GREEDY ALGORITHMS

David Kauchak
CS 140 – Spring 2024

1

Admin

Assignment 6

ChatGPT

Collaboration

2

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?

3

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

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

Algorithm to solve it?

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

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

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

6

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

7

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!

8

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?

9

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

10

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!

11

Greedy vs. divide and conquer

Greedy

To solve the general problem:

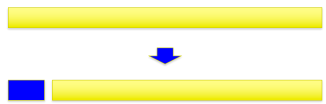
Pick a locally optimal solution and repeat

12

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

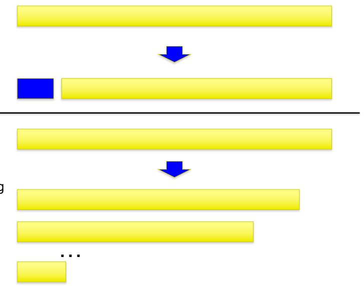
13

Greedy vs. DP

greedy

Only recurse on one subproblem

dynamic programming



Need to solve (recurse on) subproblems to figure out optimal answer

14

Proving correctness: greedy choice property

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

The greedy choice results in an optimal solution

15

Making change!

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

Solve:

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

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

16

Proving greedy choice property

Option 1: proof by contradiction

- Assume you have an optimal solution to the problem
  - Sometimes you have to think about it ordered/arranged a particular way
- Assume that somewhere along the way the solution contains a decision that is different than your greedy algorithm
- Argue this results in a contradiction, i.e., that the solution you're considering is not optimal

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

$$c_1, c_2, c_3, \ldots, c_i, \ldots, c_m$$
$$g_1\ g_2, g_3, \ldots, g_i, \ldots, g_n$$

Any problem contradiction?

23

Greedy choice property

Proof by contradiction:

$$c_1, c_2, c_3, \ldots, c_i, \ldots, c_m$$
$$g_1\ g_2, g_3, \ldots, g_i, \ldots, g_n$$

$g_i > c_i$. 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

Greedy choice property

$g_i > c_i$

$g_i = 5$
$c_i = 1$

- there are at least 4 other pennies
- could always replace 5 pennies with a nickel to create shorter solution!

25

Greedy choice property

$g_i > c_i$

$g_i = 10$
$c_i = 5$

- there are at least 2 nickels (assuming we've dealt with pennies first)
- could always replace those coins with a dime to create a shorter solution

26

Greedy choice property

$g_i > c_i$

$g_i = 25$
r = remaining sum
coins(r – 25): number of coins to get remaining sum - 25

c_i = 10: 10 + 10 + 5 + coins(r-25)
c_i = 5: 5 + 5 + 5 + 5 +5 + coins(r-25)

The greedy solution will always be better

27

Greedy choice property fails

Coins: 9, 4, 1

What's the best way to make 12?

28

Greedy choice property fails

Coins: 9, 4, 1

$g_i > c_i$

$g_i = 9$
r = remaining sum
coins(r – 9): number of coins to get remaining sum - 9

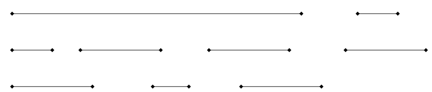
c_i = 4: 4 + coins(r-4)

There is no way to guarantee that we would have to use the same set of coins are coins(r-9)

29

Interval scheduling

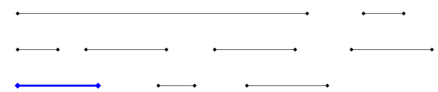
Given n activities A = $[a_1, a_2, .., a_n]$ where each activity has start time s_i and a finish time f_i. Schedule as many as possible of these activities such that they don't conflict.



30

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.

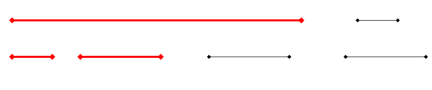


Which activities conflict?

31

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?

32

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
```

33

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
```

34

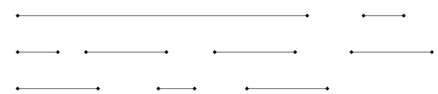
Can we do better?

Dynamic programming

- $O(n^2)$

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

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



35

Overview of a greedy approach

Greedily pick an activity to schedule

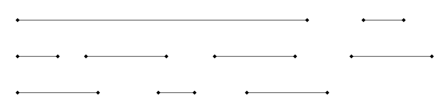
Add that activity to the answer

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

Repeat on A' until A' is empty

36

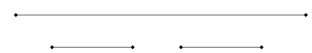
Greedy options



37

Greedy options

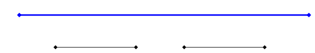
Select the activity that starts the earliest, i.e. argmin$\{s_1, s_2, s_3, \ldots, s_n\}$?



38

Greedy options

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

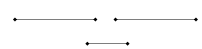


non-optimal

39

Greedy options

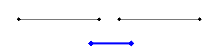
Select the shortest activity, i.e. argmin$\{f_1-s_1, f_2-s_2, f_3-s_3, \ldots, f_n-s_n\}$



40

Greedy options

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

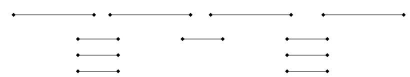


non-optimal

41

Greedy options

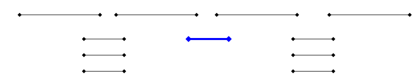
Select the activity with the smallest number of conflicts



42

Greedy options

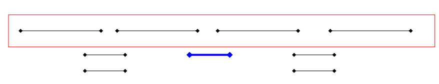
Select the activity with the smallest number of conflicts



43

Greedy options

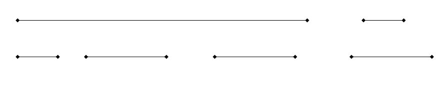
Select the activity with the smallest number of conflicts



44

Greedy options

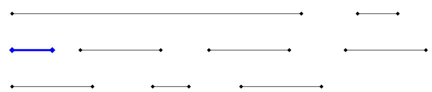
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



45

Greedy options

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

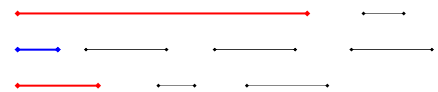


remove the conflicts

46

Greedy options

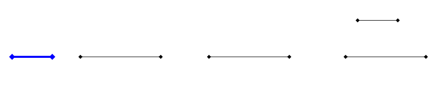
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



47

Greedy options

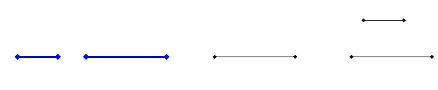
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



48

Greedy options

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

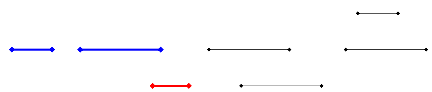


remove the conflicts

49

Greedy options

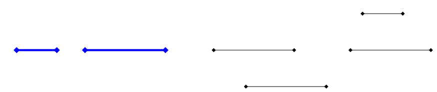
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



50

Greedy options

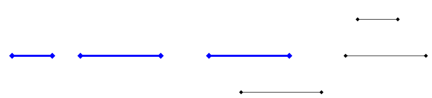
Select the activity that ends the earliest, i.e. argmin$\{f_1, f_2, f_3, \ldots, f_n\}$?



51

Greedy options

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



52

Greedy options

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

53

Greedy options

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

Multiple optimal solutions

54

Greedy options

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

55

Greedy options

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

56

Efficient greedy algorithm

Once you've identified a reasonable greedy heuristic:
- Prove that it always gives the correct answer
- Develop an efficient solution

57

Is our greedy approach correct?

Option 1: proof by contradiction

Option 2: "Stays ahead" argument:

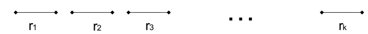
show that no matter what other solution someone provides you, the solution provided by your algorithm always "stays ahead", in that no other choice could do better

58

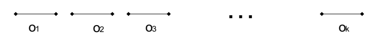
Is our greedy approach correct?

"Stays ahead" argument

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



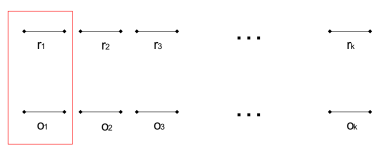
Let $o_1, o_2, o_3, \ldots, o_k$ be another optimal solution



Show our approach "stays ahead" of any other solution

59

Stays ahead

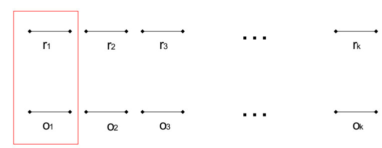


Compare first activities of each solution

what do we know?

60

Stays ahead

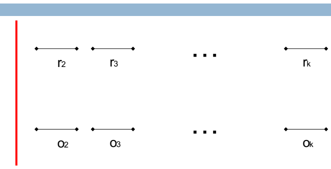


$\text{finish}(r_1) \leq \text{finish}(o_1)$

what does this imply?

61

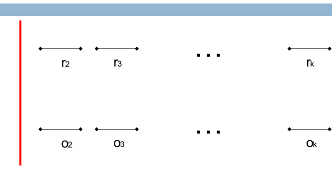
Stays ahead



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

62

Stays ahead



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

What kind of proof is this?

63

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
```

64

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:

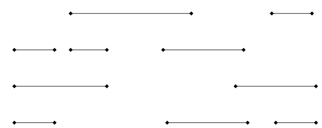
$O(n!)$

$O(n^2)$

65

Scheduling *all* intervals

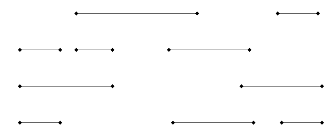
Given *n* activities, we need to schedule **all** activities.
Goal: minimize the number of resources required.



66

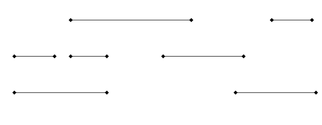
Greedy approach?

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



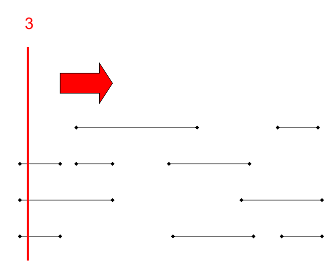
67

Calculating max conflicts efficiently



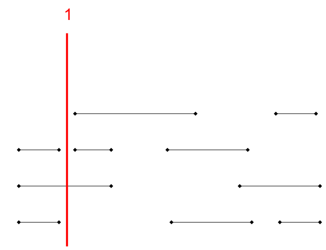
68

Calculating max conflicts efficiently



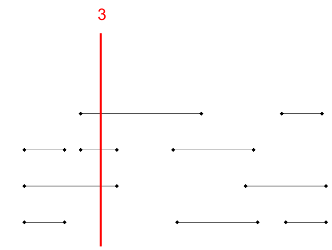
69

Calculating max conflicts efficiently



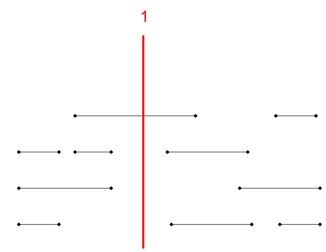
70

Calculating max conflicts efficiently



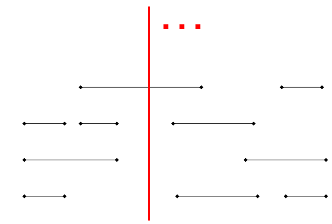
71

Calculating max conflicts efficiently



72

Calculating max conflicts efficiently



73

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
```

74

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
```

75

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
```

76

Knapsack problems: Greedy or not?

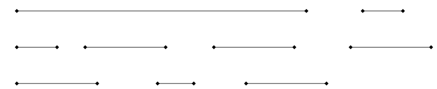
0-1 Knapsack – A thief robbing a store finds n items worth v_1, v_2, .., v_n dollars and weight w_1, w_2, …, w_n pounds, where v_i and w_i are integers. The thief can carry at most W pounds in the knapsack. Which items should the thief take if he wants to maximize value.

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 $0.2v_i$.

77

Handout

78



Here are some options for greedy algorithms. Do they work?
Can you come up with counterexamples?

- Starts earliest
- Least number of conflicts
- Shortest

79

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 $w_1, w_2, \ldots, w_n$ 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 $0.2v_i$.

80