 needs to run from time 1 to 4
- job 3 needs to run from time 3 to 6
- job 4 needs to run from time 5 to 6

(Recall: one job can run one job at a time.)

**Question:**

- Which jobs are scheduled?

---

**Correctness**

"schedule is compatible and optimal"

**Lemma 1:** schedule of algorithm is compatible

**Proof:** (by induction, straightfoward)

**Def:**

- let $i_1, ..., i_k$ be jobs scheduled by greedy
- let $j_1, ..., j_m$ be jobs scheduled by OPT

**Goal:** show $k = m$.

**Approach:** "Greedy Stays Ahead"

**Lemma 2:** for $r \leq k, f(i_r) \leq f(j_r)$

**Proof:** (induction on $r$)

base case: $r = 0$

- add dummy job 0 with $s(0) = f(0) = -\infty$
- only change: OPT and GREEDY schedule dummy
- so $f(i_0) = f(j_0)$

inductive hypothesis: $f(i_r) \leq f(j_r)$

inductive step:

- Let $I = \{\text{jobs starting after } f(i_r)\}$

  $J = \{\text{jobs starting after } f(j_r)\}$
- IH $\implies J \subseteq I$
- GREEDY $\implies f(i_{r+1}) = \min_{j \in I} f(j)$

  $\leq \min_{j \in J} f(j)$

  $\leq f(j_{r+1})$

**Theorem:** Greedy alg. is optimal

**Proof:** (by contradiction)

- OPT has job $j_{k+1}$ but greedy terminates at $k$.
- lemma 2 (with $r = k$)

$$\implies f(i_k) \leq f(j_k) \tag{1}$$

- $j_{k+1}$ is compatible with $j_k$

$$\implies f(j_k) \leq s(k_{k+1}) \tag{2}$$

- (1) & (2)

  $\implies f(i_k) \leq s(j_{k+1})$

  $\implies j_{k+1}$ is compatible with $i_k$

  $\implies$ alg doesn't terminate at $k$

## Greedy by Value

"to pick a *feasible* set with maximum total value"

**Algorithm:** Greedy-by-Value

1. $S = \emptyset$
2. Sort elts by decreasing value.
3. For each elt $e$ (in sorted order):

    if $\{e\} \cup S$ is feasible

        add $e$ to $S$

    else discard $e$.

### Minimum Spanning Tree

"maintaining minimal connectivity in a network, e.g., for broadcast"

**input:**

- graph $G = (V, E)$
- costs $c(e)$ on edges $e \in E$

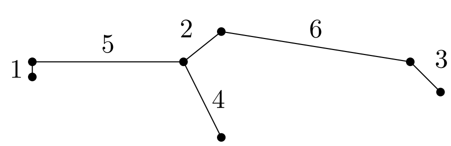
**output:** *spanning tree* with minimum total cost.

**Def:** a **spanning tree** of a graph $G = (V, E)$ is $T \subseteq E$ s.t.

- $(V, T)$ is connected.
- $(V, T)$ is acyclic.

**Note:** Greedy-by-Value = Kruskal's Alg

**Example:**




### Runtime

$\Theta(m \log n)$

- $\Theta(m \log n)$ to sort.
- check connectivity with *union-find* data structure

amortized $O(\log^* n)$ runtime per operation.

(recall $l = \log^* n \Leftrightarrow n = \underbrace{2^{2^{2^2}}}_{l \text{ times}}$ )

total $O(m \log^* n)$ runtime.

## Correctness

"output is tree and has minimum cost"

**Goal:** understand why greedy-by-value works.

**Lemma 1:** Greedy outputs a forest.

**Proof:** Induction.

**Lemma 2:** if $G$ is connected, Greedy outputs a tree.

**Proof:** (by contradiction)

**Theorem:** Greedy-by-Value is optimal for MSTs

**Approach:** "greedy stays ahead"

**Proof:** (by contradiction of first mistake)

- Greedy and OPT have $n-1$ edges (Fact 1)
- Let $I = \{i_1, ..., i_{n-1}\}$ be elt's of Greedy. (in order)
- Assume for contradiction: $c(I) > c(J)$
- Let $r$ be first index with $c(j_r) < c(i_r)$
- Let $I_{r-1} = \{i_1, ..., i_{r-1}\}$
- $|I_{r-1}| < |J_r|$ & Augmentation Lemma

  $\Rightarrow$ exists $j \in J_r \setminus I_{r-1}$

  such that $I_{r-1} \cup \{j\}$ is acyclic.
- Suppose $j$ considered after $i_k$ ($k \leq r-1$)
- $I_k \subseteq I_{r-1}$

  $\Rightarrow I_k \cup \{j\} \subseteq I_{r-1} \cup \{j\}$
- $I_{r-1} \cup \{j\}$ acyclic & Fact 2

  $\Rightarrow$ all subsets are acyclic

  $\Rightarrow I_k \cup \{j\}$ acyclic

  $\Rightarrow j$ should have been added.

**Structural Observations about Forests**

**Def:** $G' = (V, E')$ is a subgraph of $G = (V, E)$ if $E' \subseteq E$.

**Def:** An acyclic undirected graph is a **forest**

**Fact 1:** and MST on $n$ vertices has $n-1$ edges.

**Lemma 1:** If $G = (V, F)$ is a forest with $m$ edges then it has $n-m$ connected components.

**Proof:** Induction (on number of edges)

case case: 0 edges, $n$ CCs.

IH: assume true for $m$.

IS: show true for $m+1$.

- IH $\implies n-m$ CCs
- add new edges.
- must not create cycle

$\Rightarrow$ connects two connected components.

$\Rightarrow$ these 2 CCs become 1 CC.

$\Rightarrow n-m-1$ CCs.

**QED**

**Lemma 2:** (Augmentation Lemma) If $I, J \subset E$ are forests and $|I| < |J|$ then exists $e \in J \setminus I$ such that $I \cup \{e\}$ is a forest.

**Proof:**

Lemma 1

$\Rightarrow$ # CCs of $(V, I) >$ # CCs of $(V, J) \geq$ # CCs of $(V, I \cup J)$

$\Rightarrow$ add elements $e \in J$ to $I$ until # CCs change.

[PICTURE]

$\Rightarrow (V, I \cup \{e\})$ is acyclic.

**Fact 2:** subgraphs of acyclic graphs are acyclic