GREEDY ALGORITHMS

David Kauchak
CS 140 – Spring 2023

1

Admin

Assignment 5

2

Checkpoint 1

2 pages of notes

Up through 2/15 (no material from this week)

Wednesday class: review session, QA session, work session

3

A problem

Input: a number k

Output: $\{n_p, n_n, n_d, n_q\}$, where $n_p+5n_n+10n_d+25n_q=k$ and $n_p+n_n+n_d+n_q$ is minimized

What is this problem?
How would you state it in English?

4

Making change!

Input: a number k

Output: $\{n_p, n_n, n_d, n_q\}$, where $n_p+5n_n+10n_d+25n_q=k$ and $n_p+n_n+n_d+n_q$ is minimized

Provide (U.S.) coins that sum up to *k* such that we minimize the number of coins

5

Making change!

Input: a number k

Output: $\{n_p, n_n, n_d, n_q\}$, where $n_p+5n_n+10n_d+25n_q=k$ and $n_p+n_n+n_d+n_q$ is minimized

Algorithm to solve it?

6

Making change!

Input: a number k

Output: $\{n_p, n_n, n_d, n_q\}$, where $n_p+5n_n+10n_d+25n_q=k$ and $n_p+n_n+n_d+n_q$ is minimized

$n_q = \lfloor k \,/\, 25 \rfloor$ pick as many quarters as we can

Solve:
$n_p + 5nn + 10nd = \mathrm{k} - 25\lfloor k \,/\, 25 \rfloor$ recurse

7

Algorithms vs heuristics

What is the difference between an algorithm and a heuristic?

Algorithm: a set of steps for arriving at the *correct* solution

Heuristic: a set of steps that will arrive at some solution

8

Making change!

$n_q = \lfloor k / 25 \rfloor$ pick as many quarters as we can

Solve:
$n_p + 5nn + 10nd = \text{k} - 25\lfloor k / 25 \rfloor$ recurse

Algorithm or heuristic?

Need to prove its correct!

9

Greedy algorithms

What is a greedy algorithm?

Algorithm that makes a local decision with the goal of creating a globally optimal solution

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to sub-problems

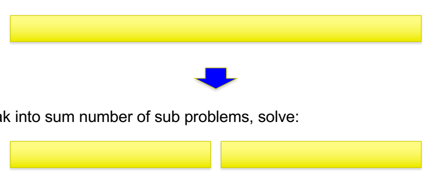
What does this mean? Where have we seen this before?

10

Greedy vs. divide and conquer

Divide and conquer

To solve the general problem:



Break into sum number of sub problems, solve:

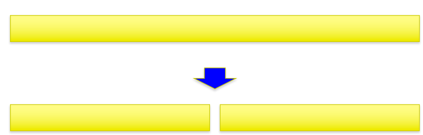
then possibly do a little work

11

Divide and conquer

Divide and conquer

To solve the general problem:



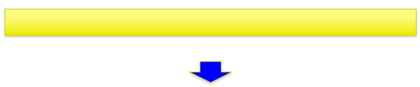
The solution to the general problem is solved with respect to solutions to sub-problems!

12

Greedy vs. divide and conquer

Greedy

To solve the general problem:



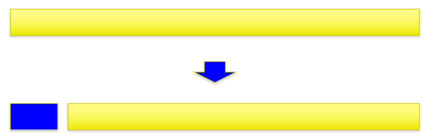
Pick a locally optimal solution and repeat

13

Greedy vs. divide and conquer

Greedy

To solve the general problem:



The solution to the general problem is solved with respect to solutions to sub-problems!

Slightly different than divide and conquer

14

Proving greedy algorithms correct

One approach, prove:

1) Optimal substructure: The optimal solution contains within it the optimal solution to subproblems

2) Greedy choice property: The greedy choice is contained within some optimal solution

15

Making change!

$n_q = \lfloor k / 25 \rfloor$ pick as many quarters as we can

Solve:

$n_p + 5nn + 10nd = \mathrm{k} - 25\lfloor k / 25 \rfloor$ recurse

$\{c_1, c_2, c_3, \dots, cm\}$ solution: individual coins selected

16

Optimal substructure

If $\{c_1, c_2, c_3, \ldots, cm\}$ is optimal for k, then $\{c_2, c_3, \ldots, cm\}$ is optimal for k-c_1

We can combine a greedy choice with the optimal solution for the remaining problem and get a solution to the general problem

17

Optimal substructure

Proof by contradiction:

Assume $\{c_1, c_2, c_3, \ldots, cm\}$ is optimal for k, *but* $\{c_2, c_3, \ldots, cm\}$ is not optimal for k-c_1

What does that imply?

18

Optimal substructure

Proof by contradiction:

Assume $\{c_1, c_2, c_3, \ldots, c_m\}$ is optimal for k, *but* $\{c_2, c_3, \ldots, c_m\}$ is not optimal for k-c_1

There is some other set of coins $\{c'2, c'3, \ldots, c'_n\}$ where n < m that add up to k-c_1

Any problem contradiction?

19

Optimal substructure

Proof by contradiction:

Assume $\{c_1, c_2, c_3, \ldots, c_m\}$ is optimal for k, *but* $\{c_2, c_3, \ldots, c_m\}$ is not optimal for k-c_1

There is some other set of coins $\{c'_2, c'_3, \ldots, c'_n\}$ where n < m that add up to k-c_1

$\{c_1, c'_2, c'_3, \ldots, c'_n\}$ would be a solution, but since n < m this implies that our original solution wasn't optimal!

20

Optimal substructure

If $\{c_1, c_2, c_3, \ldots, c_m\}$ is optimal for

$\{ c_2, c_3, \ldots, c_m\}$ is optimal for k-c_1

We can make greedy decisions

21

Greedy choice property

Greedy choice property: The greedy choice is contained within some optimal solution

The greedy choice results in an optimal solution

22

Greedy choice property

Proof by contradiction:

Let $\{c_1, c_2, c_3, \ldots, c_m\}$ be an optimal solution

Assume it is ordered from largest to smallest

Assume that it does not make the greedy choice at some coin c_i

Any problem contradiction?

23

Greedy choice property

Proof by contradiction:

Let $\{c_1, c_2, c_3, \ldots, c_m\}$ be an optimal solution
Assume it is ordered from largest to smallest

Assume that it does not make the greedy choice at some coin c_i

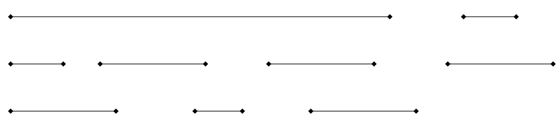
$g_i > c_i$. We need at least one more lower denomination coin because g_i can be made up of c_i and one or more of the other denominations

but that would mean that the solution is longer than the greedy!

24

Interval scheduling

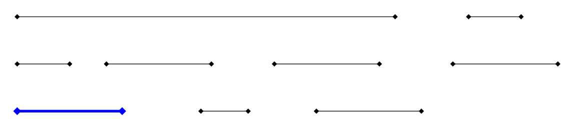
Given *n* activities A = [a_1, a_2, .., a_n] where each activity has start time s_i and a finish time f_i. Schedule as many as possible of these activities such that they don't conflict.



25

Interval scheduling

Given *n* activities A = [a_1, a_2, .., a_n] where each activity has start time s_i and a finish time f_i. Schedule as many as possible of these activities such that they don't conflict.

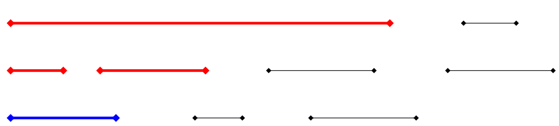


Which activities conflict?

26

Interval scheduling

Given *n* activities A = [a_1, a_2, .., a_n] where each activity has start time s_i and a finish time f_i. Schedule as many as possible of these activities such that they don't conflict.



Which activities conflict?

27

Simple recursive solution

Enumerate all possible solutions and find which schedules the most activities

```
IntervalSchedule-Recursive(A)
if A = {}
    return 0
else
    max = −∞
    for all a ∈ A
        A' ← A minus a and all conflicting activites with a
        s = IntervalSchedule-Recursive(A')
        if s > max
            max = s
    return 1 + max
```

28

Simple recursive solution

Is it correct?

- max{all possible solutions}

Running time?

- O(n!)

```
INTERVALSCHEDULE-RECURSIVE(A)
if A = {}
    return 0
else
    max = −∞
    for all a ∈ A
        A' ← A minus a and all conflicting activites with a
        s = INTERVALSCHEDULE-RECURSIVE(A')
        if s > max
            max = s
    return 1 + max
```

29

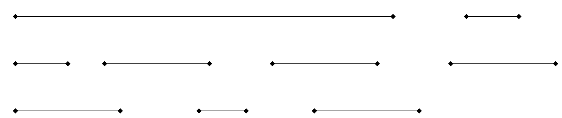
Can we do better?

Dynamic programming

- $O(n^2)$ (more on this soon!)

Greedy solution – Is there a way to repeatedly make local decisions?

- Key: we'd still like to end up with the *optimal* solution



30

Overview of a greedy approach

Greedily pick an activity to schedule

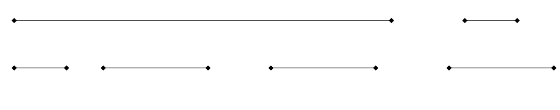
Add that activity to the answer

Remove that activity and all conflicting activities. Call this A'.

Repeat on A' until A' is empty

31

Greedy options



32

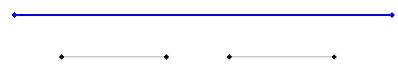
Greedy options

Select the activity that starts the earliest, i.e. argmin$\{s_1, s_2, s_3, \ldots, s_n\}$?

33

Greedy options

Select the activity that starts the earliest, i.e. argmin$\{s_1, s_2, s_3, \ldots, s_n\}$?



non-optimal

34

Greedy options

Select the shortest activity, i.e. argmin$\{f_1\text{-}s_1, f_2\text{-}s_2, f_3\text{-}s_3, \ldots, f_n\text{-}s_n\}$

35

Greedy options

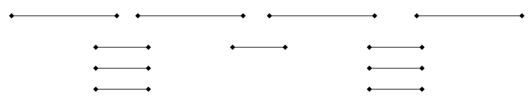
Select the shortest activity, i.e. argmin$\{f_1\text{-}s_1, f_2\text{-}s_2, f_3\text{-}s_3, \ldots, f_n\text{-}s_n\}$

non-optimal

36

Greedy options

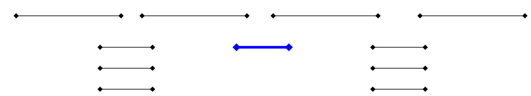
Select the activity with the smallest number of conflicts



37

Greedy options

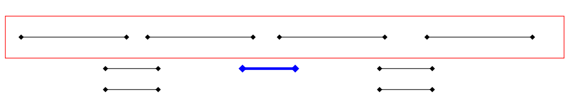
Select the activity with the smallest number of conflicts



38

Greedy options

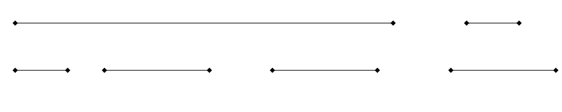
Select the activity with the smallest number of conflicts



39

Greedy options

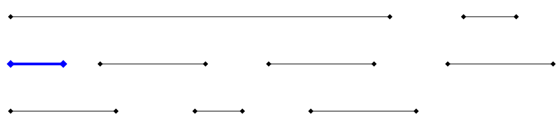
Select the activity that ends the earliest, i.e. argmin{f_1, f_2, f_3, …, f_n}?



40

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

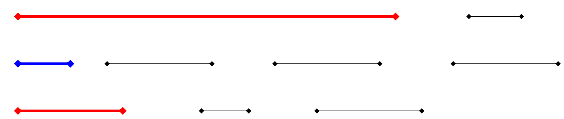


remove the conflicts

41

Greedy options

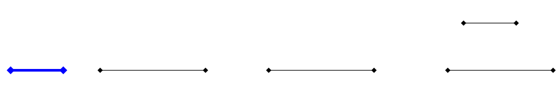
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



42

Greedy options

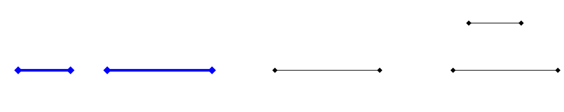
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



43

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

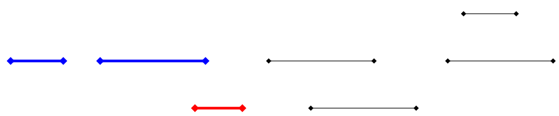


remove the conflicts

44

Greedy options

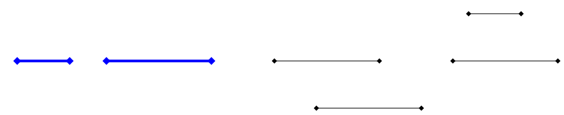
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



45

Greedy options

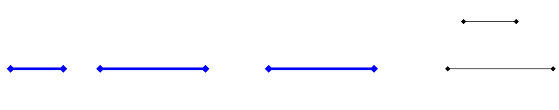
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



46

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



47

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

48

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

Multiple optimal solutions

49

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

50

Greedy options

Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?

51

Efficient greedy algorithm

Once you've identified a reasonable greedy heuristic:

- Prove that it always gives the correct answer
- Develop an efficient solution

52

Is our greedy approach correct?

"Stays ahead" argument:

show that no matter what other solution someone provides you, the solution provided by your algorithm always "stays ahead", in that no other choice could do better

53

Is our greedy approach correct?

"Stays ahead" argument

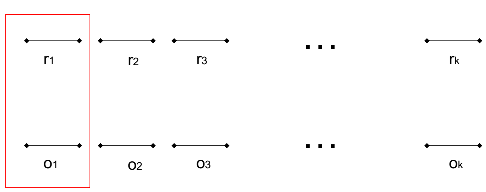
Let $r_1, r_2, r_3, \ldots, r_k$ be the solution found by our approach

Let $o_1, o_2, o_3, \ldots, o_k$ be another optimal solution

Show our approach "stays ahead" of any other solution

54

Stays ahead

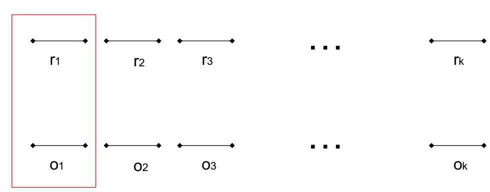


Compare first activities of each solution

what do we know?

55

Stays ahead

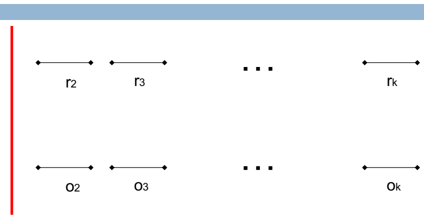


$finish(r_1) \leq finish(o_1)$

what does this imply?

56

Stays ahead



We have **at least** as much time as any other solution to schedule the remaining 2…k tasks

57

An efficient solution

INTERVALSCHEDULE-GREEDY(A)

```
sort A based on finish times f_i
for i ← 1 to n
        add a_i to R
        finish ← f_i
        while s_i < finish
                i ← i + 1
return R
```

58

Running time?

INTERVALSCHEDULE-GREEDY(A)

```
sort A based on finish times f_i        Θ(n log n)
for i ← 1 to n                          ┐
        add a_i to R                    │
        finish ← f_i                    ├ Θ(n)
        while s_i < finish              │
                i ← i + 1               ┘
return R
```

Overall: Θ(n log n)

Better than:

O(n!)

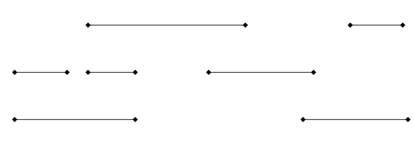
$O(n^2)$

59

Scheduling *all* intervals

Given n activities, we need to schedule **all** activities.

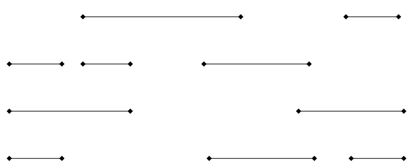
Goal: minimize the number of resources required.



60

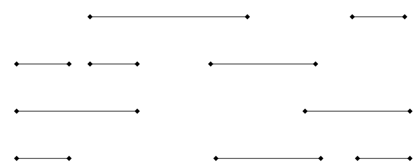
Greedy approach?

The best we could ever do is the maximum number of conflicts for any time period



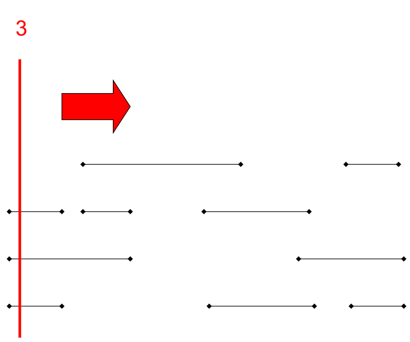
61

Calculating max conflicts efficiently



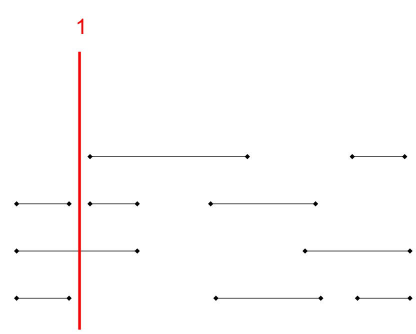
62

Calculating max conflicts efficiently



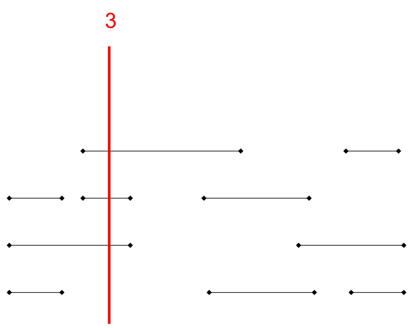
63

Calculating max conflicts efficiently



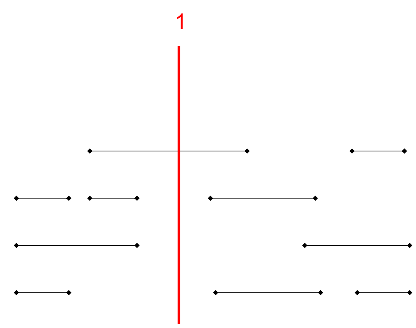
64

Calculating max conflicts efficiently



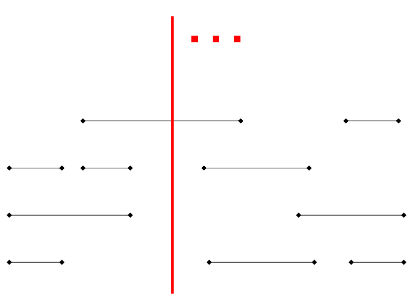
65

Calculating max conflicts efficiently



66

Calculating max conflicts efficiently



67

Calculating max conflicts

```
AllIntervalScheduleCount(A)
Sort the start and end times, call this X
current ← 0
max ← 0
for i ← 1 to length[X]
        if x_i is a start node
                current++
        else
                current--
        if current > max
                max ← current
return max
```

68

Correctness?

We can do no better then the max number of conflicts.
This exactly counts the max number of conflicts.

```
AllIntervalScheduleCount(A)
Sort the start and end times, call this X
current ← 0
max ← 0
for i ← 1 to length[X]
        if x_i is a start node
                current++
        else
                current--
        if current > max
                max ← current
return max
```

69

Runtime?

O(2n log 2n + n) = O(n log n)

```
AllIntervalScheduleCount(A)
Sort the start and end times, call this X
current ← 0
max ← 0
for i ← 1 to length[X]
        if x_i is a start node
                current++
        else
                current--
        if current > max
                max ← current
return max
```

70

Knapsack problems: Greedy or not?

0-1 Knapsack – A thief robbing a store finds n items worth v_1, v_2, .., v_n dollars and weight w_1, w_2, …, w_n pounds, where v_i and w_i are integers. The thief can carry at most W pounds in the knapsack. Which items should the thief take if he wants to maximize value.

Fractional knapsack problem – Same as above, but the thief happens to be at the bulk section of the store and can carry fractional portions of the items. For example, the thief could take 20% of item i for a weight of $0.2w_i$ and a value of $0.2v_i$.

71