Checkpoint 1

2 pages of notes

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

Wednesday class: review session, QA session, work session

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?
**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

Algorithm to solve it?

---

**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
Making change!

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

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

Algorithm or heuristic?
Need to prove its correct!

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?

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

Divide and conquer
To solve the general problem:

The solution to the general problem is solved with respect to solutions to sub-problems!
Greedy vs. divide and conquer

To solve the general problem:
- Pick a locally optimal solution and repeat

Slightly different than divide and conquer

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

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

\[ \{ c_1, c_2, c_3, \ldots, c_m \} \]
- solution: individual coins selected
Optimal substructure

If \( \{c_1, c_2, c_3, \ldots, c_m\} \) is optimal for \( k \), then
\( \{c_2, c_3, \ldots, c_m\} \) 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.

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 \( k-c_1 \)

What does that imply?

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?

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!
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

Greedy choice property

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

The greedy choice results in an optimal solution

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?

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!
Interval scheduling

Given \( n \) activities \( A = [a_1, a_2, \ldots, a_n] \) 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.

---

Simple recursive solution

Enumerate all possible solutions and find which schedules the most activities.

```
intervalScheduleRecursive(A)
1. If A = {} return 0
2. else set max = -\( \infty \)
3. for all a in A
4. set A’ = A minus a and all conflicting activities with a
5. set s = intervalScheduleRecursive(A’)
6. if s > max
7. set max = s
8. return 1 + max
```
Simple recursive solution

Is it correct?
- \( \max \{ \text{all possible solutions} \} \)

Running time?
- \( O(n!) \)

```plaintext
FUNCTION SCHEDULE-RECURSIVE(A)
  1: if \( A = \emptyset \) then
  2: return 0
  3: else
  4: \( \mathit{max} = -\infty \)
  5: for all \( a \in A \) do
  6: \( A' = A \text{ minus } a \text{ and all conflicting activities with } a \)
  7: \( \mathit{s} = \text{SCHEDULE-RECURSIVE}(A') \)
  8: if \( \mathit{s} > \mathit{max} \) then
  9: \( \mathit{max} = \mathit{s} \)
 10: return 1 + \mathit{max}
```

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

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

Greedy options
Greedy options

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

33

Greedy options

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

non-optimal

34

Greedy options

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

35

Greedy options

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

non-optimal

36
Greedy options
Select the activity with the smallest number of conflicts

Greedy options
Select the activity with the smallest number of conflicts

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

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

remove the conflicts

Greedy options

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

Greedy options

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

Greedy options

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

remove the conflicts
Select the activity that ends the earliest, i.e. $\arg\min\{f_1, f_2, f_3, \ldots, f_n\}$?
Greedy options

Select the activity that ends the earliest, i.e. \( \arg\min \{f_1, f_2, f_3, \ldots, f_n\} \)?

Multiple optimal solutions

Efficient greedy algorithm

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

Is our greedy approach correct?

“Stays ahead” argument

Let \( r_1, r_2, \ldots, r_k \) be the solution found by our approach

\[
\begin{array}{c}
\text{\( r_1 \)} \\
\text{\( r_2 \)} \\
\text{\( r_3 \)} \\
\vdots \\
\text{\( r_k \)}
\end{array}
\]

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

\[
\begin{array}{c}
\text{\( o_1 \)} \\
\text{\( o_2 \)} \\
\text{\( o_3 \)} \\
\vdots \\
\text{\( o_k \)}
\end{array}
\]

Show our approach “stays ahead” of any other solution

Stays ahead

Compare first activities of each solution

what do we know?

Stays ahead

\[
\begin{array}{c}
\text{\( r_1 \)} \\
\text{\( r_2 \)} \\
\text{\( r_3 \)} \\
\vdots \\
\text{\( r_k \)}
\end{array}
\]

\[
\begin{array}{c}
\text{\( o_1 \)} \\
\text{\( o_2 \)} \\
\text{\( o_3 \)} \\
\vdots \\
\text{\( o_k \)}
\end{array}
\]

\[\text{finish}(r_1) \leq \text{finish}(o_1)\]

what does this imply?
Stays ahead

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

An efficient solution

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

Running time?

Scheduling all intervals

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

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

Calculating max conflicts efficiently

Calculating max conflicts efficiently

Calculating max conflicts efficiently
Calculating max conflicts efficiently

```
ALL_INTERVAL_SCHEDULE_COUNT(A)
1 Sort the start and end times, call this X
2 current ← 0
3 max ← 0
4 for i ← 1 to length(X)
5 if x_i is a start node
6 current ++
7 else current --
8 if current > max
9 max ← current
10 return max
```
Correctness?

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

```java
ALL_INTERVAL_SCHEDULE_COUNT(A)
1 Sort the start and end times, call this X
2 current ← 0
3 max ← 0
4 for i ← 1 to length[X]
5 if xi is a start node
6 current ++
7 else
8 current ← −
9 if current > max
10 max ← current
11 return max
```

Runtime?

\[ O(2n \log 2n + n) = O(n \log n) \]

```java
ALL_INTERVAL_SCHEDULE_COUNT(A)
1 Sort the start and end times, call this X
2 current ← 0
3 max ← 0
4 for i ← 1 to length[X]
5 if xi is a start node
6 current ++
7 else
8 current ← −
9 if current > max
10 max ← current
11 return max
```

Knapsack problems:

Greedy or not?

0-1 Knapsack – A thief robbing a store finds n items worth \( v_1, v_2, \ldots, v_n \) 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 0.2\( w_i \) and a value of 0.2\( v_i \).