Lecture 8

Text justification: In the problem of text justification, we are given a long text which is a sequence of words separated by a string. We have a book that can fit pageWidth characters per line, and we would like to organize the text on separate lines to minimize the ugliness of the layout. We imagine we have some penalty function which tells us how ugly a line looks (the bigger the penalty, the uglier the line). In particular in today's lecture, if we don't use n possible character slots on a given line, the penalty is n^3. The penalty of an entire layout is the sum of penalties over all lines. We would like to fit words on our pages so as to minimize the maximum penalty. This can be done via dynamic programming.

```
def penalty(length, pageWidth):
    if length==0:
        return 0
    n = pageWidth - length
    return n**3

def memoize(L, at, length, pageWidth, mem):
    if at==len(L):
        return penalty(length, pageWidth)
    elif mem[at][length] != -1:
        return mem[at][length]
    mem[at][length] = pageWidth**3
    # start a new line
    if length > 0:
        mem[at][length] = penalty(length, pageWidth) + memoize(L,
                                 at+1, len(L[at]), pageWidth, mem)
    # put the word on this line, and we need a space to separate it
    # from the previous word
    if len(L[at])+1 <= pageWidth - length:
        mem[at][length] = min(mem[at][length],
                              memoize(L, at+1, length+len(L[at])+1,
                                      pageWidth, mem))
    return mem[at][length]

def textJustify(text, pageWidth):
    L = text.split(' ')
    for word in text:
        if len(word) > pageWidth:
            return 'Impossible'
    mem = []
    for i in xrange(len(L)):
        x = []
        for j in xrange(pageWidth+1):
```

```
        x += [-1]
    mem += [x]
return memoize(L, 1, len(L[0]), pageWidth, mem)
```