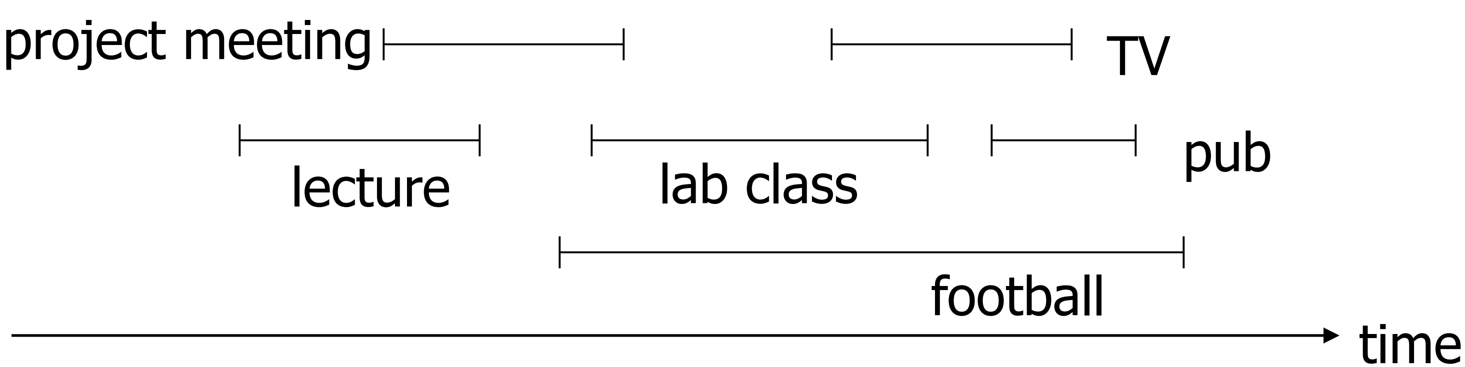
Chapter 6
Greedy Algorithms

References:
[KT 4.1]
[CLRS 16.1-16.2]

Activity Selection

- Given:
  - A set of activities, with starting and finishing times
- Goal:
  - To join as many activities as possible (without conflicting times)
- Example:
  - What is the maximum no. of activities one can join?

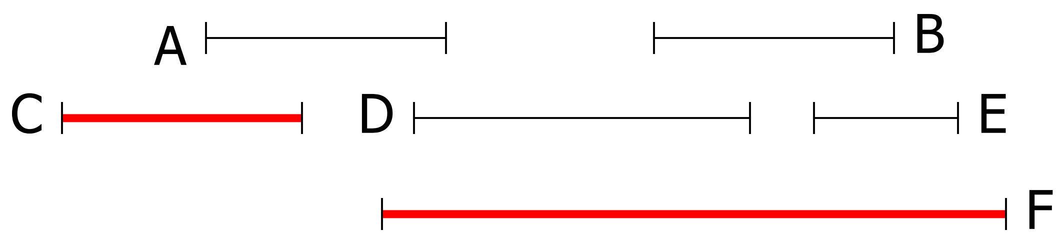


As Interval Selection

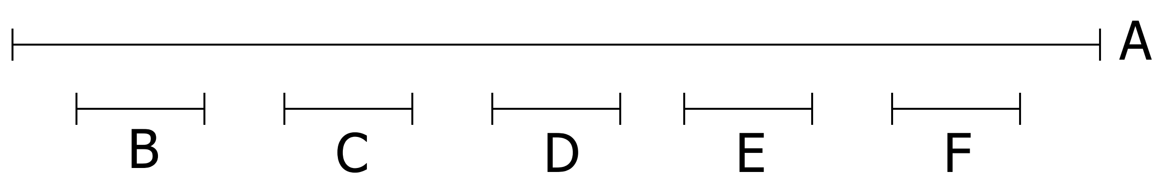
- Very often, an abstract formalization of the problem is useful
  - Problems arising from different contexts may turn out the same / similar
- Abstraction:
  - Given a set of intervals
  - Each interval i with starting time s(i) and finishing time f(i)
  - Intervals i and j in conflict if $s(i) < s(j) < f(i)$ or $s(j) < s(i) < f(j)$
- Algorithm?

Attempt #1: Earliest Arrival First

- From left to right, choose intervals not in conflict with chosen ones
  - On previous example:

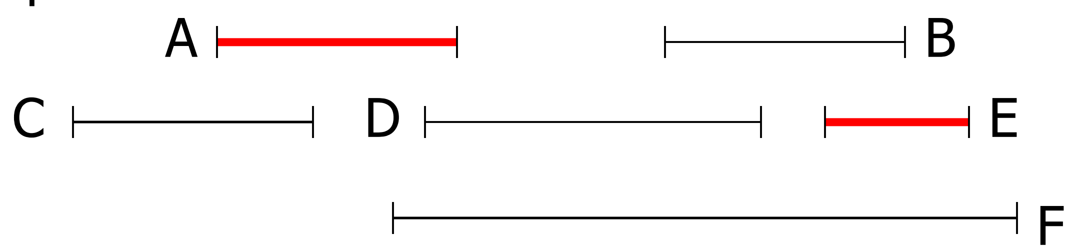


- Can be very bad:
  - Optimal solution: n-1 intervals
  - Our algorithm: 1

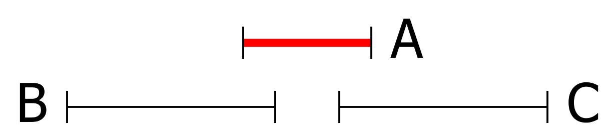


Attempt #2: Shortest Interval First

- Idea: short intervals less likely to create conflict
- Algorithm: repeat choosing the shortest interval that is not in conflict with already-chosen ones
  - Example:



- Not correct! Another example:

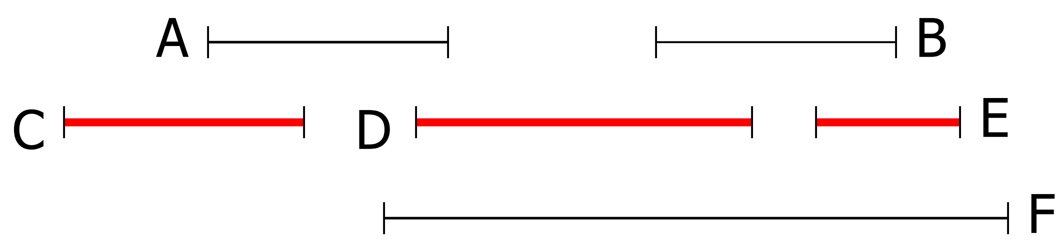


Principle of Greedy Algorithms

- The above are examples of *greedy algorithms*
- Principle:
  - Build solution incrementally
  - Each time, make the best choice as seen now (optimize some local criterion)
  - The choice results in a smaller subproblem, which we solve recursively
- Properties:
  - Easy to understand
  - Usually fast running time (e.g. O(n log n))
  - Rarely produce optimal solution (e.g., chess)
  - Sometimes gives approximately good solutions, but sometimes can be terribly bad

Attempt #3: Earliest Finishing First

- Greedy on: choosing the interval with earliest finishing time
  - (as long as it is not in conflict with already-chosen ones)
  - Idea: leave as much time as possible for other intervals
- Example:

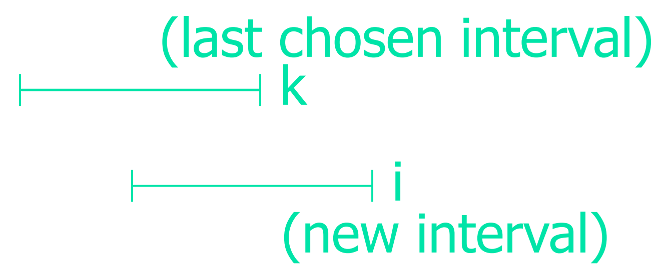


- It always gives the optimal solution

Attempt #3 Pseudocode

```
IntervalSelection(S[1..n]) {
  /* s() = start time, f() = finish time
     S = list of intervals, sorted in
     increasing order of f() */
  A := S[1]; k := 1
  for i := 2 to n {
    if ( s(S[i]) >= f(S[k]) ) {
      A := A union {S[i]}
      k := i
    }
  }
  return A  /* answer set */
}
```

(only need to check overlap with last chosen one – why?)



Running Time

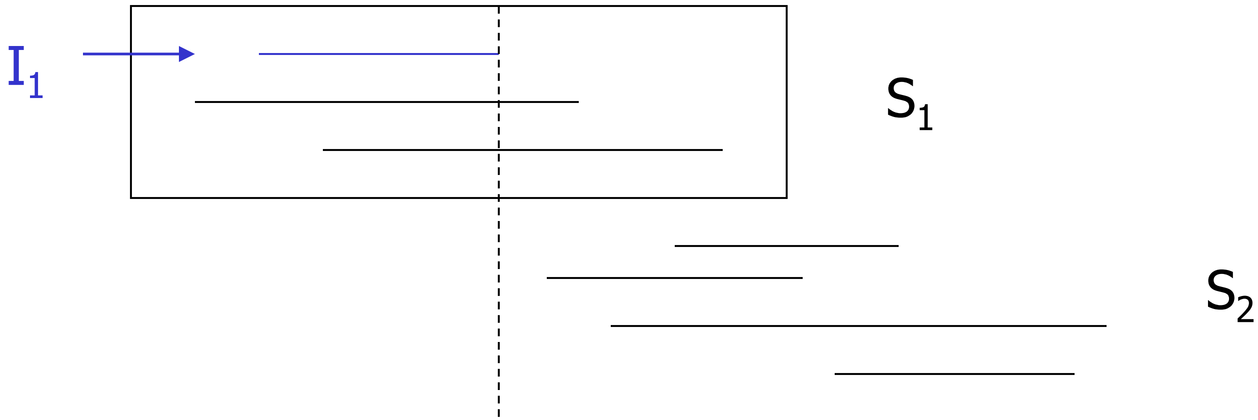
- Running time of Earliest-finishing-first:
  - Sort the intervals by finishing time: O(n log n)
  - Process each interval: at most n of them
    - Check overlapping of each interval: O(1) per interval
  - Total O(n log n) time

Need of Correctness Proofs

- Do you think the correctness of greedy algorithms is “obvious”?
- Consider the following problem:
  - You want to make an exact amount using the fewest number of coins. E.g. to make 74p, it can be done using 4 coins (50p+20p+2p+2p)
  - What is a greedy algorithm for this problem?
  - Does it always give the optimal solution?
  - (No, it doesn’t. See surgery)

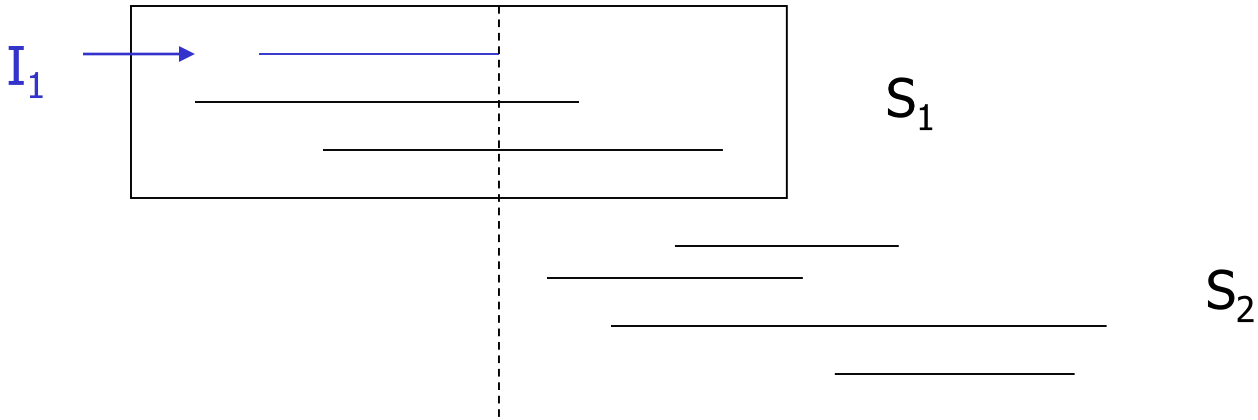
Correctness of Greedy Selection

- Proof by induction on number of intervals, n
- Base case n = 0: trivial
- Given n intervals, split them into two groups:
  - S_1: those that start before I_1 (earliest finishing interval) finish
  - S_2: those that start after I_1 finish



Proof of Correctness (cont'd)

- The optimal solution S* can include at most 1 interval in S_1
  - Choosing I_1 is "safe", because it is not in conflict with any interval in S_2
  - In other words, if S* chose some other interval in S_1, it can be replaced with I_1 with no harm
  - Similarly, if S* does not include anything from S_1, it can add I_1 with no harm
- Then left with a smaller set of intervals, S_2
  - By induction, greedy gives optimal solution for S_2!



The Knapsack Problem

- Given:
  - A set of objects, each of weight w_i and value v_i
  - A knapsack with maximum possible weight W
- Goal:
  - Find the set of objects with maximum value and total weight at most W
- Example:
  - 3 items: (2kg, \$60), (3kg, \$100), (5kg, \$120), W = 6 kg
  - Optimal solution: 1^{st} and 2^{nd} item, value = \$160

Fractional Knapsack

- Algorithms? Greedy?
  - Smallest weight first?
  - Largest value first?
  - Largest value/weight ratio first?
- None of these give optimal solution
  - Example: largest value/weight ratio first
    - (1.01kg, \$99), (1.01kg, \$99), (2kg, \$100), W = 2 kg
- However, if we assume we can take *fractional items* (e.g. liquid, gold sand, …), there is a greedy algorithm that always gives optimal solutions
  - Idea: greedy on the value/weight ratio

Fractional Knapsack: Greedy Works

```
FractionalKnapsack(S) {
  Sort the objects by descending order of
    value/weight ratio
  TotalWeight := 0
  for object i := 1,2, … {
    if (TotalWeight + w_i <= W) {
      add object i to knapsack;
      TotalWeight := TotalWeight + w_i
    }
    else {
      add a fraction of object i that makes
        TotalWeight = W
      return
    }
  }
}
```

Optimality of Greedy Knapsack

- Same example:
  - 3 items: (2kg, \$60), (3kg, \$100), (5kg, \$120), W = 6 kg
  - Optimal solution: take all of (3kg, \$100), all of (2kg, \$60), and 1kg from (5kg, \$120). Value = \$(100 + 60 + 120/5) = \$184

- Note that the quantity (as a fraction) of items chosen (in descending order of value/weight ratio) is always (1,1, …, 1, x, 0, 0, …, 0) where $0 \le x \le 1$
- Proof idea: can always exchange for higher value-per-profit items keeping same weight