homework1.pdf homework6.tex hw3-1a.fig.bak hw5ans.log infpost01.cpp

University of Nevada, Las Vegas Computer Science 477/677 Fall 2023

Assignment 6: Due Saturday April 6, 2024, 11:59 PM

Name:____

You are permitted to work in groups, get help from others, read books, and use the internet. Turn the assignment in to Canvas, following the instructions given to you by Sabrina Wallace.

1. **Storing an Abstract Array as a 1-Dimensional Array:**

   Read this internet page:

   `https://www.prepbytes.com/blog/arrays/base-address-of-a-two-dimensional-array/`

   The discussion on that page presumes that the computer's random access memory (RAM) is a 1-dimensional array of *cells*, indexed by integers starting at 0, and that each cell consist of 4 bits of memory. That number could be different. Each addressable location could consist of 4, or 16, or 32, or whatever, bits. We write `RAM[i]` to be the i^{th} cell (actually, the $(i+1)^{\text{st}}$ because of the zero address.) An item to be stored in RAM could need any number of cells; that number is called *size* in that article. An array declared by your program would (normally) be stored as a contiguous block of cells starting at a *base adress*, which is chosen by the compiler. For example, if you declare the array int A[100], and each integer requires four cells, and the compiler chooses 1024 to be the base adress, 400 cells are allocated to store A. A[0] is stored at RAM[1024] ... RAM[1027], while A[i] has base address 1024+4i and is stored in RAM[1024+4i] ... RAM[1024+4i+3]. The number 4i is called the *offset* of A[i].

   Basic Rule: If a number of items are stored, the offset of any item is equal to the number of prdecessors of that item times the size of each item. To get the address in RAM, add the offset to the base address.

   Example: An array int A[5][3] is stored in row-major order, the base address is 2048, and once again, an integer uses 4 cells. The elements of A are stored in this order: A[0][0], A[0][1], A[0][2], A[1][0], ... A[4][1], A[4][2]. On the other hand, if they are in column-major order, their order in RAM is A[0][0], A[1][0], ... A[3][2], A[4][2].

   (a) In the example, if int A[5][3] is stored in RAM with offset 2048, what is the RAM address of A[3][1] if the storage is row-major? What if it is column-major?

   If row major, that entry has 3*3 + 1 = 10 predecessors, each of which takes 4 cells, and the base address of the array is 2048. Thus A[3][1] has address 2048 + 4*10 = 2088.

   If column major, A[3][1] has 5*1 + 4 = 9 predecessors, each of which takes 4 cells, and the base address of the array is 2048. Thus A[3][1] has address 2048 + 4*9 = 2084.

   (b) If X[10][25][30] is stored with base address 8192, and each entry of the array requires 8 cells, what is the RAM address of X[8][11][15] if X is stored in row-major order? what if in column-major order?

   If row major, that entry has 8*25*30 + 11*30 + 15 = 6345 predecessors, each of which takes 8 cells, and the base address of the array is 8192. Thus X[8][11][15] has address 8192 + 8*6345 = 58952.

If column major, that entry has 10*25*15 + 10*11 + 8 = 3868 predecessors, each of which takes 8 cells, and the base address of the array is 8192. Thus X[8][11][15] has address 8192 + 8*3868 = 38136.

2. **Sparse Arrays:**

   Crawley's Department Store hired a CS graduate to set up a system which could access any customer's complete record, containing all the information that Crawley's wants to save for that customer, by entering her[1] social security number.

   The graduate (who slept late that day in CS477) started by defining a structured type called `record` and then declaring an array `record customer[1000000000]` because a social security number has nine digits and there are one billion strings of nine digits. But the number of customers that Crawley's has ever had is no more than twenty thousand.

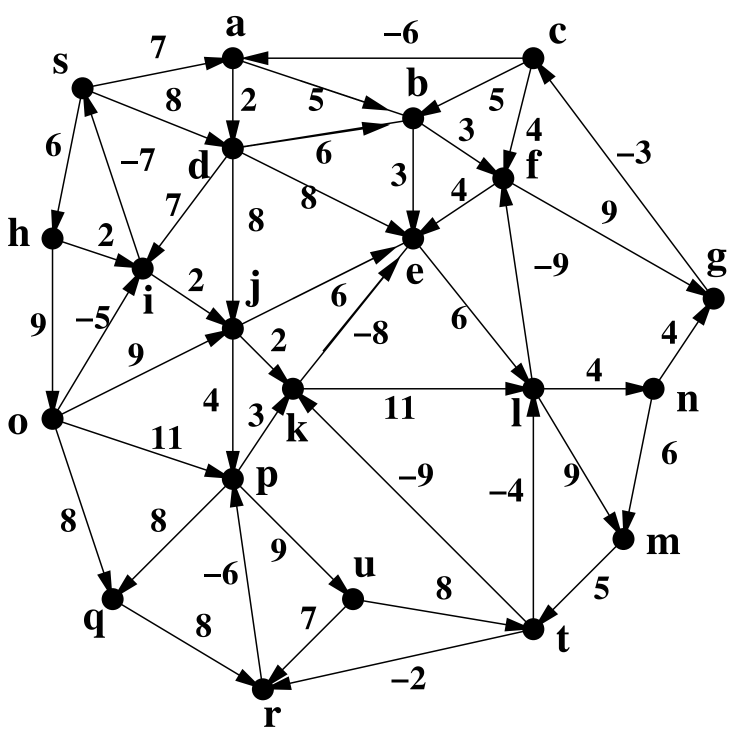
   Instead, he should have stored the records in a *sparse array.* If Amanda Jones was a customer and had SSN x, then customer[x] will return her record, but if there is no customer with SSN y, then customer[y] will return a default value, such as zero, or perhaps the message "not found."

   The array `customer` is then a *sparse array.* There are a number of ways to implement sparse arrays, but my favorite is as a search structure of memos. The structure in indexed by SSN. A memo is an ordered pair (s,A[s]). where the memo consists of the social security number of an actual customer, followed by the record of that customer. The command fetch[s] returns the record of the customer whose SSN is s, otherwise a default. The **store** command either overwrites an existing record or creates a new memo.

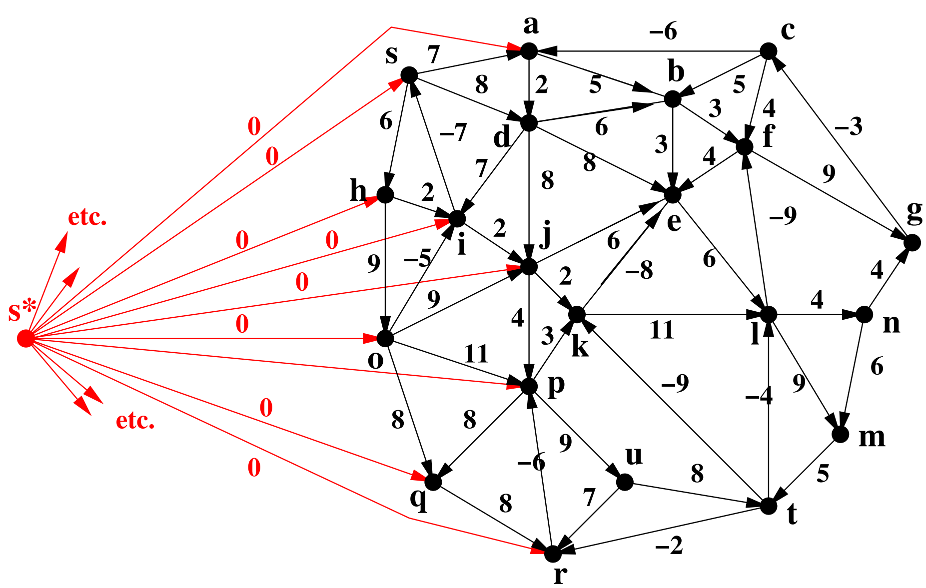
   Memos are stored as a sparse array, which we can implement as a search structure, as described in Problem 2, except that, if there is no entry for a given index, that entry must be computed and then stored. I recommend that an ordinary (not balanced) binary search tree not be used, as it does not perform well in examples I have worked.

[1] By convention, the customer is always "she."

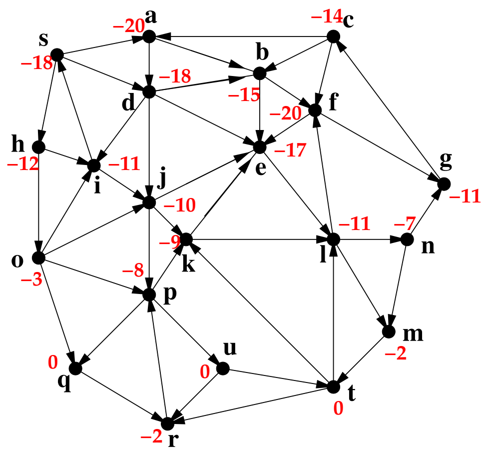
3. The figure below shows a weighted directed graph. Partially work Johnson's algorithm on that graph.

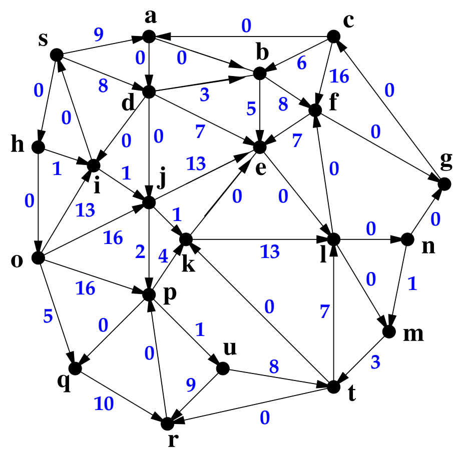


We first augment the graph by adding one more vertex, s, togeher with an arc of weight 0 from s to each other vertex, as shown below.



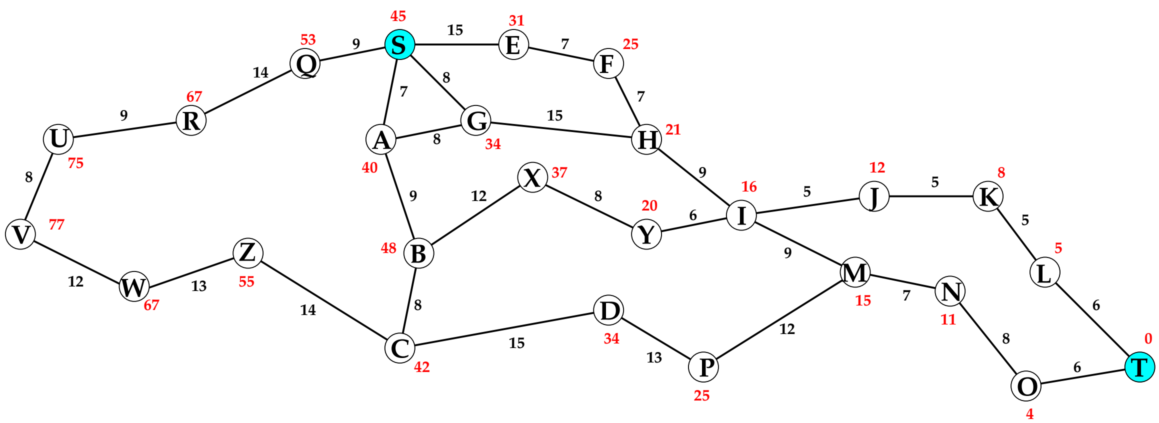
We then use the Bellman-Ford algorithm to compute the weight of the shortest path from s to each other vertex. These weights are shwon below in the figure to the left, while the right-hand figure shows the adjusted ar weights, which non-negative.





4. **A* Algorithm**

Walk throuth the A^* algorithm for the following weighted graph, finding the least cost path from S to T. The edge weights are in black and the heuristics are in red. The heuristics are both admissible and consistent. Your answer should label each fully processed vertex with both f and g values. Not all vertices will be processed.



Values of f are shown in magenta, values of g in green, and back pointers are shown in red.