Home
Syllabus
Lecture Notes
Assignments
Project 4

David Galles
Computer Science
Univerisity of San Francisco

Programming Assignment 4: Dijkstra, Binomial Heaps, Hash Tables, and More!

For your final project, you will implement Dijksra's algorithm for finding the shortest path between a source vertex and all other vertices in the graph. This project will combine Graphs, Dijkstra's Algorithm, Binomial Queues, Hash Tables and Lists (whew!).

Nodes with Labels

For this assignment, each vertex in the graph will be labeled with an arbitrary string. How can we run Dijkstra's algorithm when the vertices are not integers? First, we will assign an integer to each vertex string (the easiest way to do this is to assign the first string we see to 0, the second string we see to 1, etc). We can then store the vertex strings in an array, where the index of the array is the number that we assign to the vertex string stored at that index. For instance, if the datafile contains the vertex names "San Francisco", "Los Angeles", and "New York", we can assign 0 to San Francisco (storing the string “San Francisco” at index 0 of our array), assign 1 to Los Angles (storing the string “Los Angles” at the index 1 of our array) and we can assign 2 to New York (storing the string “New York” at index 2 of our array). Now, given any number n, it is easy to find the appropriate string assigned to that number (by looking at index n in our array).

How can we get an index given a string? One method would be to search through our entire array looking for the appropriate string. A better solution (and the solution that you are required to use for this assignment) is to create a hash table to store the vertex string / vertex number combinations. You can use the vertex string as the key to enter the vertex number into the hash table (so each hash table entry will have a vertex number data value, and a string key value). You can use any hashig strategy that you wish, though you might find that open hashing is slightly easier to implement. Note that you are not allowed to use any built-in hash table functionality in Java, you will need to write the hash table yourself!

Implementing the Graph

One you have created your array of vertex names and your hash table to look up vertex numbers, you are ready to build the graph. Your graph need not contain any information about which vertex number is assigned to which node string – that information can be kept entirely in you node string array and your hash table. You will find the hash table quite useful when creating the graph, however!

Dijkstra's Algorithm

Once you have your graph, you are ready to run Dijkstra's algorithm, using Binomial Queues. Start with the specified initial vertex. Create an instance of the Dijksra table. All "cost" entries in the Dijkstra table (other than the intial vertex) should be infinity (you can use Integer.MAX_VALUE) and all "path" entries should be -1. Add all of the vertices to the BinomialQueue (with priority of Integer.MAX_VALUE, except for the initial vertex, with priority 0). While the priority queue is not empty, you will need to remove the element with the smallest cost, then update the costs of all of its neighbors – also updating the costs in the priority queue! Thus your priority queue will need some way of finding elements in the queue quickly. To do this, you should maintain an array of pointers into your Binomial Queue. Once you have created the table, you will need to print out the cost of the shortest path from the intial vertex to all other vertices, as well as the path itself. See the section on output format to get the formatting correct.

Binomial Queue

The Binomial Queue that you use for Dijktra's algorithm will need to implement the following methods:

- void insert(int elem, int priority) Inserts the integer elem in to the queue, using the given priority
- booelan empty() Returns true if the Queue is empty.
- int removeSmallest() Removes the element with the smallest priority from the queue, and returns it (the element, not the priority)
- void decreaseKey(int elem. int new_priority) Reduce the priority of the element elem in the priority queue to new_priority, rearranging the queue as necessary. To implement this method efficiently, you will need to keep track of where each element is in the queue. Your best bet is probably an array of pointers into the queue.
- void printQueue() Prints out the priority queue, showing the value and priority of each node in the tree, and shape of the queue (like we have done for all human-readible tress this semester). See the sample output for the included Test.java file for more information. This method is mostly useful for debugging (though we will also use it to test if your BinomialQueue is correct!)

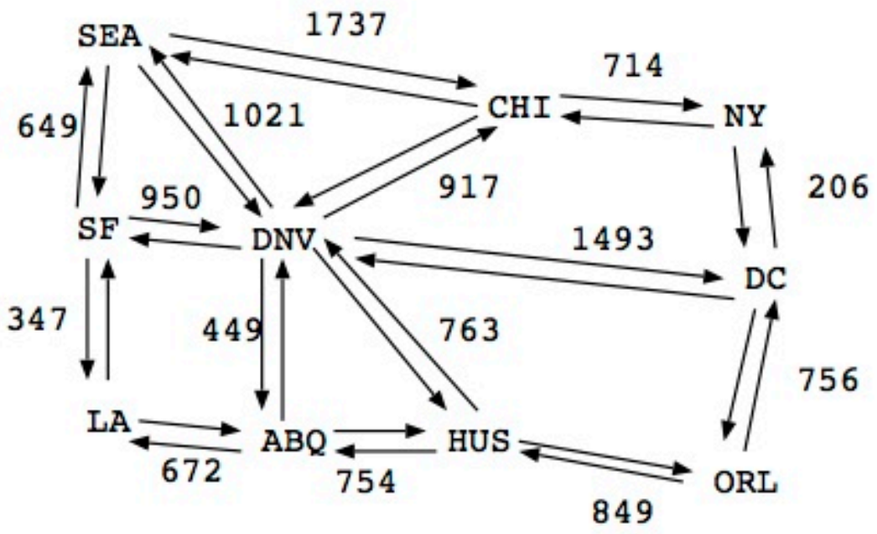
Quick Overview of Data Structures

So, while your algorithm is running, you will have the following data structures:

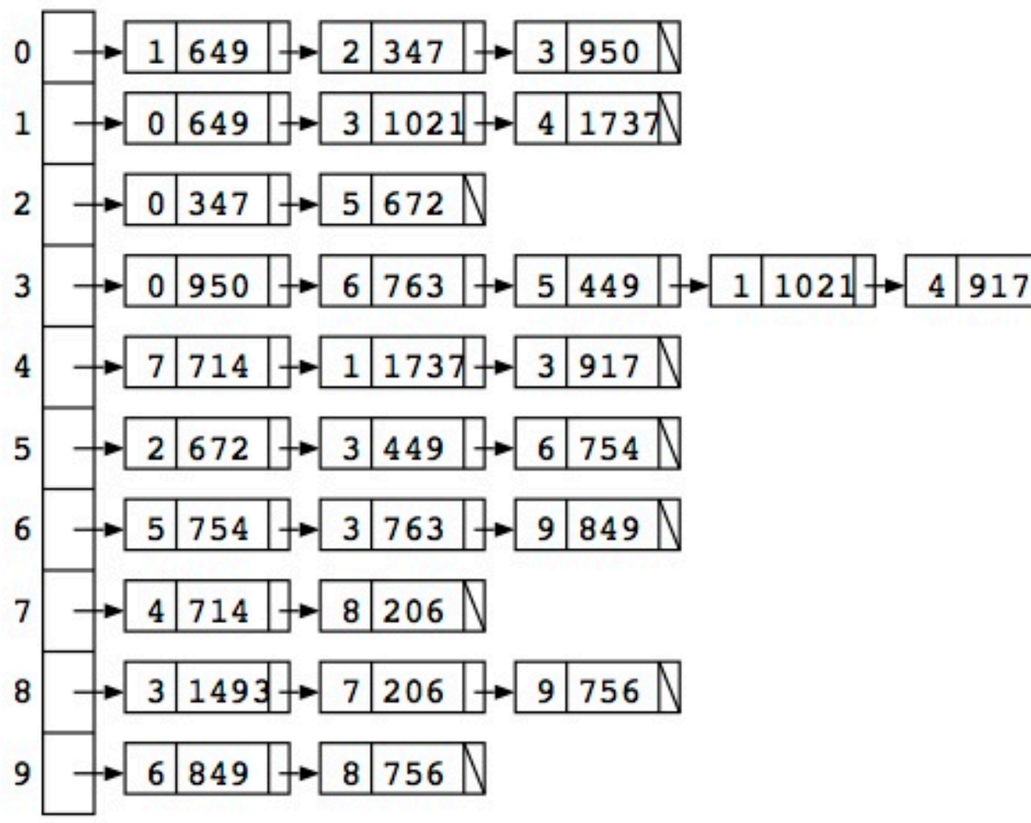
- Graph (Adjacency List)
- List used to associate vertex numbers with vertex names
- Hash table used to associate vertex names with vertex numbers
- Dijkstra Table
- Binomial Queue (including an array of pointers into the queue for easy updating)

Thus, just before you start running Dijkstra's algorithm, your data structures might look something like:

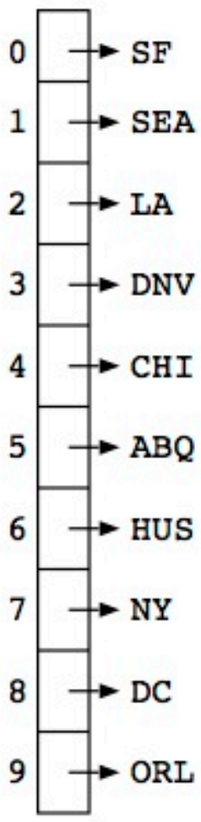
Logical Graph



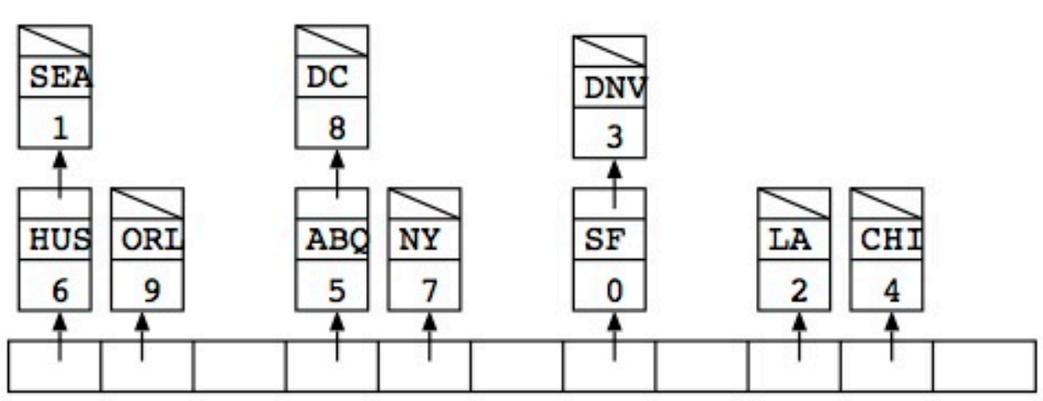
Adjacency List



Name List

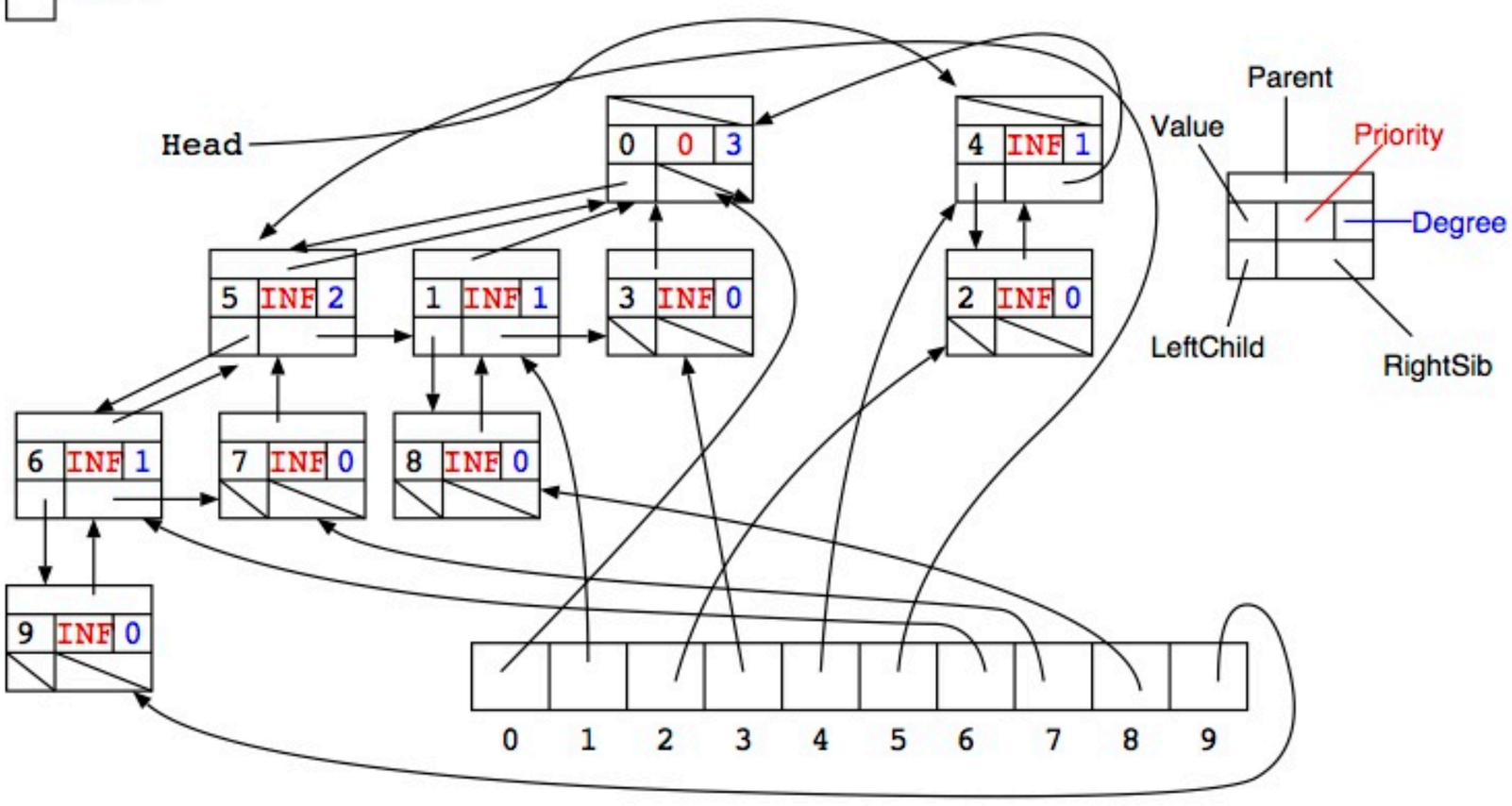


Hash Table



Dijkstra Ta

0	0	-1
1	INF	-1
2	INF	-1
3	INF	-1
4	INF	-1
5	INF	-1
6	INF	-1
7	INF	-1
8	INF	-1
9	INF	-1



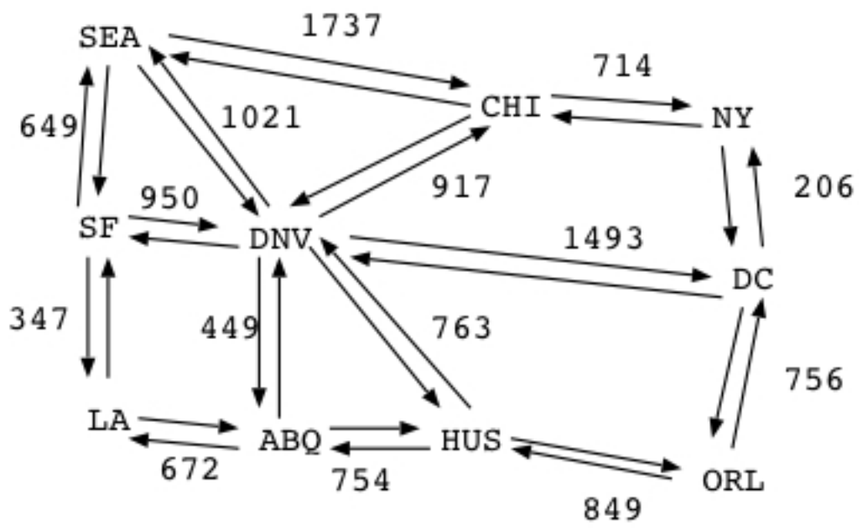
Binomial Queue

Input File Format:

The data file will have the following format:

- A list of the vertex labels in the graph, one per line. The first vertex is the starting vertex
- A line containing the single character "."
- A list of edges in the following form:
  - First endpoint of the edge, on a single line
  - Second endpoint of the edge, on a single line
  - Cost of the edge (integer) on a single line

Thus, the graph:



with the intial node SF, could be represented by the input file:

```
SF
LA
SEA
DNV
ABQ
HUS
CHI
NY
ORL
DC
.
SEA
SF
649
SF
LA
347
LA
SF
347
SF
SEA
649
SF
DNV
950
DNV
SF
950
LA
ABQ
672
ABQ
LA
672
DNV
ABQ
449
ABQ
DNV
ABQ
DNV
449
SEA
DNV
1021
DNV
SEA
1021
SEA
CHI
1737
CHI
SEA
1737
DNV
CHI
917
CHI
DNV
917
DNV
HUS
763
HUS
DNV
763
HUS
ORL
849
ORL
```

```
DC
756
ORL
HUS
849
DC
ORL
756
NY
DC
206
DC
NY
206
CHI
NY
714
NY
CHI
714
DC
DNV
1493
DNV
DC
1493
ABQ
HUS
754
HUS
ABQ
754
```

Your program should get the name of the input file from a command line parameter. Thus, you should run your program as:

```
% java Dijkstra inputfile
```

Ouput File Format:

Your program output to standard out, using the following format:

- First, print out the graph that was read in, in an adjacency list format (using vertex names, not vertex numbers)
- Next, print out a separating line
- Next, print out for each vertex:
  - Vertex name
  - Cost of the shortest path to that vertex
  - Actual shortest path to that vertex

Thus, a legal output for the input file above would be:

```
Original Graph
SF: SEA 649, DNV 950, LA 347
LA: SF 347, ABQ 672
SEA: SF 649, DNV 1021, CHI 1737
DNV: SF 950, SEA 1021, ABQ 449, HUS 763, DC 1493, CHI 917
ABQ: LA 672, DNV 449, HUS 754
HUS: ABQ 754, DNV 763, ORL 849
CHI: SEA 1737, DNV 917, NY 714
NY: CHI 714, DC 206
ORL: HUS 849, DC 756
DC: DNV 1493, NY 206, ORL 756

Shortest Paths

SF  0: SF
LA  347: SF LA
SEA  649: SF SEA
DNV  950: SF DNV
ABQ  1019: SF LA ABQ
HUS  1713: SF DNV HUS
CHI  1867: SF DNV CHI
NY  2581: SF DNV CHI NY
ORL  2562: SF DNV HUS ORL
DC 2443: SF DVN DC
```

Project Submission

Submit all files required to run your project to:

```
https://www.cs.usfca.edu/svn/<username>/cs245/Project4/
```

Your project is required to have the following classes: (names must match!)

- Dijkstra
  - No required methods, but must have a main that runs the entire project
- BinomialQueue

    - void insert(int elem, int priority)
    - int removeSmallest()
    - void decreaseKey(int elem. int new_priority)
    - void printQueue()
- HashTable
    - void insert(String key, int value)
    - int find(String key)
    - void delete(String key)

Random Details

- Do not use any Java Collection classes in this project You may use:
    - Basic arrays
    - Your own classes
    - ... pretty much nothing else

  Therefore, you may **not** use *any* of the following:
    - ArrayList
    - LinkedList
    - HashMap
    - HashSet
    - PriorityQueue
    - ... anything that implements the java.util.Collection interface

  When in doubt, ask if it is allowed. In general, assume that no Java libraries are OK for this project. Yes, when you are coding "for real" you will be using libraries, but the point of this class is for you to understand how those libraries work, not just how to use them!
- This is a largish project, which will require several classes. Start early!
- Write pieces of your project (like the hash table and Binomial Queue) and test them separately before combining them into the final project.

Provided Files

- testio.java Sample code that that reads in a graph file and prints it out, to give you an example of dealing with file I/O in java. Feel free to modify it for your own use, or just use it for reference. (Or you could ignore it completely, if you already understand Java file I/O) Ask if you have any questions!
- Test.java Some code to test PriorityQueue and HashTable clases. Sample output: testOutput.