Lecture 20

Shortest Paths: The Last Lab

Decimal vs. Fraction
Shortest Paths Problem

Given:
• Network: \((N, A)\)
• Costs = Travel times: \(c_{ij}, (i, j) \in A\)
• Home (root): \(r \in N\)

Problem: Find shortest path from every node in \(N\) to root.
Dijkstra’s Algorithm

Notation:
• Put $v_i = \text{min time from } i \text{ to } r$
  - Called label in networks literature.
  - Called value in dynamic programming literature.
• $F = \text{set of finished nodes (labels are set)}$.
• $h_i, i \in N = \text{next node to visit after } i \text{ (heading)}$.

Dijkstra’s Algorithm:
• Initialize:
  $F = \emptyset$
  
  $v_i = \begin{cases} 
  0 & j = r \\
  \infty & j \neq r 
  \end{cases}$
• Iterate:
  - Select unfinished node with smallest $v_k$. Call it $j$.
  - Add $j$ to set of finished nodes $F$.
  - For each unfinished node $i$ having an arc connecting it to $j$:
    - If $c_{ij} + v_j < v_i$, then set
      
      $$v_i = c_{ij} + v_j$$
      $$h_i = j$$
  • Stop: when no unfinished nodes remain
Dijkstra’s Algorithm - Complexity

• Each iteration finishes one node: \( m \) iterations
• Work per iteration:
  - Selecting an unfinished node:
    - Naively, \( m \) comparisons.
    - Using appropriate data structures, a heap, \( \log m \) comparisons.
  - Update adjacent arcs
• Overall: \( m \log m + n \)
Fractions

Two choices:

Create a `Fraction` class consisting of two integers, `num` and `den`, and write all code using `Fraction` instead of `double`.

Write all code using `double` to represent numbers. Convert from/to fraction format on input/output.

- **Advantages:**
  - Seems safer - one implements exactly what one expects: greatest common denominator, reduction to simplest form, etc.
  - Simple to program.
  - Very little danger of overflow.
  - Easy to switch between decimal and fraction format.

- **Disadvantages:**
  - Integer overflow during temporary computation is a danger.
  - Code is hard to read: e.g. `z = x.add(y)` to add `Fraction x` and `y` and store in `z`.
  - Could make small mistakes.
Converting Reals to Fractions

Two methods:
- **Brute Force**: Start with $\text{den} = 1$ and try each possible $\text{den}$ until the associated $\text{num}$ is an integer (with a small tolerance). This works but is terribly inefficient.
- **Continued Fractions**: Represent a real number $x$ by its continued fraction expansion:

$$x = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{b_3 + \ldots}}}$$

Truncate after a finite number of terms. This method is amazingly efficient.

**Computing the $b_j$’s:**
- Put $t_0 = x$
- Put $b_j = \text{greatest\_integer}(t_j)$
- Put $t_{j+1} = 1/(t_j - b_j)$.
- Repeat.
Rationalizing Continued Fractions

How many terms? Unlike series expansions, you can’t just evaluate a continued fraction from left to right stopping when the change gets small.

The obvious way to compute starts with a blind guess of how many terms to use, then starts at the right and works back up to the left.

But there is a beautiful way to compute from left to right.

Let:

\[
\frac{A_j}{B_j} = b_0 + \frac{1}{b_1 + \frac{1}{b_2 + \ldots + \frac{1}{b_j}}} \]

Initialize:

\[
\begin{align*}
A_{-1} &= 1 & A_1 &= b_0 \\
B_{-1} &= 0 & B_1 &= 1
\end{align*}
\]

Compute each ratio successively:

\[
\begin{align*}
A_j &= b_{j-1}A_{j-1} + A_{j-2} \\
B_j &= b_{j-1}B_{j-1} + B_{j-2}
\end{align*}
\]

Proof by induction:

First check \( j = 1 \):

\[
\frac{A_1}{B_1} = b_0 + \frac{1}{b_1} = \frac{b_1b_0 + 1}{b_1} = \frac{b_1A_0 + A_{-1}}{b_1B_0 + B_{-1}}
\]

Now assume true for all \( 1, 2, \ldots, j - 1 \) and check it for \( j \).
Method contFrac in class Format

static public String contFrac(int width, int precision, double t) {
    int bj=0, Aj=0, Aj1, Aj2, Bj=1, Bj1, Bj2, num, den;
    boolean pos;
    double tj, maxDen = Math.pow(10, precision);
    String numstr0, numstr;
    if (t >= 0) { pos = true;  tj =  t; }        
    else        { pos = false; tj = -t; }        
    bj = (int) (tj+1.0e-12);
    tj = 1/(tj-bj);
    Aj = bj; Aj1 = 1;
    Bj = 1;  Bj1 = 0;
    num = Aj;
    den = Bj;
    if (!pos) {num = -num;} 
    numstr0 = "";
    numstr = fracString(num, den, width, precision);
    while (Math.abs(t - Aj/(double)Bj) > 1.0e-12
        && numstr.length() < width && Bj < maxDen ) {
        Aj2 = Aj1; Aj1 = Aj;
        Bj2 = Bj1; Bj1 = Bj;
        bj = (int) (tj+1.0e-12);
        tj = 1/(tj-bj);
        Aj = bj*Aj1 + Aj2;
        Bj = bj*Bj1 + Bj2;
        num = Aj;
        den = Bj;
        if (!pos) {num = -num;} 
        numstr0 = numstr;
        numstr = fracString(num, den, width, precision);
    }
    return numstr0;
}
A Test Program

```java
import myutil.*;

public class ContFrac {
    public static void main(String[] args) {
        double x = Math.PI;

        /****************************************************
         * using static formatting methods                  *
        /****************************************************
        System.out.println();
        System.out.println("Using static formatting methods");

        System.out.println();
        System.out.println("pi = " + Format.floating(10, 5, x));
        for (int j=0; j<10; j++) {
            System.out.println("pi = " + Format.contFrac(2*j+1, j, x));
        }
    }
}
```