1 101100001000 101100001000 101100001000 101100001000 101100001000 1

CSCI 2824: Discrete Structures

Lecture 30: Recurrences and Dynamic Programming

Rachel Cox

Department of Computer Science

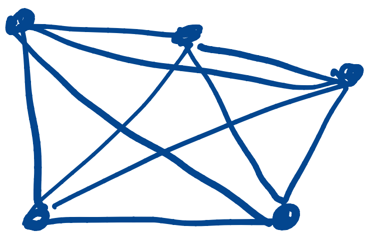
Homework 11 - Due Friday at Noon.

0 101100001000 101100001000 101100001000 101100001000 101100001000 0

Recursion

Previously – We solved counting problems using combinatorics as well as recurrences.

Example: How many games must be played in a round robin tournament with n teams? (in a round-robin tournament, each team plays each other team once)



Recurrences:

$$R(n) = R(n-1) + (n-1)$$

combinatorics:

$$C(n,2) = \frac{n!}{(n-2)!\,2!} = \frac{n(n-1)(n-2)!}{(n-2)!\,2!} \boxed{= \frac{n(n-1)}{2}}$$

Recursion

Example: What is the number of n −permutations of n objects? (Call it p_n)

Combinatorics: $p_n = P(n,n) = n!$

Recursion: Suppose p_{n-1} is the number of permutations of $n-1$ objects

When we show up with the n^{th} object, we need to place it. For each of the p_{n-1} permutations of $n-1$ objects, there are n choices of where to place the n^{th} object.

p_n - number of permutations with n objects

$\Longrightarrow \underbrace{p_n = n \cdot p_{n-1}}$, with $p_1 = 1$

abcd
3!

Example: $n = 4$: abc**d**, ab**d**c, a**d**bc, **d**abc

abc 3!

Recursion

Example: We wish to roll a 6-sided die r times to obtain a sum of k. Let $D(r,k)$ be the number of ways to obtain a sum of k from r rolls. Find an expression for $D(r,k)$.

times we are rolling the die

sum

$D(r, k)$

$D(1,1) = 1$

$D(1,7) = 0$

$D(2,3) = 2$

1+2
2+1

$D(5,8) =$

1+1+1+1+4

1+1+4+1+1

1+1+1+4+1

⋮

2+3+1+1+1

Difficult!

- Last roll is a 1
  $D(r-1, k-1)$
- Last roll is a 2
  $D(r-1, k-2)$
- Last roll is a 3
  $D(r-1, k-3)$

⋮

- Last roll is a 6
  $D(r-1, k-6)$

Recursion

Example: We wish to roll a 6-sided die r times to obtain a sum of k. Let $D(r,k)$ be the number of ways to obtain a sum of k from r rolls. Find an expression for $D(r,k)$.

(continued)

$$D(r,k) = D(r-1,k-1) + D(r-1,k-2) + D(r-1,k-3) + D(r-1,k-4) + D(r-1,k-5) + D(r-1,k-6)$$

$\vdots$

- Need to define base cases.
- Need to determine when $D(r,k) = 0$

Recursion

Example: Find a recurrence for the number of n length bit-strings that do not have two consecutive 0s. How many such bit-strings are there of length 5?

Let a_n be the number of n −length bit-strings with no consecutive 0s.

Case 1: Last bit is a 0

this must be 1

$$\underbrace{\_\ \_\ \_\ \_\ \_\ \cdots\ \_}_{a_{n-2}}\ \underline{1}\ \underline{0}$$

Case 2: Last bit is a 1

$$\underbrace{\_\ \_\ \_\ \_\ \_\ \cdots\ \_\ \_}_{a_{n-1}}\ \underline{1}$$

$$\boxed{\begin{aligned} a_1 &= 2 \\ a_2 &= 3 \\ a_n &= a_{n-1} + a_{n-2} \end{aligned}}$$

Recursion

To summarize: Constructing a recurrence for the number of valid n −length bit-strings, assuming that we know the number of valid shorter bit-strings.

- If we have a valid bit-string of length n that ends in a 1, it must have been built up from putting a 1 onto the end of a valid bit-string of length $n-1$
  - There are a_{n-1} of these.
- If we have a valid bit-string of length n that ends in a 0, it must have been built up from putting a 10 on the end of a valid bit-string of length $n-2$
  - There are a_{n-2} of these.

Recursion

The recurrence is given by:

$$a_n = a_{n-1} + a_{n-2}$$
$$a_1 = 2$$
$$a_2 = 3$$

no two consecutive 0's

Question: How many valid bit-strings are there of length 5? → What is a_5?

$a_1 = 2$

$a_2 = 3$

$a_3 = 5$

$a_4 = 8$

$\boxed{a_5 = 13}$

Recursion

Example: In the kingdom of Hyrule, currency comes in denominations of 1, 2, and 5 rupees. After purchasing a pair of iron boots from a shop, you are owed 17 rupees in change. If the shopkeeper places each gem on the counter one at a time, how many ways are there for him to make your change?

This is similar to the die-rolling problem – consider the roll sum to be the total of n rupees.

Let a_n be the number of ways to make n rupees in change.

- If the first gem is a 1-rupee piece, then there are a_{n-1} ways
- If the first gem is a 2-rupee piece, then there are a_{n-2} ways
- If the first gem is a 5-rupee piece, then there are a_{n-5} ways

So there are $\underbrace{a_n = a_{n-1} + a_{n-2} + a_{n-5}}_{\text{recurrence}}$ ways to make n rupees in change.

Recursion

Example (continued).

$a_n = a_{n-1} + a_{n-2} + a_{n-5}$ ways to make n rupees in change. Now we need base cases:

$\Longrightarrow$ Recurrence goes back 5 stages (a_{n-5}) so we probably will need 5 base cases

$a_0 = 1$

$a_1 = 1$

$a_2 = 2$

$a_3 = 3$

$a_4 = 5$

Recursion

Example: Suppose we want to find the Fibonacci numbers, which are governed by the recurrence.

$F_0 = 1$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$, for $n > 2$

- We could use the recurrence relation and a recursive approach, or we could use a dynamic programming (DP) approach.

Recursion

Dynamic Programming

- Break a problem up into sub-problems
- Solve each sub-problem exactly once and store it for later use (instead of solving again)

1 Fibonacci numbers

1.1 recursive fibonaci

```
In [1]: def r_fib(n):
            assert n>=0, "use non-negative input, dummy"

            # base cases
            if n==0: return 1
            if n==1: return 1

            # actual recursion
            return r_fib(n-1)+r_fib(n-2)
```

1.2 dynamic version

```
In [2]: def d_fib(n):
            assert n>=0, "seriously? come on. non-negative input, please!"

            # initialize
            fib = [-1]*(n+1)

            # base cases
            fib[0] = 1
            if n==0: return 1
            fib[1] = 1
            if n==1: return 1

            # actual calculation
            for ii in range(2,n+1):
                fib[ii] = fib[ii-1]+fib[ii-2]

            return fib[n]
```

1.3 Horse race!

```
In [3]: from time import time
```

```
# timing for recursive calculation of 35th fibonacci number
tbeg = time()
foo = r_fib(35)
tend = time()
print('took {} seconds to calculate recursively'.format(tend-tbeg))

# timing for dynamic calculation of 35th fibonacci number
tbeg = time()
foo = d_fib(35)
tend = time()
print('took {} seconds to calculate dynamically'.format(tend-tbeg))
```

```
took 4.940013885498047 seconds to calculate recursively
took 4.601478576660156e-05 seconds to calculate dynamically
```

---

2 Rupee change-making problem

```
In [4]: def r_change(n):
            assert n>=0, "can only make nonnegative change!"

            # base cases
            if n==0: return 1
            if n==1: return 1
            if n==2: return 2
            if n==3: return 3
            if n==4: return 5

            # recursion
            return r_change(n-1) + r_change(n-2) + r_change(n-5)
```

2.0.1 Check the answer from the example in class

```
In [5]: r_change(17)
```

```
Out[5]: 5357
```

2.1 Dynamic change-making

```
In [6]: def d_change(n):
            assert n>=0, "can only make nonnegative change!"

            # initialize
            change = [-1]*(n+1)
```

```
    # base cases
    change[0] = 1
    if n==0: return 1
    change[1] = 1
    if n==1: return 1
    change[2] = 2
    if n==2: return 2
    change[3] = 3
    if n==3: return 3
    change[4] = 5
    if n==4: return 5

    # actual dynamic calculation
    for ii in range(5, n+1):
        change[ii] = change[ii-1] + change[ii-2] + change[ii-5]

    return change[n]
```

2.1.1 Check answer from the example again, to make sure d_change is working okay

```
In [7]: d_change(17)
```

```
Out[7]: 5357
```

2.2 Horse race!

```
In [8]: from time import time

        # timing for recursive calculation of number of ways to make 35 rupees in change
        tbeg = time()
        foo = r_change(35)
        tend = time()
        print('took {} seconds to calculate recursively'.format(tend-tbeg))

        # timing for dynamic calculation of number of ways to make 35 rupees in change
        tbeg = time()
        foo = d_change(35)
        tend = time()
        print('took {} seconds to calculate dynamically'.format(tend-tbeg))
```

```
took 9.312153816223145 seconds to calculate recursively
took 5.316734313964844e-05 seconds to calculate dynamically
```

```
In [ ]:
```

Next: Recursion!