Harvey Mudd College
Computer Science 60
Spring 2007

Assignment 3
Applications Using Functional Programming
Due. 11:59 p.m., Sunday, 10 February 2008

All problems in this assignment are to be done in a purely functional style using the Scheme language. Submit a single file named hw03.scm containing all solutions using the web page http://www.cs.hmc.edu/~cs60grad/submissions/. As always, document your functions clearly. **Be sure to spell each function name exactly as given**, otherwise the automatic grading script might not give you credit for the function.

1. [10 of 20 points] Construct a *tail-recursive* (as defined in the lecture) version of a Fibonacci number generator **`my-fib`** that, with argument N, outputs the Nth element in the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, ... in which each term is the sum of the previous two. Test sample:

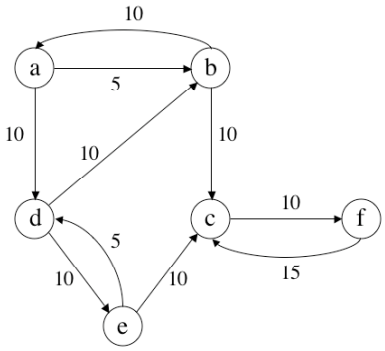
```
(test (map my-fib '(0 1 2 3 4 5  6  7  8  9 10))
                  '(1 1 2 3 5 8 13 21 34 55 89))
```

2. [10 of 20 points] Construct a *tail-recursive* version of the function **`find-max`**, which expects a non-empty list of reals as an argument. It returns a list consisting of the largest number in the argument list, followed by the index at which that number occurs. If the largest number occurs more than once, the lowest index is what is returned. Test samples:

```
(test (find-max '(1.5))                    '(1.5 0))
(test (find-max '(1.5 2.7 -3 4.9 1.9))     '(4.9 3))
(test (find-max '(1.5 2.7 -3 4.9 1.9 4.9)) '(4.9 3))
```

3. [60 points] Implement function `shortest-path-lengths` that computes the shortest paths from a specified source node to each node in a directed graph having non-negative distances associated with each arc. For this problem, a graph is a list of nodes. Each node is a list beginning with a *name*, which is a symbol. The rest of the list is an a*ssociation list* that specifies the nodes to which this node is directly connected, together with a *direct distance* for that connection. An example graph is shown below:

```
(define graph1 '(
  (a (b 5)   (d 10))
  (b (a 10)  (c 10))
  (c (f 10))
  (d (b 10)  (e 10))
  (e (c 10)  (d 5))
  (f (c 15))
  ))
```



In the graph shown, the set of nodes is {a, b, c, d, e, f}. There is a direct connection from a to b with a distance of 5, from a to d with a distance 10, etc. The direct distances are not necessarily symmetric, i.e. the distance from a to b is 5, while the direct distance from b to a is 10, as in this example. In any case, the direct distance is not necessarily the shortest directed distance from the source to other nodes. There may be paths that go through several nodes that are shorter than the given distance.

The function shortest-path-lengths has two arguments, a source node and a graph, such as the one above. It returns a list of the shortest distance of *each* node in the graph to the source, together with the predecessor of that node in the shortest path. Moreover, the nodes are listed in increasing order from the source. (If there is a tie, we use the order within tied nodes according to string<? applied to the names after converting the symbols to strings.) Here are some test examples for graph1 above. The shortest path from a to f has length 25, for example, and the predecessor to f on the shortest path is c.

```
(test (shortest-path-lengths 'a graph1)
    '((a 0 _) (b 5 a) (d 10 a) (c 15 b) (e 20 d) (f 25 c)))

(test (shortest-path-lengths 'd graph1)
    '((d 0 _) (b 10 d) (e 10 d) (a 20 b) (c 20 b) (f 30 c)))

(test (shortest-path-lengths 'f graph1)
    '((f 0 _) (c 15 f) (a +inf.0 _) (b +inf.0 _)
      (d +inf.0 _) (e +inf.0 _)))
```

A path length of +inf.0 (meaning numeric *infinity* in our version of Scheme) indicates that the corresponding node is *not reachable* from the source. Built-in arithmetic functions such as > and + do the correct thing with respect to +inf.0.

The symbol '_ should be used to indicate that there is no predecessor in a shortest path, e.g. in the case of the source node or an unreachable node. Please use that symbol so that your answers will agree with ours.

4. [20 points] Create function shortest-paths-to-all that lists the paths to each reachable node from the start node, in reverse. The result is a list of lists, where each inner list is a list of nodes, beginning with a reachable node and ending with the start node. If a node is not reachable, it does not appear anywhere in the result.

```
(test (shortest-paths-to-all 'a graph1)
      '((a) (b a) (d a) (c b a) (e d a) (f c b a)))

(test (shortest-paths-to-all 'd graph1)
      '((d) (b d) (e d) (a b d) (c b d) (f c b d)))

(test (shortest-paths-to-all 'f graph1) '((f) (c f)))
```

For example, in the shortest path from a to f, the node before f is c, the node before c is b, and the node before b is a. Thus (f c b a) appears as an element of the result. In the third example, there are several nodes unreachable from f, which thus do not appear.

Suggestions for solving this problem:

a. Dijkstra's algorithm, as explained in the lecture and in c-e below, is highly recommended. Diagrams of Dijkstra's algorithm in action are included below.

b. The recursive implementation of Dijkstra's algorithm will "retire" one node at a time, beginning with the source node. When a node is retired, its minimum distance from the source is known. (This can be justified, but might not be totally obvious.)

c. Do not create modified node structures, as we had occasion to do in the cycle-testing example in lecture. Instead, create a new list of "wrapped" nodes that also contains the shortest distance found so far, as well as the previous node on that path.

d. On each major step, the active node with the minimum distance is retired and the best estimates of minimum distances to the remaining active nodes are updated for posterity, to reflect the possibility of a path from the newly-retired node to the remaining active nodes. (Updating requires changing the distance component of the active nodes, but not the nodes themselves.) This is the essence of Dijkstra's algorithm. (When a lesser distance is installed, the predecessor is updated too.)

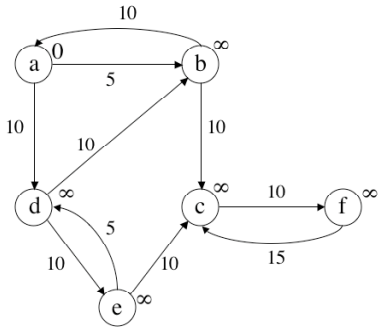
e. Maintain two lists of wrapped nodes: The list of "retired" nodes and a list of "active" nodes. All nodes start on the active list. Reachable nodes move to the retired list, never to return.

f. One way to find the minimum of a list is to sort the items and choose the first. This is not optimal, but it will be acceptable for this problem. To use the sort function built into Scheme (which is advised), you will need to specify an appropriate comparison function for wrapped nodes.

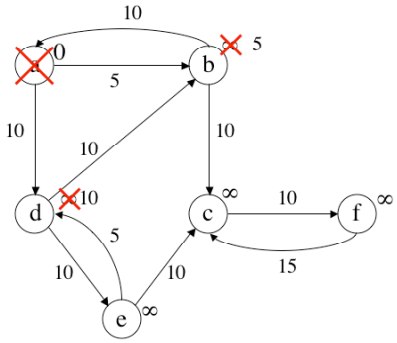
Extra credit [20 points]: For additional enlightenment, note that substantial list-sharing is possible among the elements of the resulting list of `shortest-paths-to-all`. For example, `(c b a)` is replicated as the rest of `(f c b a)`. After you have the basic solution, you might try to construct a solution that realizes this sharing. If you can, and can document your approach, that is worth 20 points extra credit. (Attempting to optimize this way in your initial solution is generally not advised, as it may get in the way of basic correctness concerns.)

Dijkstra's algorithm on the given graph example:
X'ed nodes are retired. X'ed numbers are distance updates.

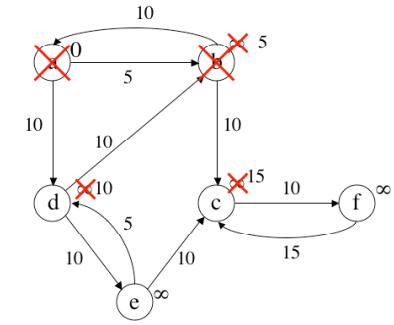
1.
Original node distance assignments (source is node a)



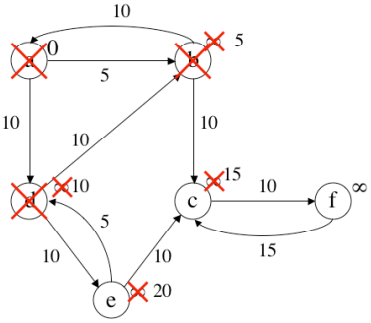
2.
Retiring minimum distance active node: a



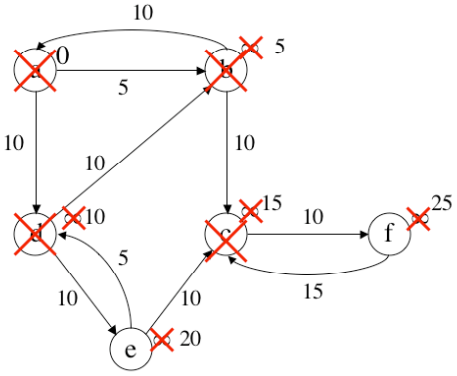
3.
Retiring minimum distance active node: b



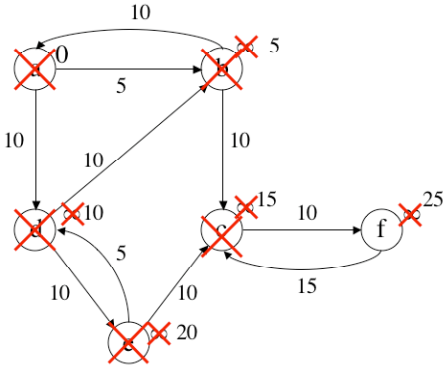
4.
Retiring minimum distance active node: d



5.
Retiring minimum distance active node: c



6.
Retiring minimum distance active node: e

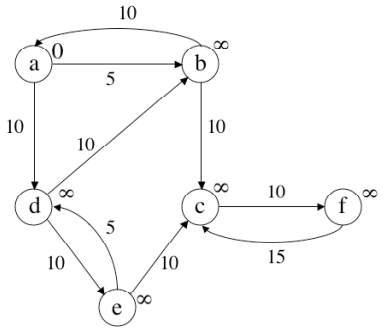


The final step is not shown, for reasons of brevity.

Dijkstra's algorithm showing predecessor references (dashed arrows):

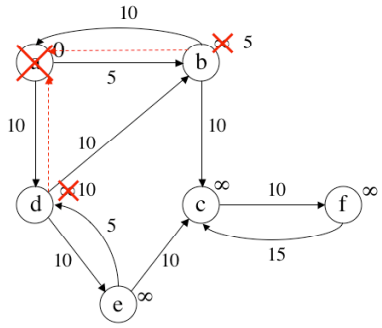
1.

Original node distance assignments (source is node a)



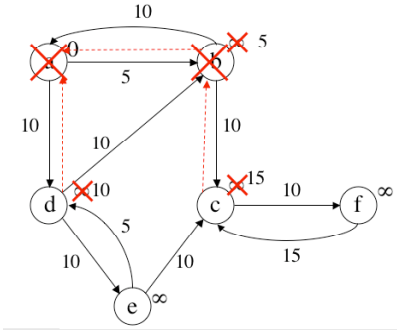
2.

Retiring minimum distance active node: a



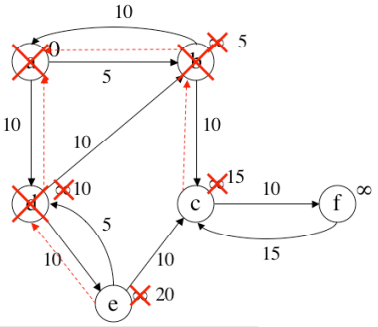
3.

Retiring minimum distance active node: b



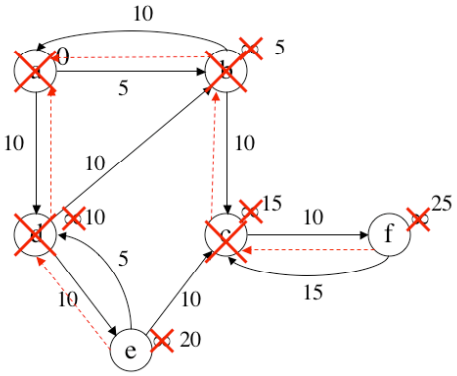
4.

Retiring minimum distance active node: d



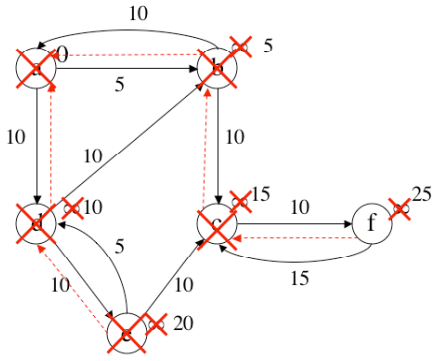
5.

Retiring minimum distance active node: c



6.

Retiring minimum distance active node: e



The final step is not shown, for reasons of brevity.