Algorithms

Backtracking (part 2)

Recap:

- HW1 - due tonight
- HW0 - graded! please check
- Reading - due Sunday
- HW2 - post today

Warning: ♥ sections - usually harder!
If I really need you to understand, I'll spend class time.

- Check in: all working?
  (please email)

Backtracking: the pattern

Need to make a sequence of decisions:

- Turns in a game
  - ↳ need to choose a space
- Placing a queen → ≤ n decisions
- Is next element in the set? → 2 decisions

So: recursion! (reinforces recursion)

Need a decision

↳ recurse on all possible answers

Requires: some "state" info, (large) so we can build up the solution (or game).

Downside: SLOW

Example: Subset Sum

Given a set X of positive integers and a target value t, is there a subset of X which sums to t?

Ex: $X = \{8, 6, 7, 3, 10, 5, 9\}$

$t = 15$

Yes! $8+7$

$7+3+5$

(many in this example)

more?

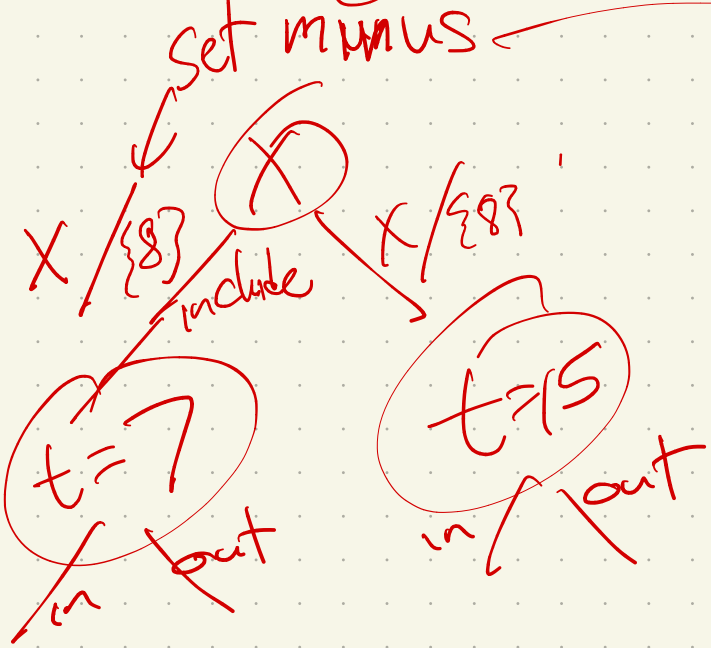
How would we solve?

recursively

Consider one at a time:

$$\left(X = \{\cancel{8}, 6, 7, 5, 3, 1, 9\}, 15 \right)$$

Consider 8: either in, or out



Formalize this: recursion!

try: $(X / X[1], t - X[1])$

$(X / X[1], t)$

or base case?

If $t = 0$, true

$t < 0$, fail

$X = \{0\}$, $t > 0$, fail

Algorithm:

reset to use ~~arrays~~ fixing input

```
⟨⟨Does any subset of X sum to T?⟩⟩
SubsetSum(X, T):
  if T = 0
      return True
  else if T < 0 or X = ∅
      return False
  else                                         rec. case
      x ← any element of X
      with ← SubsetSum(X \ {x}, T − x)      ⟨⟨Recurse!⟩⟩
      wout ← SubsetSum(X \ {x}, T)          ⟨⟨Recurse!⟩⟩
      return (with ∨ wout)
```

```
⟨⟨Does any subset of X[1..i] sum to T?⟩⟩
SubsetSum(X, i, T):
  if T = 0
      return True
  else if T < 0 or i = 0                       same
      return False                             run off end of array
  else
      with ← SubsetSum(X, i − 1, T − X[i])   ⟨⟨Recurse!⟩⟩
      wout ← SubsetSum(X, i − 1, T)          ⟨⟨Recurse!⟩⟩
      return (with ∨ wout)
```

if works I'm OK

remove X[i], so set now X[1..i-1]

Correctness: inductive proof, on $\underbrace{\text{size of } X}$, $\underbrace{i}$ makes X smaller

X[1..n] X[1...n-1] X[1..n-2]

Base cases:

$i = |X| = 0$ (so $X = \{\}$):

if $t=0$, true (alg does this)

if $t>0$ or $t<0$, alg gives false

Ind Hyp: algorithm works for $X[1..n-1]$ or smaller arrays.

Ind step: Full array $X[1..n]$

Consider $X[n]$:

are only 2 possibilities:

$X[n]$ in set,

or not in

Try both → by IH, ind. fairy gives me correct answer for each one

Then if one works, I also returns true.

Text Segmentation

Fix a "language", so can recognize "words".

Ex: - English text: dictionary

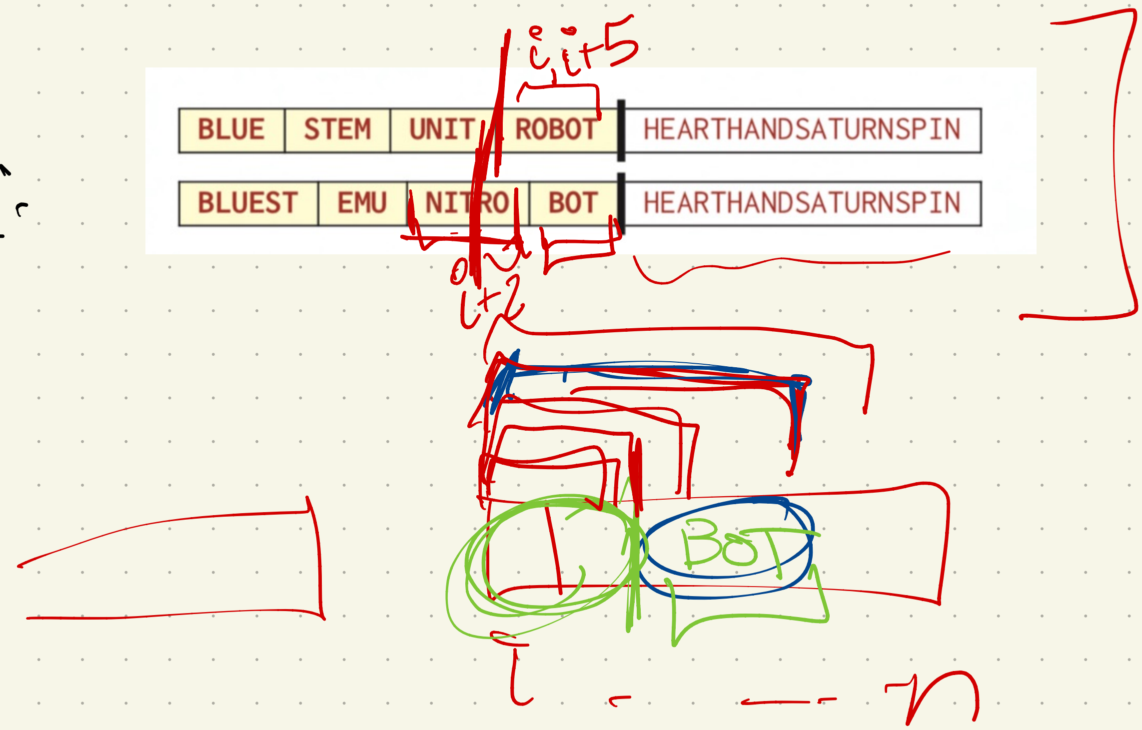
- palindromes: code a fcn that gives true for pal.

- genetic data: code fcn that is true for particular genetic sequences

⇒ Subroutine IsWord(s) will be given

Q: What happens to a smaller word that overlaps or is later?

Ex:



make $n-i+1$ recursive attempts

Code:

```
SPLITTABLE(A[1..n]):
  if n = 0
      return TRUE
  for i ← 1 to n
      if ISWORD(A[1..i])
          if SPLITTABLE(A[i+1..n])
              return TRUE
  return FALSE
```

Not A subarray.

isword | rec fairy | say it works

I work

Runtime:

$$S(n) \le \cancel{(n-1)S(n-1)} = \sum_{i=1}^{n-1} S(n-i)$$

exp

Issue w/ passing arrays:

If I pass A back (by ref, by global), then I need to indicate size is shrinking

His solution: (language independent!)

use an index

Passing by index / ptr / global / etc.

Given an index i, find a segmentation of the suffix $A[i\,..\,n]$.

easy :)

Formalize an (ugly?) recursion:

$$Splittable(i) = \begin{cases} \text{TRUE} & \text{if } i > n \\ \displaystyle\bigvee_{j=i}^{n} \big(\text{IsWord}(i,j) \wedge Splittable(j+1)\big) & \text{otherwise} \end{cases}$$

painful

& then code it:

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
      return TRUE
  for j ← i to n
      if IsWord(i, j)
          if SPLITTABLE(j + 1)
              return TRUE
  return FALSE
```

??

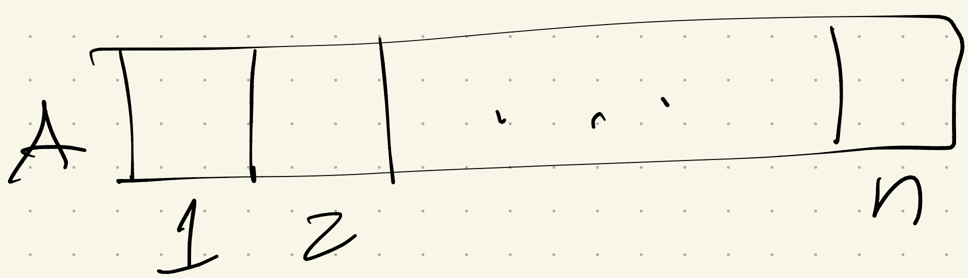
<u>Note</u>: this is harder than it looks!!

Longest Increasing Subsequence

Why "jump to the middle"?

Need a recursion!

First: how many subsequences?



Backtracking approach:

At index i:

Result:

Given two indices i and j, where $i < j$, find the longest increasing subsequence of $A[j\,..\,n]$ in which every element is larger than $A[i]$.

Recursion:

$$LISbigger(i,j) = \begin{cases} 0 & \text{if } j > n \\ LISbigger(i, j+1) & \text{if } A[i] \geq A[j] \\ \max \left\{ \begin{array}{c} LISbigger(i, j+1) \\ 1 + LISbigger(j, j+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

Code version:

```
LISBIGGER(i, j):
    if j > n
        return 0
    else if A[i] ≥ A[j]
        return LISBIGGER(i, j + 1)
    else
        skip ← LISBIGGER(i, j + 1)
        take ← LISBIGGER(j, j + 1) + 1
        return max{skip, take}
```

Problem – what did we want??

So:

```
LIS(A[1..n]):
    A[0] ← −∞
    return LISBIGGER(0, 1)
```