Announcements

Announcements

- Homework 4 due tonight
- Homework 5 out tonight

Dynamic Programming

Introduction to Dynamic Programming

- Dynamic programming is a method of solving a problem in which the solution to a large problem is based on the solutions to smaller subproblems

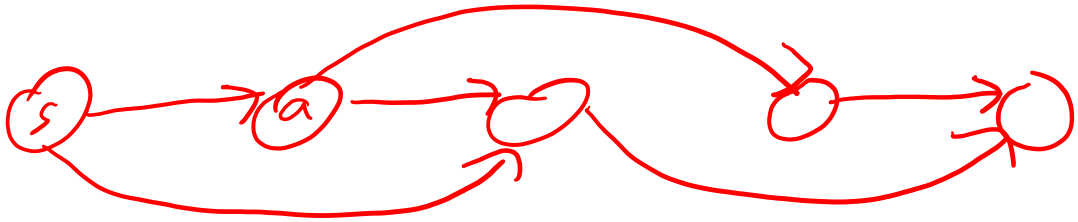
DP ≠ D+C → start with large problem, work down to small s-p

↙ storing solutions to sub-problems because of redundancy

↘ bottom-up

Example: Shortest Paths in a DAG

- Suppose we want to find the shortest path from S to D. How can we do this in one scan of the linearized DAG?



expd

```
initialize all dist(·) values to ∞
dist(s) = 0
for each v ∈ V\{s}, in linearized order:
    dist(v) = min_{(u,v)∈E} {dist(u) + l(u, v)}
```

after s

O(E)

update equation

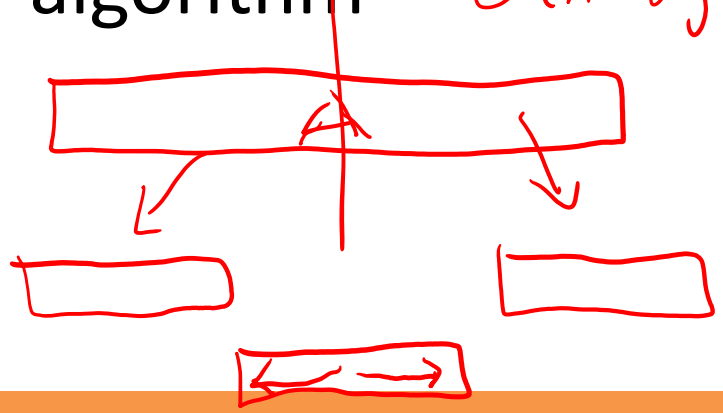
In-Class Exercise: Maximum Contiguous Sequence

Max Subarray Sum

could be negative

- You have a sequence of numbers $x_1, x_2, x_3, \ldots, x_n$
- Find the contiguous subsequence $[x_i, \ldots, x_j]$ with the greatest sum
- Not allowed to skip elements!
- Use dynamic programming to find an O(n) algorithm

O(n log n) with D&C

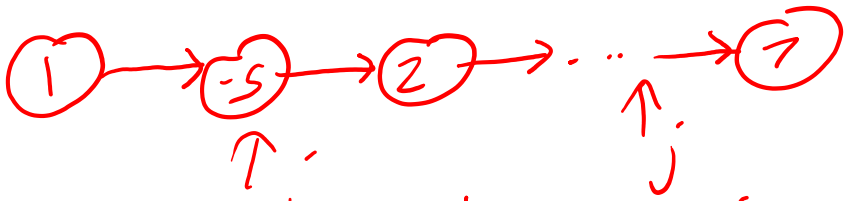


In-Class Exercise: Maximum Contiguous Sequence

$O(n)$

$x_1 = 1$, $x_2 = -5$, $x_3 = 2$, ..., $x_n = 7$

DAG-based



1. Define the DAG.
   a. Define the nodes: One node per element, s.p.: what is the max sum I can get from a CSS ending at this element?
   b. Define the edges: Draw an edge from $i-1 \rightarrow i$ for all $i > 1$.
   c. Add weights to nodes equal to the value of the element
2. Find the longest node-weighted path with no spec. start

In-Class Exercise: Maximum Contiguous Sequence

$x_1 \; x_2 \; x_3 \; \ldots \; x_n$

Update equation solution

1. S.p.: What is the MSS I can get from a CSS ending at a particular index?

2. Define a variable, $M(i)$ = answer to ↑ for index i

3. Write pseudocode

$M(1) = x_1$

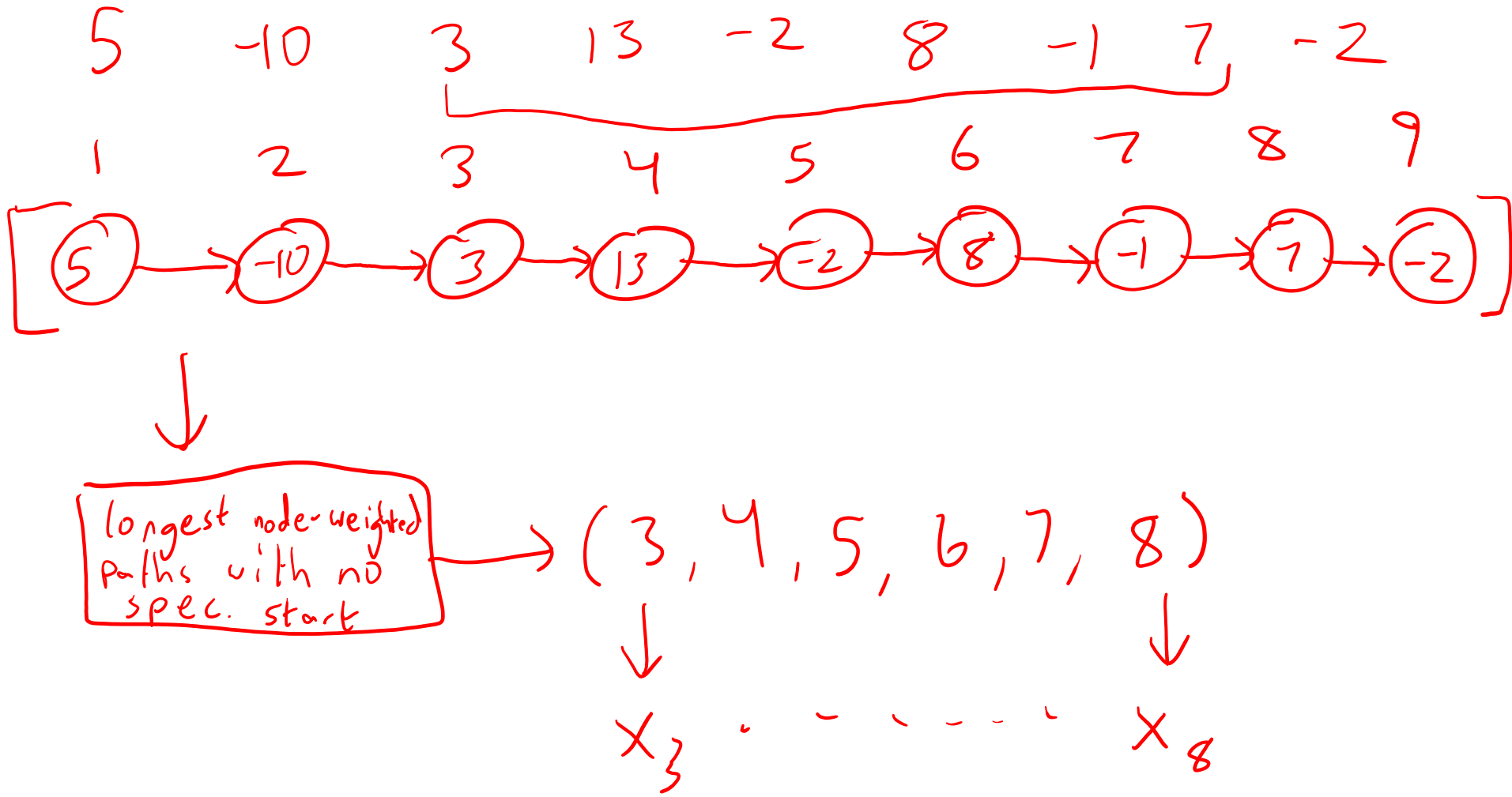
For $i = 2 : n$

$M(i) = (\max\{M(i-1) + x_i, x_i\})$

return $\max\{M(i)\}$

$O(n)$

In-Class Exercise: Maximum Contiguous Sequence



In-Class Exercise: Splitting a String

- You are given a string of characters without any spaces or punctuation. Itlookslikethissentence. You want to figure out whether it is possible to insert spaces to split the string into valid words.
- You are given a dictionary to check whether a sequence of characters is a valid word.
- Create a dynamic programming algorithm to determine whether the string can be split into valid words.

Ithi ? S(i) = ?

Ith S(i-1) = True

In-Class Exercise: Splitting a String

S.p.: Is it possible to split the ~~sequence~~ string $A_1 \cdots A_n$ into a sequence of valid words?

DAG solution $O(n^2)$ → all prefixes

1. Nodes? (One node per character, in order plus empty substring node at beginning)
2. Edges? Draw an edge from index $i \rightarrow$ index j if $i<j$ AND $S[i+1 \rightarrow j]$ is a valid word
3. Is there a path from $0 \rightarrow n$? I think

In-Class Exercise: Splitting a String

Update equation

SP[i] = is it possible to split S[1...i] into a sequence of valid words?

W[i] = answer to ↑

W[0] = True

for i = 1...n:

W[i] = { True if there exists j<i: W[j] = True and S[j+1...i] is a word
       { False otherwise

for j = 1...i-1 check W[j] and S[j+1...i]

return W[n]

In-Class Exercise: Knapsack with Repetition

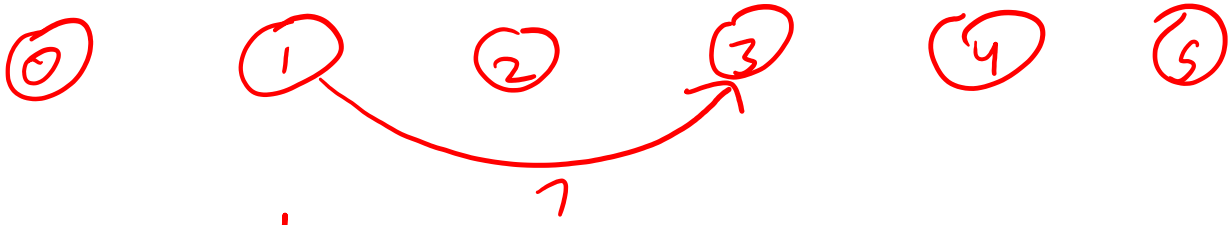
- You have a knapsack that can contain W pounds
- There is a collection of n items, where item i has weight w_i and value v_i.
- Assume all values are integers, and W is an integer.
- Goal is to maximize the value of items put into the knapsack
- Assume you can take as many copies of some item as you wish
- Draw the DAG for this problem (hint: nodes in the DAG correspond to different values of W).
- Formulate a solution using dynamic programming.

In-Class Exercise: Knapsack with Repetition

DAG - based solution

S.p.: Subproblem i is the question "What is the greatest value I can get using a total capacity of exactly i?"

Nodes: ↑



Edges: Draw an edge from $i \to j$ if $i < j$ and there is an item with weight $= j - i$. Weight each edge with value of the corr. item

In-Class Exercise: Knapsack with Repetition

Run longest edge-weighted path algorithm starting at 0-node, ending anywhere

Return the max value of these paths