6.055J/2.038J (Spring 2009)

Solution set 2

Do the following warmups and problems. Due in class on ***Monday, 09 Mar 2009****.*

Open universe: *Collaboration, notes, and other sources of information are* ***encouraged.*** *However, avoid looking up answers until you solve the problem (or have tried hard). That policy helps you learn the most from the problems.*

Homework will be graded with a light touch: P (made a reasonable effort), D (did not make a reasonable effort), or F (did not turn in).

Several questions reference `decline.txt`: It is the plain-text file on the course website that contains volume 1 of Gibbon's *Decline and Fall*. It is also available (like any other file on the course website) on any Athena machine as `~6.055/data/decline.txt` where the notation `~6.055` refers to the home directory of the `6.055` user.

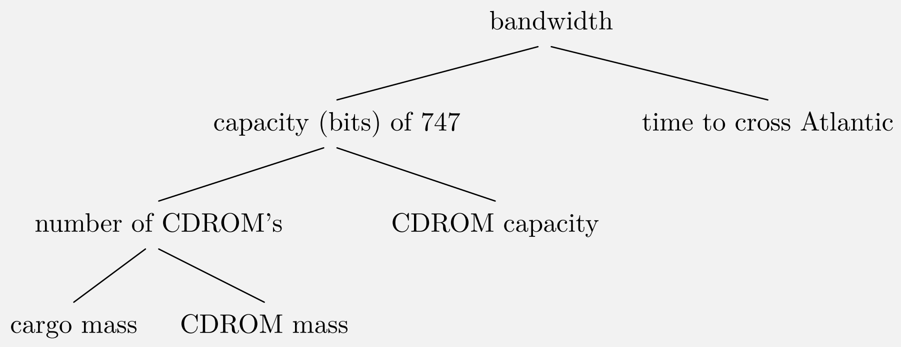
On `linux.mit.edu` (the Athena GNU/Linux machine), an (American) English dictionary lives in `/usr/share/dict/words`

Warmups

1. Bandwidth

Estimate the bandwidth (bits/s) of a 747 crossing the Atlantic filled with CDROM's, and explain your estimate using a tree.

Divide and conquer! Here's a tree on which to fill values:



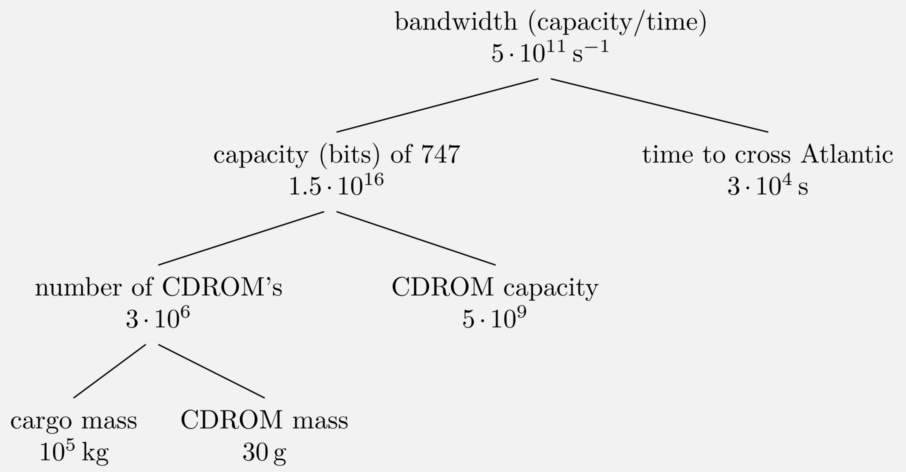
First I estimate the cargo mass. A 747 can easily carry about 400 people, each person having a mass (with luggage) of, say 140 kg. The total mass is

$$m \sim 400 \times 140\,\text{kg} \sim 6 \cdot 10^4\,\text{kg}.$$

A special cargo plane, with no seats or other frills for passengers, probably can carry 10^5 kg.

Here are the other estimates. A CDROM's mass is perhaps one ounce or 30 g. So the number of CDROM's is $3 \cdot 10^6$. The capacity of a CDROM is 600 MB or about $5 \cdot 10^9$ bits. The time to cross the Atlantic is about 8 hours or $3 \cdot 10^4$ s.

Now propagate the values toward the root of the tree:



The bandwidth is 0.5 terabits per second.

Despite the large bandwidth offered by a 747 carrying CDROM's (not to mention DVDROM's), trans-Atlantic Internet connections go via undersea fiber-optic cables. Low latency is important!

2. Integrals

Evaluate these definite integrals:

a. $\displaystyle\int_{-10}^{10} x^3 e^{-x^2}\,dx$

The integrand $x^3 e^{-x^2}$ is antisymmetric: Replacing x by $-x$ changes the function's sign. Therefore integrating it over a symmetric range such as -10 to 10 produces zero.

b. $\displaystyle\int_{-\infty}^{\infty} \frac{x^3}{1+7x^2+18x^8}\,dx$

This integrand is also antisymmetric, so integrating it over a symmetric range such as $-\infty$ to ∞ produces zero.

[As a physics undergraduate, I spent many hours with the table of integrals that we knew affectionately as Gradshteyn. The table was so massive and complete that when we could not locate an integral in it, we suspected that the integral should be zero and went looking for the symmetry.]

3. Explain a Unix pipeline

What does this pipeline do?

```
ls -t | head | tac
```

The `ls -t` lists the files and subdirectories in a directory ordered by modification time with the most recently modified files at the beginning. The `head` selects the first ten lines, which means the first ten names. The `tac` reverses the order of the lines, so the 10th-most-recently-modified file (or subdirectory) comes first, then the 9th-most-recently-modified file, etc. with the most-recently-modified file at the end of the list.

4. Different word counts by different methods

Why do the following two commands produce different counts:

```
$ wc -w < decline.txt
268863
$ tr -cs 'a-zA-Z' '\n' < decline.txt  | wc -l
264164
```

The second command pipline first turns every non-letter character into a newline, removes the repeated newlines, and then counts the lines. It does almost the same operation as counting the words directly (the first pipeline), but not quite. For example, the first pipeline will count numbers, such as 257, whereas the second pipeline will turn each digit into a newline, and then squeeze out the repeated newlines, so 257 won't get counted.

5. Spell checkers

Why, from the point of view of making abstractions, is it worth separating spell checking into its own program rather than building it into a word processor?

Many other activities, such as writing an email, require spell checking. If the spell checker is embedded into the word processor, then the email-composition program, and many similar programs, need to have its own spell checker embedded into it.

UNIX therefore adopts the abstraction approach: The spell checker `ispell` will check any text file. So, it works with TEX files or with email messages (composed as plain text in an editor, for example).

A related point: To make the spell checker broadly useful, the various programs should use plain-text files, rather than each having its own binary format (such as the ones used by word processors such as OpenOffice).

Problems

6. Number sum

Use symmetry to find the sum of the integers between 200 and 300 (inclusive).

Reversing the order of the terms is the symmetry operation because the sum is the same in reverse:

$$200 + 201 + 202 + \cdots + 300 = 300 + 299 + 298 + \cdots + 200.$$

Add these sums as follows:

$$\begin{array}{r} 200+201+202+\cdots+300 \\ +\,300+299+298+\cdots+200 \\ \hline =500+500+500+\cdots+500. \end{array}$$

There are 101 copies of the 500, so this duplicated sum is $500 \times 101 = 50500$. The original sum is one-half of the duplicated sum, so it is 25250.

A quick confirmation is the following Unix pipeline:

```
seq 200 300 | awk '{total += $1}; END {print total};'
```

which produces 25250.

7. Searching text files

a. What English word, besides `angry`, ends in `gry`?

Humans are much worse than computers at this question, because we store words not by their endings but more by their beginnings and meanings. For a computer, it's all bit strings, and computers don't care whether the bit string happens at the beginning or end of the word (and there's no meaning).

The regular expression that matches words ending in `gry` is `gry$`. In the following pipeline, the first `grep` finds all those words, and the second `grep` excludes `angry` from the list:

```
grep 'gry$' /usr/share/dict/words | grep -v '^angry$'
```

The result is just one line: 'hungry'.

b. How many times does the word `Empire` (uppercase E, then all lowercase) occur in `decline.txt`? Give a pipeline that does the counting.

Divide and conquer! First turn all non-letters into newlines (squeezing out repeated newlines); second, look for lines that exactly match 'Empire'; and third, count the lines. Those three stages are the three stages of the following pipeline:

```
tr -cs 'a-zA-Z' '\012' < ./data/decline.txt | grep '^Empire$' | wc -l
```

It produces '37'.

Optional

8. Email indexer

Design a set of shell scripts for doing quick keyword searches of a large database of emails. Assume that each email is stored in its own plain-text file. Perhaps one shell script generates an index, and a second script searches the index.

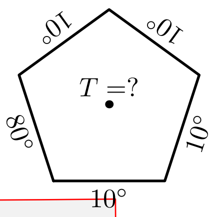
9. Running time

Ordinary long multiplication requires $O(n^2)$ digit-by-digit multiplications. Show that the Karatsuba multiplication method explained in lecture requires $O(n^{\log_2 3}) \approx O(n^{1.58})$ digit-by-digit multiplications.

10. Heat equation

In lecture we used symmetry to argue that the temperature at the center of the metal sheet is the average of the temperatures of the sides.

Check this result by making a simulation or, if you are bold but crazy, by finding an analytic solution of the heat equation.



This simulation in Python was written to be clear though not necessarily efficient. It uses a lattice to approximate the continuous sheet, and implements so-called relaxation: At each step, the temperature at each point is replaced by the average temperature of the neighbors. The main complications are:

1. The edges of the pentagon are held at fixed temperatures (10 degrees for four edges and 80 degrees for the fifth edge). However, the relaxation step does not maintain those fixed values. So they are re-imposed after each sweep through the lattice.

2. Only one of the edges lies along a coordinate direction. The other four edges have funny slopes, and need to be rasterized. It is the identical problem to rendering lines on a laser printer: Which pixels get the toner? Bresenham's algorithm does the rasterization.

```
# Relaxation simulation of the temperature at the center of the pentagon.
# Four edges are held at 10 degrees, and the fifth at 80 degrees.

from scipy import *

# rounds to nearest integer, and returns an integer
def intround(f):
    return int(round(f))

# Bresenham's algorithm: returns list of lattice points on the line connecting
# r1 and r2
def line(r1, r2):
    x0, y0 = intround(r1[0]), intround(r1[1])
    x1, y1 = intround(r2[0]), intround(r2[1])
    points = []
    steep = abs(y1 - y0) > abs(x1 - x0)
    if steep:
        x0,y0 = y0,x0
        x1,y1 = y1,x1
    if x0 > x1:
        x0,x1=x1,x0
        y0,y1=y1,y0
    deltax = x1 - x0
    deltay = abs(y1 - y0)
    error = 0
    deltaerr = float(deltay) / deltax
    y = y0
    if y0 < y1:
        ystep = 1
    else:
        ystep = -1
    for x in range(x0,x1+1):
        if steep:
            points.append((y,x))
        else:
            points.append((x,y))
        error += deltaerr
        if error >= 0.5:
            y += ystep
            error -= 1.0
    return points

def complex2pair(c):
    return (real(c),imag(c))

def set_edge_temps(grid):
    for e in lo_temp_edges:
        grid[e[0]][e[1]] = 10
    for e in hi_temp_edges:
        if e in lo_temp_edges:
            grid[e[0]][e[1]] = 45 # corner joining high- and lo-temp edges
        else:
            grid[e[0]][e[1]] = 80

angle    = 72.0/180*pi          # 72 degrees
# use complex plane to find the vertices
r        = exp(angle*1j)
# pentagon vertices in the complex plane, with first vertex duplicated
# at the end of the list
corners  = array([r**(i+0.25) for i in range(6)])
# translate pentagon into first quadrant
corners -= complex(min(real(corners)), min(imag(corners)))
corners *= 50                   # grid spacing (bigger means finer spacing)
center   = sum(corners[0:5])/5  # center of pentagon

# use Bresenham's algorithm to find the lattice points on the edges
lo_temp_edges = []
hi_temp_edges = []
for i in range(4):
    lo_temp_edges += line(complex2pair(corners[i]), complex2pair(corners[i+1]))
hi_temp_edges = line(complex2pair(corners[4]), complex2pair(corners[5]))
```

```
# figure out the grid dimensions
max_x = max([r[0] for r in lo_temp_edges+hi_temp_edges])
max_y = max([r[1] for r in lo_temp_edges+hi_temp_edges])
grid  = zeros((max_x+1,max_y+1))

dirs = [(-1,0), (1,0), (0,1), (0,-1)]
while True:
    newgrid = zeros((max_x+1,max_y+1))
    set_edge_temps(grid)        # impose constraint
    for x in range(max_x+1):    # relax each location to avg of its neighbors
        for y in range(max_y+1):
            total = n = 0
            for d in dirs:      # use each neighbor that's within the grid
                try:
                    total += grid[x+d[0]][y+d[1]]
                    n += 1
                except: pass    # that neighbor was not inside the grid
            newgrid[x][y] = total/n # but save new value in a new grid
    grid, newgrid = newgrid, grid   # swap new and old grid
    # print temperature at the center of the pentagon
    print grid[intround(real(center))][intround(imag(center))]
```