

## Lecture 12: Independent Sets and Interval Scheduling

Harvard SEAS - Fall 2023

2023-10-17

## 1 Announcements

Recommended Reading: CLRS Sec 16.1–16.2

- Sender-Receiver Exercise today!

## 2 Loose ends of Lec 11 : 2-colorable graphs

**Theorem 2.1.** *If  $G$  is a connected 2-colorable graph, then  $\text{BFSColoring}(G)$  will color  $G$  using 2 colors.*

*Proof.* Let  $f^*$  be a 2-coloring of  $G$ . We may assume that  $f^*(v_0) = 0$  without loss of generality (why?). Let  $f$  be the coloring of  $G$  found by  $\text{BFSColoring}(G)$ . We argue by (strong) induction on  $i$  that  $f(v_i) = f^*(v_i)$  for  $i = 0, \dots, n-1$ .

For  $i = 0$ , we observe that  $\text{BFSColoring}(G)$  sets  $f(v_0) = 0$ . Now for  $i > 0$ , we will argue that  $f^*$  satisfies the same rule used to construct  $f$ , namely:

$$f^*(v_i) = \min \{c \in \mathbb{N} : c \neq f^*(v_j) \ \forall j < i \text{ s.t. } \{v_i, v_j\} \in E\}. \quad (1)$$

In other words, the value of  $f^*$  at  $v_i$  is “forced” by its values at the previously assigned vertices  $v_j$ . Since  $f^*$  is a valid 2-coloring, the value  $c = f^*(v_i)$  satisfies the condition  $c \neq f^*(v_j)$  for all  $j < i$  such that  $\{v_i, v_j\} \in E$  automatically holds. If  $f^*(v_i) = 0$ , then it is certainly the minimum value of  $c$  satisfying this condition. If  $f^*(v_i) = 1$ , we note that that by the definition of BFS, there is a previous vertex  $v_j$  (with  $j < i$ ) with an edge to  $v_i$ . Since  $f^*$  is a valid 2-coloring, we must have  $f^*(v_j) = 0$ . So  $c = 0$ , does not satisfy the condition in Equation (1), and hence  $c = 1$  must be the minimum value satisfying the condition.

By the definition of  $\text{BFSColoring}(G)$ , we have

$$f(v_i) = \min \{c \in \mathbb{N} : c \neq f(v_j) \ \forall j < i \text{ s.t. } \{v_i, v_j\} \in E\} \quad (2)$$

By our (strong) induction hypothesis, the right-hand sides of (1) and (2) are equal, and thus  $f(v_i) = f^*(v_i)$ .  $\square$

**Corollary 2.2.** *Graph 2-Coloring can be solved in time  $O(n + m)$ .*

*Proof.* We can partition  $G$  into connected components in time  $O(n + m)$ . Then, for each connected component we can use  $\text{BFSColoring}$  on each component, which takes total time  $O(n + m)$ .  $\square$

### 3 Definitions

In the Sender-Receiver Exercise, you've seen the definition of independent sets, which are closely related to graph colorings:

**Definition 3.1.** Let  $G = (V, E)$  be a graph. An *independent set* in  $G$  is a subset  $S \subseteq V$  such that there are no edges entirely in  $S$ . That is,  $\{u, v\} \in E$  implies that  $u \notin S$  or  $v \notin S$ .

A proper  $k$ -coloring of a graph  $G$  is equivalent to a partition of  $V$  into  $k$  independent sets (each color class should be an independent set).

When we have a graph  $G = (V, E)$  representing conflicts, instead of partitioning  $V$  into a small number of conflict-free subsets (as coloring would), it is sometimes useful to instead find a single, large conflict-free subset. This gives rise to the following computational problem:

**Input** : A graph  $G = (V, E)$   
**Output** : An independent set  $S \subseteq V$  in  $G$  of maximum size

**Computational Problem** Independent Set

**Example:** Throwing a big party where everyone will get along.

Like with graph coloring, we can try a greedy algorithm for Independent Set:

```

1 GreedyIndSet( $G$ )
   Input : A graph  $G = (V, E)$ 
   Output : A "large" independent set in  $G$ 
2 Choose an ordering  $v_0, v_1, v_2, \dots, v_{n-1}$  of  $V$ ;
3  $S = \emptyset$ ;
4 foreach  $i = 0$  to  $n - 1$  do
5   | if  $\forall j < i$  s.t.  $\{v_i, v_j\} \in E$  we have  $v_j \notin S$  then  $S = S \cup \{v_i\}$ ;
6 return  $S$ 

```

And, similarly to coloring, we can only prove fairly weak bounds on the performance of the greedy algorithm in general:

**Theorem 3.2.** For every graph  $G$  with  $n$  vertices and  $m$  edges,  $\text{GreedyIndSet}(G)$  can be implemented in time  $O(n + m)$  and outputs an independent set of size at least  $n/(d_{\max} + 1)$ , where  $d_{\max}$  is the maximum vertex degree in  $G$ .

*Proof.*

Omitted (and possibly covered in section). □

However, when there is more structure in the conflict graph, a careful ordering for the greedy algorithm can yield an optimal solution. An example of such structure comes from the Interval Scheduling problem we saw in the first lecture:

**Input** : A collection of intervals  $[a_0, b_0], \dots, [a_{n-1}, b_{n-1}]$ , where each  $a_i, b_i \in \mathbb{R}$  and  $a_i \leq b_i$   
**Output** : YES if the intervals are disjoint (for all  $i \neq j$ ,  $[a_i, b_i] \cap [a_j, b_j] = \emptyset$ )  
 NO otherwise

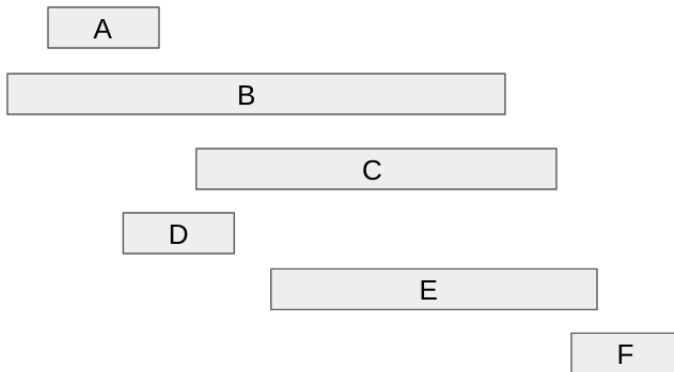
**Computational Problem** IntervalScheduling-Decision

We saw that we could solve this problem in time  $O(n \log n)$  by reduction to Sorting. However, if the answer is NO, we might be satisfied by trying to schedule *as many intervals as possible*:

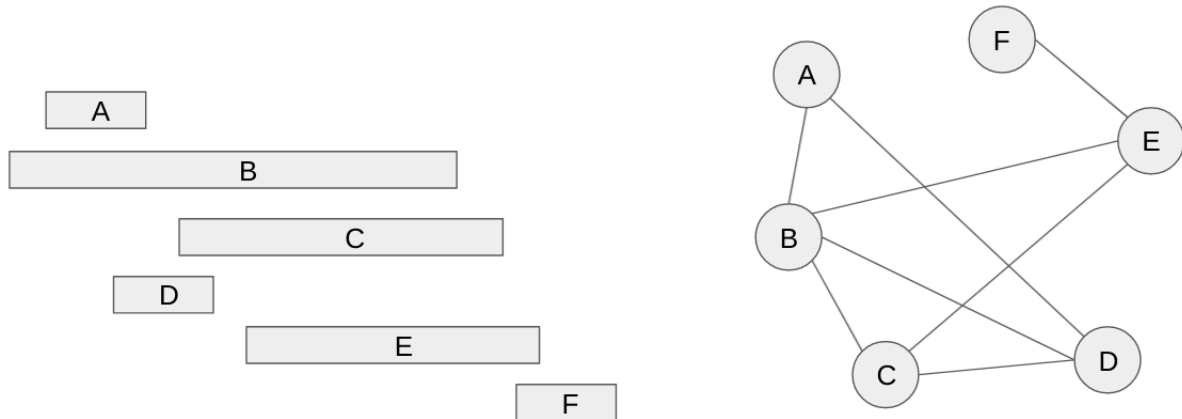
<b>Input</b>	: A collection of intervals $[a_0, b_0], \dots, [a_{n-1}, b_{n-1}]$ , where each $a_i, b_i \in \mathbb{Q}$ and $a_i \leq b_i$
<b>Output</b>	: A maximum-size subset $S \subseteq [n]$ such that $\forall i \neq j \in S, [a_i, b_i] \cap [a_j, b_j] = \emptyset$ .

**Computational Problem** IntervalScheduling-Optimization

**Example:**



**Q:** How can we model IntervalScheduling-Optimization as an Independent Set problem?



**A:** We represent each interval as a vertex, and we place an edge between two vertices (i.e. intervals) if they conflict. Then an independent set is exactly a set of intervals which have no conflicts, so maximizing the size of this is equivalent to finding the largest set of conflict-free intervals.

With this graph-theoretic modelling, we can instantiate `GreedyIndSet()` for IntervalScheduling-

Optimization:

```

1 GreedyIntervalScheduling( $x$ )
  Input    : A list  $x$  of  $n$  intervals  $[a, b]$ , with  $a, b \in \mathbb{Q}$ 
  Output   : A “large” subset of the input intervals that are disjoint from each other
2 Choose an ordering of the input intervals  $[a_0, b_0], [a_1, b_1], \dots, [a_{n-1}, b_{n-1}]$ ;
3  $S = \emptyset$ ;
4 foreach  $i = 0$  to  $n - 1$  do
5   | if  $\forall j < i$  s.t.  $j \in S$  we have  $[a_j, b_j] \cap [a_i, b_i] = \emptyset$  then  $S = S \cup \{i\}$ ;
6 return  $S$ 

```

**Q:** What ordering of the input intervals should we use?

**A:** Want to first assign the intervals with the earliest *end* time.

**Theorem 3.3.** *If the input intervals are sorted by increasing order of end time  $b_i$ , then we have that GreedyIntervalScheduling( $x$ ) will find an optimal solution to IntervalScheduling-Optimization, and can be implemented in time  $O(n \log n)$ .*

*Proof.*

Intuitively, for any interval scheduling problem (black in Figure 1), we can modify any solution  $(i_0^*, i_1^*, \dots, i_{\ell-1}^*)$  to it (gray boxes) into the greedy solution  $(i_0, i_1, \dots, i_{k-1})$  (white) by “smushing it left”, replacing the first  $j$  intervals of our solution with the first  $j$  intervals of the greedy solution to get a valid solution  $(i_0, i_1, \dots, i_{j-1}, i_j^*, \dots, i_{\ell-1}^*)$ . Also,  $k \geq \ell$ : If not, then Greedy would pick one more interval, since  $i_\ell^*$  would be valid.

Formally, let  $S^* = \{i_0^* \leq i_1^* \leq \dots \leq i_{k^*-1}^*\}$  be an optimal solution to Interval Scheduling (where we say that  $i < i'$  for intervals  $i$  and  $i'$  if  $i$  ends before  $i'$  begins). Then let  $S = \{i_0 \leq i_1 \leq \dots \leq i_{k-1}\}$  be the solution found by the greedy algorithm. Recall that  $b_{i_j}$  is the endtime of interval  $i_j$  (and above we sort both solutions on end time).

**Claim 3.4** (greedy stays ahead). *For all  $j \in \{0, \dots, k^* - 1\}$ , we have:*

1.  $j < k$ , i.e. the Greedy Algorithm schedules at least  $j + 1$  intervals, and
2.  $b_{i_j} \leq b_{i_j^*}$ , i.e. the  $j$ 'th interval scheduled by the Greedy algorithm ends no later than the  $j$ 'th interval scheduled by the optimal solution.

*Proof.* For the  $j = 0$  base case, since greedy always picks the absolute first interval by end time, the claim follows. Then assuming it holds up to  $j$ , we have  $b_{i_j} \leq b_{i_j^*} < a_{i_{j+1}^*}$ . The second inequality follows since the next interval in the optimal solution must start after the prior interval ending. But this means that interval  $i_{j+1}^*$  is *available* to the greedy algorithm after it has picked interval  $i_j$ , and since we would only not pick it if there is an available interval ending even earlier, we establish the claim for  $j + 1$  and conclude.  $\square$

Then from this claim we establish that  $k^* - 1 < k$  and so the Greedy Algorithm schedules  $k \geq k^*$  intervals. Since  $k^*$  is the optimal (maximum) number of intervals that can be scheduled, we conclude that  $k = k^*$  and the Greedy Algorithm schedules an optimal number of intervals.

For the runtime, we can order the intervals by increasing end time by sorting in time  $O(n \log n)$ . Next we observe that in Line 5 we only need to check that the start time  $a_i$  of the current interval

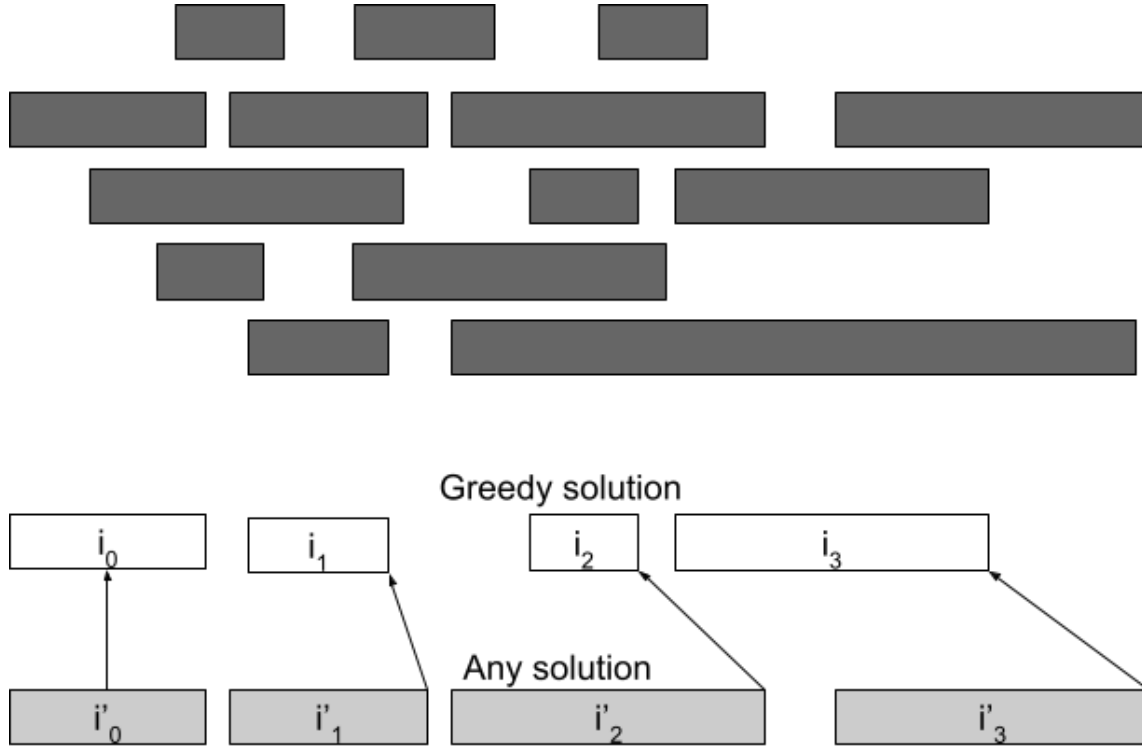


Figure 1: Transforming any interval scheduling solution into the greedy one.

is later than the end time of  $b_j$  of the most recently scheduled interval (since all others have earlier end time), so we can carry out this check in constant time. Thus the loop can be implemented in time  $O(n)$ , for a total runtime of  $O(n \log n) + O(n) = O(n \log n)$ .  $\square$