University of Nevada, Las Vegas Computer Science 477/677 Fall 2024

Answers to Assignment 6: Due Saturday November 9, 2024

Overview of hashing from the internet: "In the context of hash tables, 'open hashing' refers to a collision resolution strategy where colliding elements are stored in separate linked lists outside the main hash table array, while 'closed hashing' (also called open addressing) stores colliding elements within the array itself by probing for an available slot when a collision occurs; essentially, open hashing uses external data structures to handle collisions, while closed hashing uses the same array to store all elements, even if they collide."

1. Solve these recurrences, expressing the answers using Θ notation.

   These are all solved using the Akra–Brazzi method.

   (a) $F(n) = F(n/2) + 2F(n/4) + n$

   $a_1 = 1$, $b_1 = \frac{1}{2}$, $a_2 = 2$, $b_2 = \frac{1}{4}$, $c = 1$. $\sum_{i=1} 2a_i b_i^c = 1$, hence $F(n) = \Theta(n \log n)$.

   (b) $F(n) = 2F(2n/3) + F(n/3) + n$

   $a_1 = 2$, $b_1 = \frac{2}{3}$, $a_2 = 1$, $b_2 = \frac{1}{3}$, $c = 1$. $\sum_{i=1} 2a_i b_i^c = \frac{5}{3} > 1$. $\sum_{i=1} 2a_i b_i^2 = 1$, hence $d = 2$. $F(n) = \Theta(n^2)$.

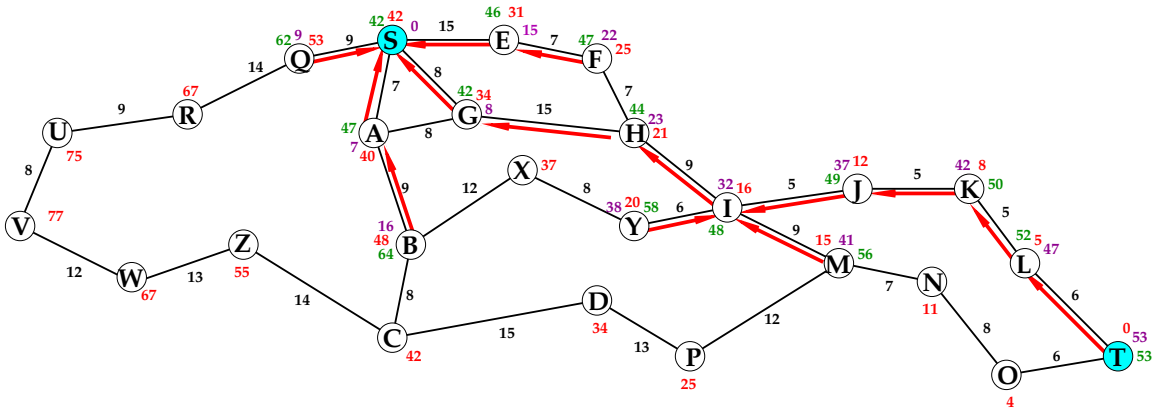
   (c) $F(n) = 3F(n/2) + 3F(n/4) + n^2$

   $a_1 = 3$, $b_1 = \frac{1}{2}$, $a_2 = 3$, $b_2 = \frac{1}{4}$, $c = 2$. $\sum_{i=1} 2a_i b_i^c = \frac{15}{16} < 1$, hence $F(n) = \Theta(n^2)$.

   (d) $T(n) = T(7n/10) + T(n/5) + n$

   $a_1 = 1$, $b_1 = \frac{7}{10}$, $a_2 = 1$, $b_2 = \frac{1}{5}$, $c = 1$. $\sum_{i=1} 2a_i b_i^c = \frac{9}{10} < 1$, hence $F(n) = \Theta(n)$.

2. 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. In the figure below, each fully processed vertex is labeled with both f and g values, and backpointers are shown. Values of g are shown in magenta (although it's hard to see that color) and values of f in green.



3. The following recursive C++ program computes a function `george(n)` for a given integer n.

```
int george(int n)
 {
  if(n <= 1} return 1;
  else
   return george(n/2) + george(n/3) + george(n/6) + n;
 }
```

You only want to compute george(n) for some n.

(a) What is the asymptotic complexity, in terms of n, of the value of george(n)?

Let george(n) = $G(n)$. We have the recurrence $G(n) = G(n/2) + G(n/3) + G(n/6) + n$. Using the Akra–Brazzi method, our solution is $G(n) = \Theta(n \log n)$.

(b) What is the asymptotic complexity, in terms of n, of the time required to find george(n) using the recursive program given above? (Hint: the answers to (a) and (b) are not the same.)

Let $T(n)$ be the time required for the program to compute george(n). We have the currence

$T(n) = T(n/2) + T(n/3) + T(n/6) + 1$ since computation of n requires $O(1)$ time. Solving that recurrence, we have $T(n) = \Theta(n)$.

(c) What is the asymptotic complexity, in terms of n, of the time required to compute george(n) using dynamic programming? That means, computing george(i) for all i from 0 to n in order.

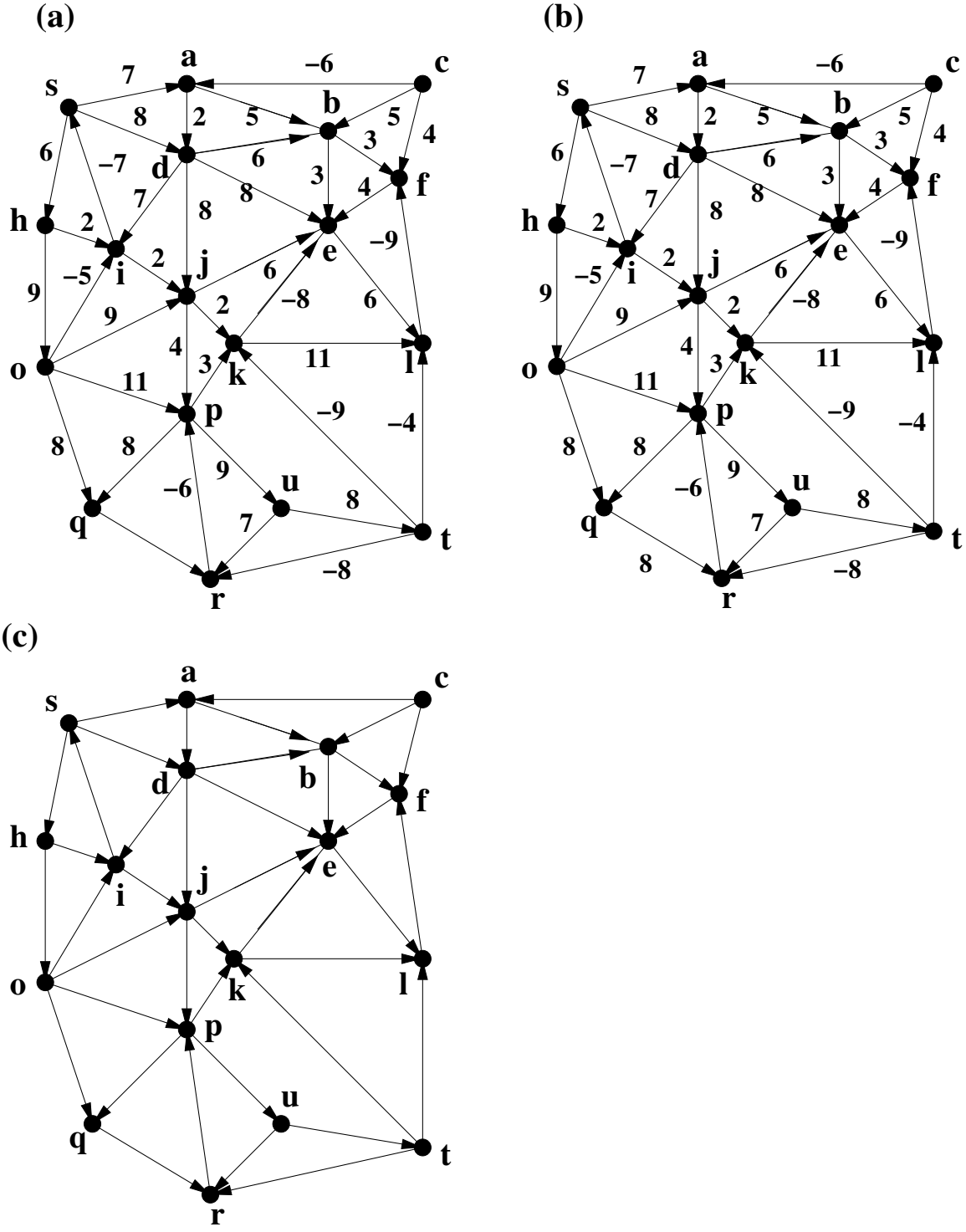
It takes $\Theta(1)$ time to compute george(i) for each i. The total time complexity is thus $\Theta(n)$.

(d) What is the asymptotic complexity, in terms of n, of the time required to compute george(n) using memoization? (Hint: how many intermediate values will be stored?)

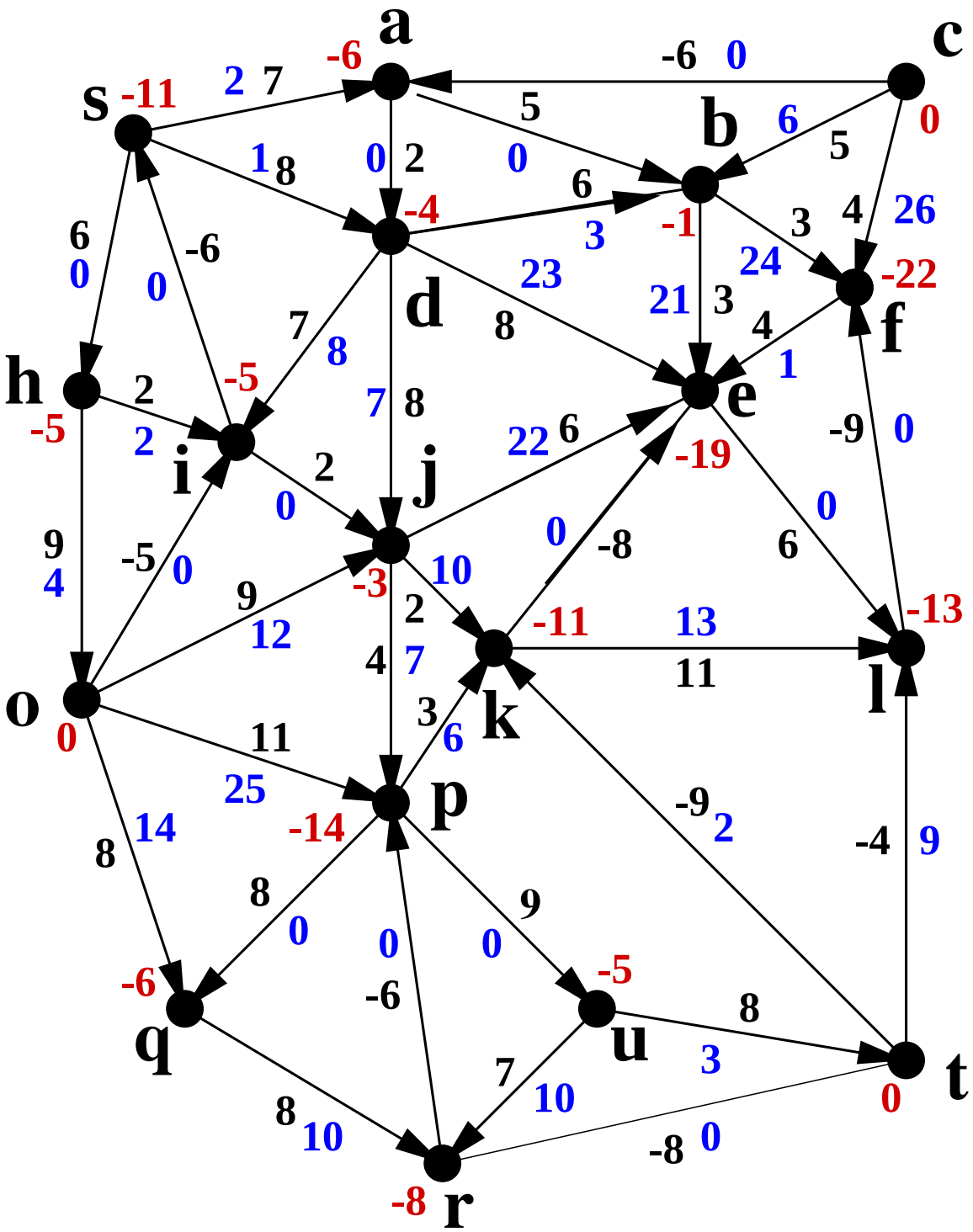
This is the hardest of these four questions. The answer is $\Theta(\log^2 n)$. Here is the explanation. To compute george(i) for any i, we must first have computed george($i/2$), george($i/3$), and george($i/6$), Thus, to compute george(n), we need to compute and store the values of george$\left(\frac{n}{2^i 3^j}\right)$ for all pairs (i,j) such that $2^i 3^j <= n$. How do we count those? Taking the logarithm of both sides, we obtain the inequality $i \log 2 + j \log 3 \leq \log n$ Both $\log 2$ and $\log 3$ are constant, and thus asymptotically we have $i + j \leq \log n$. Letting $m = \log n$, we have $i + j \leq m$. Thus, the number of such pairs (i, j) is (asymptotically) $\sum_{i=0}^{m}(m - i) \approx \frac{m^2}{2} = \Theta(m^2) = \Theta(\log^2 n)$.

4. Figure (a) below illustrates a weighted directed graph. Figures (b) and (c) are copies. In Figure (c), show the adusted arc weights needed to work Johnson's algorithm. Use (b) for your intermediate work, showing the non-positive values at the vertices. To avoid clutter, do not draw the zero-weight arcs from s to the other vertices.

   Do not finish Johnson's algorithm.



The figure below shows the original weighted digraph with the adjusted non-negative weights. The black numerals are the original weights. The red numeral at each vertex is the minimum cost of a directed path from s to that vertex. The zero-weight arcs from s to each other vertex are not shown. The blue numerals are the adjusted weights of the arcs.



5. Evaluate

(a) $\dfrac{\log_5 8}{\log_5 4} = \dfrac{3}{2}$

(b) $\dfrac{\log_3 2}{\log_3 \left(\frac{1}{2}\right)} = -1$

(c) $\log_5 9 \cdot \log_3 5 = 2$

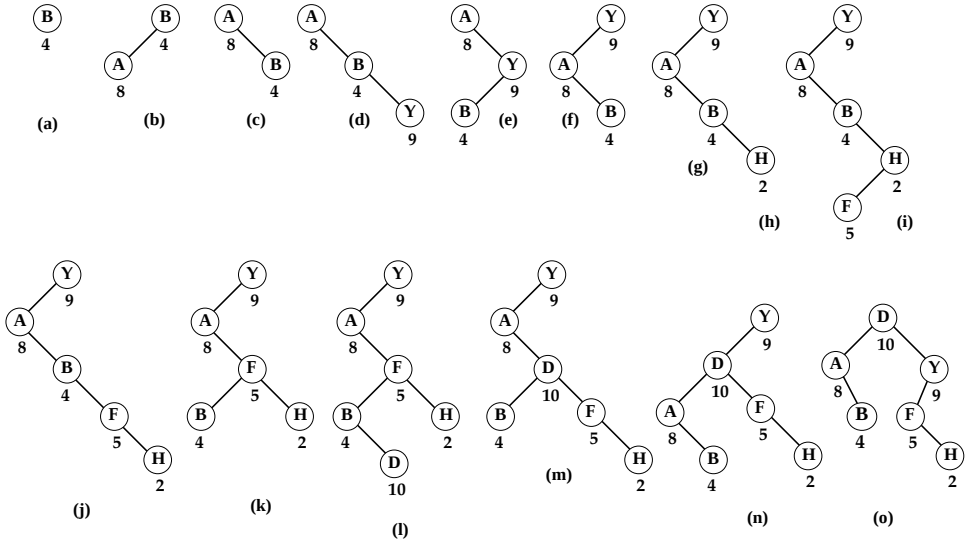
(d) $\log_2 \sqrt{8} = \dfrac{3}{2}$

(e) $2^{\log_2 9 - \log_2 3} = 3$

(f) $4^{\log_2 5} = 25$

6. Insert the following items into a treap, in the order given. The priority of each item is given. Show all the insertions and rotations. Your treap should use max-heap order. Your first seven steps are shown below. In total, there should be 15 (or so) figures.

B	4
A	8
Y	9
H	2
F	5
D	10



Hashing

Here is a link to notes from a class on hashing at Carnegie Mellon University.
`https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf`
And here are notes from a class on hashing at the Univerity of Washington.
`https://courses.cs.washington.edu/courses/cse326/06su/lectures/lecture11.pdf`

7. We are given a list of $N = 6$ data, each of which is a 3-letter name.

Bob
Ann
Sue
Ted
Van
Liz

We wish to store the data in a hash table. As explained on the CMU page, each name is converted into a binary string by replacing each letter with the binary numeral of the order of that letter in the Roman alphabet. These numerals are padded with zeros so that all have length 5, as shown in the table below. For simplicity, our code is case-insensitive, so "Bob" is written "000100111100010"

a	0	0	0	0	1		n	0	1	1	1	0
b	0	0	0	1	0		o	0	1	1	1	1
c	0	0	0	1	1		p	1	0	0	0	0
d	0	0	1	0	0		q	1	0	0	0	1
e	0	0	1	0	1		r	1	0	0	1	0
f	0	0	1	1	0		s	1	0	0	1	1
g	0	0	1	1	1		t	1	0	1	0	0
h	0	1	0	0	0		u	1	0	1	0	1
i	0	1	0	0	1		v	1	0	1	1	0
j	0	1	0	1	0		w	1	0	1	1	1
k	0	1	0	1	1		x	1	1	0	0	0
l	0	1	1	0	0		y	1	1	0	0	1
m	0	1	1	0	1		z	1	1	0	1	0

The encoding of each datum is a string of 15 bits. The spaces are just for readability.

```
bob = 00010 01111 00010
ann = 00001 01110 01110
sue = 10011 10101 00101
ted = 10100 00101 00100
van = 10110 00001 01110
liz = 01100 01001 11010
```

We let the size of our hash table be $M = 8 = 2^3$, and thus the hash value of each datum is a string of three bits, which represents an integer in the range $0 \ldots M - 1$.

Therefore, our univeral hash function u must be a 3×15 matrix of bits, which I chose using a random number generator.

1	1	0	1	0	0	0	0	0	1	1	0	1	0	1
0	1	0	0	1	0	0	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	0	1	0	1	1	1	0

The computation of the hash values of the data uses mod 2 matrix multiplication. The only constants are 0 and 1, and 1+1 = 0.

Let u be the 3×15 universal hash function, let *dat* be the 15 matrix containing the encodings of the data, and h the 3×6 matrix giving the hash values of the data. Then $u \times dat = h$, using mod 2 matrix multiplication.

For example $101101 \times 110101 = 1 * 1 + 0 * 1 + 1 * 0 + 1 * 1 + 0 * 0 + 1 * 1 \ \% \ 2 = 1$

Here is the matrix multiplication used. To make it easier to check the matrix multiplication by hand, I have left spaces.

```
                          001110
                          000001
                          000111
                          101010
                          011000

                          001000
11010 00001 10101         110001         011101
01001 00101 01011    X    111100    =    100010
11111 01101 01110         110000         100110
                          101111

                          000001
                          010011
                          011110
                          110011
                          001000
```

h(Bob) $= 011 = 3$
h(Ann) $= 100 = 4$
h(Sue) $= 100 = 4$
h(Ted) $= 101 = 5$
h(Van) $= 011 = 3$
h(Liz) $= 100 = 4$

The probability that a given pair of data will collide is $\frac{1}{M} = \frac{1}{8}$.

The number of pairs of data is $\binom{N}{2} = \frac{N(N-1)}{2} = 15$ hence the expected number of collisions is $\frac{N(N-1)}{2M} = \frac{15}{8}$. The number of actual collisions is 4, larger than expected, but not by much.