Introductions...!

Zach Dodds

Office  Olin B163
Email   dodds@cs.hmc.edu

fan of low-level AI
taker of low-quality photos
Starbucks triumph-er!

not afraid of stuffed animals!

and not good at selfies...
How I spent my summer vacation...

programming robots

visiting important landmarks!

Or, more precisely, cheering for many other folks programming robots!

My selfie-taking has gotten worse over the past couple of months!
What is this course about?

**practicing** algorithmic/programming skills

first half… until early November

**trying out** technologies/projects of interest

second half… after early November, *if you'd like*
trying out technologies/projects of interest after early November, if you'd like

Alums: What do you feel you didn't get @ HMC CS?
trying out technologies/projects of interest after early November, if you'd like

Alums: What do you feel you didn't get @ HMC CS?

- the code check-in process
- web things
- parallelizing/distributing large computations
- cmake and build systems
- web technologies (just for terminology...)
- don't try to teach web stuff
What is the first $\frac{1}{2}$ about?

practicing algorithmic/programming skills

Bessie!

Cows are the global theme of CS189's problems.
Example

Space Elevator

The cows are going to space! They plan to achieve orbit by building a sort of space elevator: a giant tower of blocks. They have $K$ ($1 \leq K \leq 400$) different types of blocks with which to build the tower. Each block of type $i$ has height $h_i$ ($1 \leq h_i \leq 100$) and is available in quantity $c_i$ ($1 \leq c_i \leq 10$). Due to possible damage caused by cosmic rays, no part of a block of type $i$ can exceed a maximum altitude $a_i$ ($1 \leq a_i \leq 40000$).

Help the cows build the tallest space elevator possible by stacking blocks on top of each other according to the rules.

**PROBLEM NAME:** elevator.X
Example

elevator.py
elevator.java
elevator.cc

Space Elevator

The cows are going to space! They plan to achieve orbit by building a sort of space elevator: a giant tower of blocks. They have K (1 <= K <= 400) different types of blocks with which to build the tower. Each block of type i has height h_i (1 <= h_i <= 100) and is available in quantity c_i (1 <= c_i <= 10). Due to possible damage caused by cosmic rays, no part of a block of type i can exceed a maximum altitude a_i (1 <= a_i <= 40000).

Help the cows build the tallest space elevator possible by stacking blocks on top of each other according to the rules.

PROBLEM NAME: elevator.X

Input

<table>
<thead>
<tr>
<th># of blocktypes</th>
<th>block height</th>
<th>max altitude</th>
<th>quantity (count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7 40 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>23 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 of them

7

8 of them

5

6 of them

2

3 of them

7
Example

Space Elevator

The cows are going to space! They plan to achieve orbit by building a sort of space elevator: a giant tower of blocks. They have $K$ (1 <= $K$ <= 400) different types of blocks with which to build the tower. Each block of type $i$ has height $h_i$ (1 <= $h_i$ <= 100) and is available in quantity $c_i$ (1 <= $c_i$ <= 10). Due to possible damage caused by cosmic rays, no part of a block of type $i$ can exceed a maximum altitude $a_i$ (1 <= $a_i$ <= 40000).

Help the cows build the tallest space elevator possible by stacking blocks on top of each other according to the rules.

PROBLEM NAME: elevator.X

Input

<table>
<thead>
<tr>
<th># of blocktypes</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>block height</td>
<td>7 40 3</td>
</tr>
<tr>
<td>max altitude</td>
<td>5 23 8</td>
</tr>
<tr>
<td>quantity (count)</td>
<td>2 52 6</td>
</tr>
</tbody>
</table>

Output

The height of the tallest tower possible

48

51
practicing algorithmic/programming skills

What

Algorithm analysis and insight
Program design and implementation

Why

Research/prototype programming
"reasonable"

Unofficial course name: CS -70
Class Meetings

alternating format

discussion sessions

• problem and program analysis
• discussion of strategy and coding tips
• deciding on functional decomposition, data structures, language facilities, and algorithms to use in teams of 2-3
• short time to work on at least 1 problem

lab sessions

• more extended team problem-solving practice: coming to the problems "cold"
• these problems count for each member of the group

• sometimes new problems, other times with known ones
• ~5-6 problems given out per week…
Course Organization

Sep 10  Welcome discussion! and DP problems ~ 5 problems
Sep 17  Lab session ~ 5 problems
Sep 24  Discussion session on graph problems ~ 5 problems
Oct  1  Lab session on graph problems ~ 5 problems
Oct  8  Guest speaker Don Chamberlin, author/inventor of SQL = 2 problems
Oct 16  Discussion session on geometry problems ~ 5 problems
Oct 22  Lab & ACM qualifying contest ~ 6 problems
Oct 29  Discussion session on something new!! ~ 5 problems
Nov  5  Final meeting: project opportunities
Nov  9  (approximate) ACM Regional contest (in Riverside...)

Rest of the term: projects or problems -- you choose --

≥ 42 problems total
You may submit problems until the end of exams...
part – but only *part* – of the motivation for CS 189:

**ACM programming contest**
USC advanced to the finals in 2011 and 2012...
75 teams...

Jackson!
active watching!
active watching!
Course webpage

A few references

Reference Links
- HMC ACM Page
- C++ & STL
- Java 1.6 API

Congratulations! to the HMC teams in the 2018 Southern California regionals. The standings out of 78 participating teams:

- 4th place -- HMC Hammer -- Ryan Brewster, Richard Porczak, and Jackson Newhouse
- 8th place -- HMC Squared -- Andrew Carter, Daniel Lubarov, and Kevin Black
- 10th place -- HMC 42 -- Emily Myers-Stanhope, Eric Alshire, and Benson Khau
- 21st place -- HMC Escher -- Fiona Tay, Jacob Bandes-Storch, and Tum Chaturapruek

Problems and progress

<table>
<thead>
<tr>
<th>Problem</th>
<th>dodds</th>
<th>0-solder</th>
<th>0-forgot</th>
<th>0-0 covariance</th>
<th>0-covphi</th>
<th>0-0 checks</th>
<th>Total</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>1.0</td>
<td>Oddz</td>
</tr>
</tbody>
</table>

Lecture Slides and Starting Code...
- Lecture 1, Fall 2012 materials (zip)
Grading

CS 189 is graded by default ... (it’s possible to take it P/F, too)

though not for CS elective credit...

Coding Guidelines

- problems can be done *any time* during the semester
- discussion of algorithms always OK
- coding should be *within teams of 1-3*
- you may use any references *except* others' solutions or partial solutions...
- use `/cs/ACM/acmSubmit <file>` to submit on knuth

<table>
<thead>
<tr>
<th># Solved (out of 42)</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>43+</td>
<td>pretty much impossible!</td>
</tr>
<tr>
<td>28-42</td>
<td>A</td>
</tr>
<tr>
<td>23-27</td>
<td>A-</td>
</tr>
<tr>
<td>20-22</td>
<td>B+</td>
</tr>
<tr>
<td>17-19</td>
<td>B</td>
</tr>
<tr>
<td>14-16</td>
<td>B-</td>
</tr>
<tr>
<td>9-13</td>
<td>C range</td>
</tr>
<tr>
<td>≤ 9</td>
<td>&lt; D range or less</td>
</tr>
</tbody>
</table>
Details

Problems are worth 150% if

- You solve them during the 4:15 - 5:30 lab sessions
- … which extend to about 11pm at night.

Language Choice?

Any *reasonable* language is OK; keep in mind that the ACM competition allows only Java, C, and C++.

Other "standard" languages for CS189 (so far):

C#, Python, Ruby, Perl, PHP, Haskell, Lua, Clojure, Lisp

you can work in teams of up to 3 people
This week's problems

Notes, starting code, slides, etc. ...

- Lecture 1, cowqueue code examples (zip)
- Fall '13 Lecture 1 slides

Problems and progress

<table>
<thead>
<tr>
<th>NAMES \ problems</th>
<th>0-smount</th>
<th>0-lazy</th>
<th>0-elevator</th>
<th>0-cowqueue</th>
<th>0-cowcash</th>
<th>0-ave</th>
<th>Total</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dodds</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>1.5</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>1.5</td>
<td>dodds</td>
</tr>
</tbody>
</table>

New to CS189? Start with this problem!

Part of the challenge is deciding which problem to tackle...

Some of this week's problems have a "dynamic programming" theme...
Max, Max, and Carl ~ dynamic programmers
Dynamic Programming

Many problems can be solved recursively...
... but with lots of repeated recursive calls!

These problems can be solved quickly with

(1) Memoization, or
(2) Dynamic programming

Idea: just don't repeat the repeated calls!
The **cowqueue** problem

**Input**

- ABACB
- AABC

Cow label sequence #1 (s1)

Cow label sequence #2 (s2)

**Output**

3

The number of the *longest common subsequence* between s1 and s2.

In this case, the longest common subsequence is **ABC** or **AAB** though the problem doesn’t require knowing these.
LCS problem

\[
LCS(i_1, i_2) = \text{length of longest common subsequence of } s_1 \text{ up to } i_1 \text{ and } s_2 \text{ up to } i_2
\]

Strategy

(1) Write a solution recursively.
(2) Then, don't make any call more than once!
LCS problem

LCS( i₁, i₂ ): 

if s₁[i₁] == s₂[i₂]:  return 1 + LCS( i₁-1, i₂-1 )  

  if the same character, count it!

else:  return max( LCS( i₁-1, i₂ ), LCS( i₁, i₂-1 ) )  

  otherwise, lose both ends and take the better result

Input

s₁ = "ABACB"  

s₂ = "AABC"  

length of longest common subsequence of s₁ up to i₁ and s₂ up to i₂
LCS code

```python
import sys
sys.setrecursionlimit(100000)

def LCS( i1, i2 ):
    """ classic LCS """

    if i1 < 0 or i2 < 0: return 0

    if s1[i1] == s2[i2]:
        return 1 + LCS(i1 - 1, i2 - 1)
    else:
        return max(LCS(i1 - 1, i2), LCS(i1, i2 - 1))

if __name__ == "__main__":
    s1 = raw_input(); L1 = len(s1)
    s2 = raw_input(); L2 = len(s2)

    result = LCS( L1-1, L2-1 )
    print result
```
### LCS idea

<table>
<thead>
<tr>
<th>string1 \ s1[:i1]</th>
<th>∅</th>
<th>A</th>
<th>AA</th>
<th>AAB</th>
<th>AABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td>AA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABA</td>
<td></td>
<td></td>
<td></td>
<td>AAB</td>
<td></td>
</tr>
<tr>
<td>ABAC</td>
<td></td>
<td></td>
<td></td>
<td>AABC</td>
<td></td>
</tr>
<tr>
<td>ABACB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LCS(4,3)</td>
</tr>
</tbody>
</table>

**Input**

- \( s_1 = "ABACB" \)
- \( s_2 = "AABC" \)

**string2** \( s_2[:i2] \)

- \( i_1 \)
- \( i_2 \)
**LCS idea**

<table>
<thead>
<tr>
<th>string1</th>
<th>s1[:i1]</th>
<th>s1</th>
<th>A</th>
<th>AA</th>
<th>AAB</th>
<th>AABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊘</td>
<td>⊘</td>
<td>⊘</td>
<td>A</td>
<td>AA</td>
<td>AAB</td>
<td>AABC</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>A</td>
<td>A</td>
<td>AA</td>
<td>AAB</td>
<td>AABC</td>
</tr>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LCS(3,3)</td>
</tr>
<tr>
<td>ABACB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LCS(4,2)</td>
<td>LCS(4,3)</td>
</tr>
</tbody>
</table>

s1 = "ABACB"  
Input  
s2 = "AABC"

\[s_1 = \text{"ABACB"}, \quad s_2 = \text{"AABC"}\]
**LCS idea**

Input:
- $s_1 = "ABACB"$
- $s_2 = "AABC"

<table>
<thead>
<tr>
<th>\emptyset</th>
<th>\emptyset</th>
<th>A</th>
<th>AA</th>
<th>AAB</th>
<th>AABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABA</td>
<td></td>
<td></td>
<td></td>
<td>LCS(2,2)</td>
<td></td>
</tr>
<tr>
<td>ABAC</td>
<td></td>
<td></td>
<td></td>
<td>LCS(3,1)</td>
<td></td>
</tr>
<tr>
<td>ABACB</td>
<td></td>
<td></td>
<td></td>
<td>LCS(4,2)</td>
<td>LCS(4,3)</td>
</tr>
</tbody>
</table>

The LCS values for each substring of $s_1$ and $s_2$ are calculated and marked in the table.
**LCS idea**

\[
\begin{array}{c|c|c|c|c|c}
\hline
& \emptyset & A & AA & AAB & AABC \\
\hline
\emptyset & & & & & \\
\hline
A & & & & LCS(1,2) & \\
\hline
AB & & & LCS(2,1) & LCS(2,2) & \\
\hline
ABA & LCS(3,0) & LCS(3,1) & & & \\
\hline
ABAC & LCS(3,0) & LCS(3,1) & & & LCS(3,3) \\
\hline
ABACB & LCS(3,0) & LCS(3,1) & & & LCS(3,3) \\
\hline
\end{array}
\]

String 1: `s1 = "ABACB"

Input

String 2: `s2 = "AABC"

\[i_1\]

\[i_2\]
### LCS idea

![LCS Diagram]

- **Input**
  - \( s_1 = "ABACB" \)
  - \( s_2 = "AABC" \)

### LCS Table

<table>
<thead>
<tr>
<th>( s_1[i_1] )</th>
<th>( \emptyset )</th>
<th>( A )</th>
<th>( AA )</th>
<th>( AAB )</th>
<th>( AABC )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>LCS(-1,-1)</td>
<td>LCS(-1,0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A</strong></td>
<td></td>
<td>LCS(0,0)</td>
<td>LCS(0,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AB</strong></td>
<td>LCS(1,-1)</td>
<td>LCS(1,0)</td>
<td></td>
<td>LCS(1,2)</td>
<td></td>
</tr>
<tr>
<td><strong>ABA</strong></td>
<td></td>
<td>LCS(2,0)</td>
<td>LCS(2,1)</td>
<td>LCS(2,2)</td>
<td></td>
</tr>
<tr>
<td><strong>ABAC</strong></td>
<td>LCS(3,-1)</td>
<td>LCS(3,0)</td>
<td>LCS(3,1)</td>
<td></td>
<td>LCS(3,3)</td>
</tr>
<tr>
<td><strong>ABACB</strong></td>
<td></td>
<td></td>
<td></td>
<td>LCS(4,2)</td>
<td>LCS(4,3)</td>
</tr>
</tbody>
</table>

- **LCS(4,3)**
- **LCS(3,3)**
- **LCS(2,2)**
- **LCS(1,2)**
- **LCS(2,1)**
- **LCS(1,1)**
- **LCS(0,1)**
- **LCS(1,0)**
- **LCS(2,0)**
- **LCS(1,-1)**
- **LCS(0,-1)**
- **LCS(-1,-1)**
- **LCS(-1,0)**
- **LCS(-2,0)**
- **LCS(-1,-2)**
- **LCS(-2,-1)**
LCS idea

\[
\begin{align*}
\text{s1} &= "ABACB" \\
\text{Input} & \quad \text{s2} = "AABC"
\end{align*}
\]

Collisions!
LCS, memoized

Put results in a dictionary.
Look up instead of recomputing.

```python
# This is the "memoizing" dictionary of all distinct calls.
# Each distinct call is made only once and stored here.
D = {}

def LCS( i1, i2):
    """ classic LCS """
    if i1 < 0 or i2 < 0: return 0  # base cases
    if (i1,i2) in D: return D[ (i1,i2) ]  # already done!
    if s1[i1] == s2[i2]:
        result = 1 + LCS(i1-1, i2-1)
    else:
        result = max( LCS(i1-1, i2), LCS(i1, i2-1) )
    D[ (i1,i2) ] = result  # memo-ize it!
    return result  # before returning

if __name__ == "__main__":
    s1 = raw_input(); L1 = len(s1)
    s2 = raw_input(); L2 = len(s2)
    result = LCS( L1-1, L2-1 )
    print result
```
import sys; sys.setrecursionlimit(100000)

class memoize:
    def __init__(self, function):
        self.function = function
        self.memoized = {}

    def __call__(self, *args):
        try:
            return self.memoized[args]
        except KeyError:
            self.memoized[args] = self.function(*args)
        return self.memoized[args]

@memoize
def LCS(i1, i2):    # slow, recursive f'n here
**LCS, DP'ed**

Compute the table of results, bottom-up!

<table>
<thead>
<tr>
<th>s1 = &quot;ABACB&quot;</th>
<th>Input</th>
<th>s2 = &quot;AABC&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i1</td>
<td>i2</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>string1 s1[:i1]</th>
<th>string2 s2[:i2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>⊖</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>AA</td>
</tr>
<tr>
<td></td>
<td>AAB</td>
</tr>
<tr>
<td></td>
<td>AABC</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td></td>
</tr>
<tr>
<td>ABA</td>
<td></td>
</tr>
<tr>
<td>ABAC</td>
<td></td>
</tr>
<tr>
<td>ABACB</td>
<td></td>
</tr>
</tbody>
</table>
```
**LCS, DP'ed**

Compute the table of results, bottom-up!

---

Input:

<table>
<thead>
<tr>
<th></th>
<th>s1 = &quot;ABACB&quot;</th>
<th></th>
<th>s2 = &quot;AABC&quot;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>s2 = &quot;AABC&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2</td>
</tr>
</tbody>
</table>

---

```python
string2 s2[:i2]

input s1[:i1]

string1 |
| Φ       |
|         |
| A       |
| A       |
| ABB     |
| ABAC    |
| ABACB   |

---

```python

```python
if __name__ == "__main__":
    
    s1 = raw_input(); L1 = len(s1)
    s2 = raw_input(); L2 = len(s2)

    DP = [ [0]*(L2+2) for i1 in range(L1+2) ]

    for i1 in range(L1):
        for i2 in range(L2):
            if s1[i1] == s2[i2]:
                DP[i1][i2] = 1 + DP[i1-1][i2-1]
            else:
                DP[i1][i2] = max( DP[i1][i2-1], DP[i1-1][i2] )

    result = DP[L1-1][L2-1]

    # for row in DP:
    # print row

    print result
```
Jotto!

A word-guessing game similar to mastermind...

Sophs | JRs | SRs | POM-CMC-SCR-PTZ | other
---|---|---|---|---

This term's first class to guess another's word earns 1 problem...
This term's last class to have its word guessed earns 1 problem...
Recent-past Jotto finale:

<table>
<thead>
<tr>
<th>Win</th>
<th>Sophs</th>
<th>Jrs</th>
<th>Srs</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>icily 0</td>
<td>icily 0</td>
<td>icily 1</td>
<td>icily 1</td>
</tr>
<tr>
<td></td>
<td>strep 2</td>
<td>strep 2</td>
<td>strep 2</td>
<td>strep 1</td>
</tr>
<tr>
<td></td>
<td>spork 1</td>
<td>spork 3</td>
<td>spork 0</td>
<td>spork 0</td>
</tr>
<tr>
<td></td>
<td>spend 2</td>
<td>spend 2</td>
<td>spend 2</td>
<td>spend 2</td>
</tr>
<tr>
<td></td>
<td>peeps 2</td>
<td>peeps 1</td>
<td>peeps 2</td>
<td>peeps 1</td>
</tr>
<tr>
<td></td>
<td>furls 1</td>
<td>furls 1</td>
<td>furls 0</td>
<td>furls 1</td>
</tr>
<tr>
<td></td>
<td>Ghost 2</td>
<td>Ghost 1</td>
<td>Ghost 1</td>
<td>Ghost 0</td>
</tr>
<tr>
<td></td>
<td>Tanks 2</td>
<td>Tanks 1</td>
<td>Tanks 2</td>
<td>Tanks 1</td>
</tr>
<tr>
<td></td>
<td>Gecko 2</td>
<td>Gecko 1</td>
<td>Gecko 1</td>
<td>Gecko 1</td>
</tr>
</tbody>
</table>
Try 1-2 of these tonight!

Notes, starting code, slides, etc. ...

- Lecture 1, cowqueue code examples (zip)
- Fall '13 Lecture 1 slides

Problems and progress

<table>
<thead>
<tr>
<th>NAMES \ problems</th>
<th>0-smount</th>
<th>0-lazy</th>
<th>0-elevator</th>
<th>0-cowqueue</th>
<th>0-cowcash</th>
<th>0-ave</th>
<th>Total</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dodds</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>1.5</td>
<td>Not Yet</td>
<td>Not Yet</td>
<td>1.5</td>
<td>dodds</td>
</tr>
</tbody>
</table>

Poster time!