Programming in R
An Introduction

Biostatistics 615
Describe 3 algorithms for tackling the connectivity problem

- Quick Find
- Quick Union
- Weighted Quick Union
Pictorial Comparison
Quick Find   Quick Union   Weighted
Quick Find Algorithm in C

```c
#define N 1000

int main()
{
    int i, p, q, set, a[N];  // Variable declarations

    for (i = 0; i < N; i++)
        a[i] = i;

    while (scanf(" %d %d", &p, &q) == 2)  // Loop through connections
    {
        if (a[p] == a[q]) continue;  // FIND

        set = a[p];  // UNION
        for (i = 0; i < N; i++)
            if (a[i] == set)
                a[i] = a[q];

        printf("%d %d is a new connection\n", p, q);
    }

    return 0;
}
```
Quick Union Algorithm in C

```c
// Loop through connections on input
while (scanf(" %d %d", &p, &q) == 2)
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    // Find
    for (i = a[p]; i != a[i]; i = a[i]) ;
    for (j = a[q]; j != a[j]; j = a[j]) ;
    if (i == j) continue;

    // Union
    a[i] = j;

    printf("%d %d is a new connection\n", p, q);
}
```
// Initialize weights
for (i = 0; i < N; i++)
    weight[i] = 1;

// Loop through connections on input
while (scanf(" %d %d", &p, &q) == 2)
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    // Find
    for (i = a[p]; i != a[i]; i = a[i]);
    for (j = a[q]; j != a[j]; j = a[j]);
    if (i == j) continue;

    // Union
    if (weight[i] < weight[j])
        { a[i] = j; weight[j] += weight[i]; }
    else
        { a[j] = i; weight[i] += weight[j]; }

    printf("%d %d is a new connection\n", p, q);
}
Today

- Introduce the R programming language
  - Syntax and constructs
  - Some useful functions
  - Example: Implementations of UNION / FIND algorithms
The R Project

- An environment for statistical computing and graphics
  - Free software
- Associated with a simple programming language
  - Similar to S and S-plus
- www.r-project.org
R Programming Language

- Interpreted language
- We will consider
  - Syntax and common constructs
  - Commonly used functions
Interactive R

- R defaults to an interactive mode
- A prompt “>” is presented to users
- Each input expression is evaluated…
- … and a result returned
R as a Calculator

> 1 + 1
[1] 2
> 2 + 3 * 4
[1] 14
> 3 ^ 2
[1] 9
> exp(1)
[1] 2.718282
> pi
[1] 3.141593
> 2*pi*6378  # Circumference of earth at equator (in km)
[1] 40074.16
> sqrt(10)
[1] 3.162278
R as a Smart Calculator

```r
> x <- 1
> y <- 3
> z <- 4
> x * y * z
[1] 12
> X * Y * Z
Error: Object "X" not found
> cube <- function(x) x * x * x
> cube(y)
[1] 27
> cube(y + z)
[1] 343
```
R does a lot more!

- Definitely not just a calculator
- R thrives on vectors
- R has many built-in statistical and graphing functions
Defining Vectors

\[
\begin{align*}
&> \text{rep}(1, 10) \\
&[1] 1 1 1 1 1 1 1 1 1 1 \\
&> \text{seq}(2, 6) \\
&[1] 2 3 4 5 6 \\
&> \text{seq}(4, 20, \text{by}=4) \\
&[1] 4 8 12 16 20 \\
&> x \leftarrow \text{c}(2, 0, 0, 4) \\
&> y \leftarrow \text{c}(1, 9, 9, 9) \\
&> x + y \\
&[1] 3 9 9 13 \\
&> x \times 4 \\
&[1] 8 0 0 16 \\
&> \text{sqrt}(x) \\
&[1] 1.414214 0.000000 0.000000 2.000000
\end{align*}
\]
Accessing Vectors

> x[1]
[1] 2
> x[-1]
[1] 0 0 4
> x[1] <- 3 ; x
[1] 3 0 0 4
> x[-1] <- 5 ; x
[1] 3 5 5 5
> y < 9
[1] TRUE FALSE FALSE FALSE
> y [4] <- 1
> y < 9
[1] TRUE FALSE FALSE TRUE
> y [y < 9] <- 2
> y
[1] 2 9 9 2
Some Useful R Functions

- `rnorm(n)` - random deviates
- `summary(x)` - summary statistics
- `ls()` - current objects
- `hist(x)` - histograms

- `help(function)`, `help.search("topic")`
Example...

```r
> x <- rnorm(1000)
> y <- rnorm(1000) + x
> summary(y)
     Min.  1st Qu.   Median      Mean  3rd Qu.     Max.  
-4.54800 -1.11000 -0.06909 -0.09652  0.86200  4.83200  
> var(y)
[1] 2.079305
> hist(x, col="lightblue")
> plot(x,y)
```
Programming Constructs

- Grouped Expressions
- Control statements
  - if … else …
  - for loops
  - repeat loops
  - while loops
  - next, break statements
Grouped Expressions

\{expr_1; expr_2; \ldots \}

- Valid wherever a single expression would be used
- Return the result of last expression evaluated
if ... else ...

```c
if (expr_1) expr_2 else expr_3
```

- The first expression should return a single logical value
  - Operators && or || may be used
- Conditional execution of code
Example: if ... else ...

# Weighted Quick Find UNION
if (weight[i] < weight[j])
    { a[i] <- j; weight[j] <- weight[j] + weight[i] } 
else
    { a[j] <- i; weight[i] <- weight[j] + weight[i] }
for

```r
for (name in expr_1) expr_2
```

- Name is the loop variable
- `expr_1` is often a sequence
  - e.g. 1:20
  - e.g. `seq(1, 20, by = 2)`
Example: for

```r
# Sample M random connections in a set of N objects
for (i in 1:M)
{
  p = sample(N, 1)
  q = sample(N, 1)

  # Use QUICK-FIND to check if objects connected
  if (a[p] == a[q])
    next

  # Additional processing as needed...
}
```
repeat

repeat expr

- Continually evaluate expression
- Loop must be terminated with break statement
Example: repeat

# Sample random connections in a set of N objects
# until a new non-redundant connections are found
repeat
{
  # keep track of total connections sampled
  m <- m + 1

  # sample a new connection
  p = sample(N, 1)
  q = sample(N, 1)

  # Use QUICK-FIND to check if objects connected
  if (a[p] != a[q])
    break;
}
while

while (expr_1) expr_2

- While expr_1 is true, repeatedly evaluate expr_2

- break and next statements can be used within the loop
Example: while

# Sample M random connections in a set of N objects
# or stop early if new connection is found
while (m < M)
{
  # keep track of total connections sampled
  m <- m + 1

  # sample a new connection
  p = sample(N, 1)
  q = sample(N, 1)

  # Use QUICK-FIND to check if objects connected
  if (a[p] != a[q])
    break;
}
Function definitions

name <- function(arg1, arg2, ...)
expression

- Arguments can be assigned default values:
  arg_name = expression

- Return value is the last evaluated expression or can be set explicitly with return()
Random Generation in R

- Generates pretty good random numbers
- Relevant functions
  - `set.seed(seed)`
  - `sample(N, 1)`
  - `sample(array, size)`
  - `runif(n, min=0, max=1)`
  - `rnorm(n, mean=0, var=1)`
  - `rexp(n, rate = 1)`
- **Use `sample` to generate random connections**
Example: Quick Union Function

QuickUnion <- function( N = 100, M = 100)
{
    a <- seq(1, N) # initialize array

    for (dummy in seq(1,M)) # for each connection
    {
        p <- sample(N, 1) # sample random objects
        q <- sample(N, 1)

        # check if connected
        i = a[p]; while (a[i] != i) i <- a[i]
        j = a[q]; while (a[j] != j) j <- a[j]

        if (i == j)
            next

        a[i] = j # update connectivity array
    }
}
Benchmarking a function

- To conduct empirical studies of a function's performance, we don't always need a stopwatch.

- Relevant functions
  - `Sys.time()` gives current time
  - `difftime(stop, start)` difference between two times
Example: Quick Find Function

QuickFind <- \textbf{function}( N = 100, M = 100)
{
    \texttt{a} <- \texttt{seq}(1, N) \quad \# \text{initialize array}

    \texttt{for} (\texttt{dummy} \texttt{in} \texttt{seq}(1,M)) \quad \# \text{for each connection}
    {
        \texttt{p} <- \texttt{sample}(N, 1) \quad \# \text{sample random objects}
        \texttt{q} <- \texttt{sample}(N, 1)

        \texttt{if} (\texttt{a}[\texttt{p}] == \texttt{a}[\texttt{q}]) \quad \# \text{check if connected}
            \texttt{next}

        \texttt{a}[\texttt{a} == \texttt{a}[\texttt{p}]] <- \texttt{a}[\texttt{q}] \quad \# \text{update connectivity array}
    }
}
Example: Slower Quick Find...

QuickFind2 <- function( N = 100, M = 100)
{
  a <- seq(1, N)  # initialize array

  for (dummy in seq(1,M))  # for each connection
  {
    p <- sample(N, 1)  # sample random objects
    q <- sample(N, 1)

    if (a[p] == a[q])  # check if connected
      next

    set <- a[p]  # update connectivity array
    for (i in 1:N)
      if (a[i] == set)
        a[i] = a[q]

  }
}
Example: Slower Quick Find...

```r
> bench <- \textbf{function}(f, N = 100, M = 100)
{\texttt{
  \texttt{cat(" N = ", N, ", M = ", M, "\n")

  start <- Sys.time()
  f(N, M)
  stop <- Sys.time()
  difftime(stop, start)
}
> bench(QuickFind, 4000, 4000)
  \texttt{N = 4000 , M = 4000}
\texttt{Time difference of 2 secs}
> bench(QuickFind2, 4000, 4000)
  \texttt{N = 4000 , M = 4000}
\texttt{Time difference of 1.066667 mins}
```
Example: Weighted Quick Union

WeightedQuickUnion <- function( N = 100, M = 100)
{
  a <- seq(1, N)  # initialize arrays
  weight <- rep(1, N)

  for (dummy in seq(1,M)) # for each connection
  {
    p <- sample(N, 1)  # sample random objects
    q <- sample(N, 1)

    i = a[p]; while (a[i] != i) i <- a[i]  # FIND
    j = a[q]; while (a[j] != j) j <- a[j]

    if (i == j) next

    if (weight[i] < weight[j])  # UNION
      { a[i] = j; weight[j] <- weight[j] + weight[i]; }
    else
      { a[j] = i; weight[i] <- weight[i] + weight[j]; }
  }
}
Learning More About R

- Excellent documentation is available at www.r-project.org

- "An Introduction to R" by Venables and Smith in the Documentation Section

- Another good book to browse is "Data Analysis and Graphics in R" by Maindonald and Braun