University of Nevada, Las Vegas Computer Science 477/677 Fall 2022

Answers to Assignment 7: Due Friday December 9, 2022

This version: Mon Dec 12 08:08:35 PST 2022

1. In each blank, write Θ if correct, otherwise write O or Ω, whichever is correct.

   (a) $n^2 = O(n^3)$

   (b) $\log(n^2) = \Theta(\log(n^3))$

   (c) $\log(n!) = \Theta(n \log n)$

   (d) $\log_2 n = \Theta(\log_4 n)$

   (e) $n^{0.000000000001} = \Omega(\log n)$

2. True or False. Write "T" or "F." If the answer is not known to science at this time, write "O" for "Open."

   (a) **T** There is a mathematical statement which is true, yet cannot be proven.

   (b) **T** The subproblems of a dynamic program form a directed acyclic graph.

   (c) **T** A hash function should appear to be random, but cannot actually be random.

   (d) **F** Open hashing uses open addressing.

   (e) **F** No good programmer would ever implement a search structure as an unordered list.

   (f) **F** Computers are so fast nowadays that there is no longer any point to analyzing the time complexity of a program.

   (g) **T** A complete graph of order 4 is planar.

   (h) **T** Heapsort can be considered to be a sophisticated implementation of selection sort.

   (i) **T** Binary tree sort (also called "treesort") can be considered to be a sophisticated implementation of insertion sort.

3. Solve each recurrence, expressing the answer as an asymptotic function of n. Use O, Ω, or Θ, whichever is most appropriate.

   (a) $F(n) \leq 2F(n/2) + n^2$

   $F(n) = O(n^2)$ by the master theorem, since $2(1/2)^2 < 1$.

   (b) $F(n) \geq 3F(n/9) + 1$

   $F(n) = \Omega(\sqrt{n})$ by the master theorem, since $3(1/9)^{1/2} = 1$.

   (c) $F(n) = F(3n/5) + 4F(2n/5) + n^2$

   $F(n) = \Theta(n^2 \log n)$ by the generalized master theorem, since $(3/5)^2 + 4(2/5)^2 = 1$.

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

   $F(n) = \Theta(n)$ by the generalized master theorem, since $(1/5) + (7/10) < 1$.

(e) $F(n) = F\left(\frac{n}{2}\right) + n$

$F(n) = \Theta(n)$ by the master theorem, since $\frac{1}{2} < 1$.

(f) $F(n) = 2F\left(\frac{n}{2}\right) + n$

$F(n) = \Theta(n \log n)$ by the master theorem, since $2(\frac{1}{2}) = 1$.

(g) $F(n) = 4F\left(\frac{n}{2}\right) + n$

$F(n) = \Theta(n^2)$ by the master theorem, since $4(\frac{1}{2})^2 = 1$.

(h) $F(n) \geq F\left(\frac{n}{2}\right) + 2F\left(\frac{n}{4}\right) + n$

$F(n) = \Omega(n \log n)$ by the generalized master theorem, since $\frac{1}{2} + 2(\frac{1}{4}) = 1$

Error Correction

(i) $F(n) = F(n-1) + \sqrt{n}$

$F(n) - F(n-1) = \sqrt{n}$ Hence $F(n) = \Theta\left(n^{3/2}\right)$, by the anti-derivative method.

(j) $F(n) = 2F(n/2) + n$ $F(n) = \Theta(n \log n)$, same as problem 3f.

(k) $G(n) = G(n/2) + 1$

$G(n) = \Theta(\log n)$ by the master theorem, since $(1/2)^0 = 1$.

(l) $K(n) \leq 4K(n/2) + n^2$

$K(n) = O(n^2 \log n)$ by the master theorem, since $4(1/2)^2 = 1$

(m) $J(n) \geq J(3n/5) + J(4n/5) + 1$

$J(n) = \Omega(n^2)$ by the generalized master theorem, since $(3/5)^2 + (4/5)^2 = 1$

(n) $L(n) = L(n - \sqrt{n}) + n$

$L(n) = \Theta(n^{3/2})$ by the anti-derivative method, since

$$\frac{L(n) - L(n - \sqrt{n})}{\sqrt{n}} = n^{1/2}$$

(o) $H(n) \leq H(\sqrt{n}) + 1$

$H(n) = O(\log \log n)$ using substitution.

Let $m = \log n$ and let $F(m) = H(2^m)$ for any m, hence $H(n) = F(\log n) = F(m)$ for any n. Then $H(\sqrt{n}) = F(\log \sqrt{n}) = F\left(\frac{\log n}{2}\right) = F(m/2)$.

Substituting into the original recurrence, we have $F(m) \leq F(m/2) + 1$, hence, by the master theorem, $F(m) = O(\log m)$. Substituting again, we have $H(n) = O(\log \log n)$.

4. Give the asymptotic time complexity, in terms of n, for each of these code fragments. Students are only required to give the answers, but I have supplied an explanation for each problem.

We use summation or integration in most of these examples. We first work a sample problem:

```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
    cout << "Hello world!" << endl;
```

During the i^{th} iteration of the outer loop, the inner loop iterates approximately $\log i$ times. The time complexity can thus be approximated by the summation:

$$\sum_{i=1}^{n} \log i$$

Since the natural logarithm is a constant multiple of the base 2 logarithm, that summation is asymptotically equivalent to

$$\int_{x=1}^{n} \ln x \, dx = [x \ln x - x]_{x=1}^{x=n} = (n \ln n - n) - (1 \ln 1 - 1) = n \ln n - n + 1 = \Theta(n \log n)$$

(i)
```
for(int i = 0; i < n; i++)
  for(int j = n; j > i; j = j/2)
    cout << "Hello world!" << endl;
```

The answer is $\Theta(n)$. We substitute `k = log j` in the inner loop, obtaining

```
for(int i = 0; i < n; i++)
  for(int k = log n; k > log i; k--)
    cout << "Hello world!" << endl;
```

The inner loop iterates $\log n - \log i$ times during the i^{th} iteration of the outer loop. We use integration, substituting the real variable x for i. Recall $\ln x = \Theta(\log x)$. The asymptotic time complexity of the code is

$$\int_{x=1}^{n} (\ln n - \ln x) dx = [x \ln n - x \ln x + x]_1^n = n \ln n - n \ln n + n = \Theta(n)$$

(ii)
```
for(int i = 0; i < n; i++)
  for(int j = i; j > 0; j = j/2)
    cout << "Hello world!" << endl;
```

The answer is $\Theta(n \log n)$. We substitute `k = log j` in the inner loop, obtaining

```
for(int i = 0; i < n; i++)
  for(int k = log i; k > 0; k--)
   cout << "Hello world!" << endl;
```

The inner loop iterates approximately $\log i$ times for the i^{th} iteration of the outer loop. We use integration, substituting the real variable x for i. Recall $\ln x = \Theta(\log x)$. The asymptotic time complexity of the code is

$$\int_{x=1}^{n} (\ln x) dx = [x \ln x - x]_1^n = n \ln n - n + 1 = \Theta(n \log n)$$

(iii)
```
for(int i = 2; i < n; i=i*i)
  cout << "Hello world!" << endl;
```

The answer is $\Theta(\log \log n)$. Substitute `j = log i` and `m = log n`. Since $\log(i^2) = 2 \log i$ we have

```
for(int j = 1; j < m; j = 2j)
  cout << "Hello world!" << endl;
```

Substitute `k = log j` and `p = log m`. Since $\log(2j) = \log j + 1$ we have

```
for(int k = 0; k < p; k++)
  cout << "Hello world!" << endl;
```

The loop iterates p times. The asymptotic complexity is $\Theta(p) = \Theta(\log m) = \Theta(\log \log n)$.

(iv)
```
for(int i = 1; i*i < n; i++)
  cout << "Hello world!" << endl;
```

The answer is $\Theta(\sqrt{n})$. Let $m = \sqrt{n}$. Taking the square root of both sides of the boundary condition of the for loop, we obtain

```
for(int i = 1; i < m; i++)
  cout << "Hello world!" << endl;
```

The asymptotic time complexity is $\Theta(m) = \Theta(\sqrt{n})$.

(v)
```
for(i = 0; i < n; i = i+1);
  cout << "Hello world!" << endl;
```

The answer is $\Theta(n)$.

(vi)
```
for(int i = 1; i < n; i = i+i)
  cout << "Hello world" << endl;
```

The answer is $\Theta(\log n)$. Let `j = log i` and `m = log n`. $\log(i + i) = \log i + 1 = j + 1$. We have

```
for(int j = 0; j < m; j = j+1)
  cout << "Hello world" << endl;
```

The asymptotic time complexity is $\Theta(m) = \Theta(\log n)$.

(vii)
```
for(int i = 2; i < n; i = i*i)
  cout << "Hello world" << endl;
```

This is a duplicate of Problem (iii).

(viii)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
    cout << "Hello world" << endl;
```

The answer is $\Theta(n \log n)$.

(ix)
```
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
    cout << "Hello world" << endl;
```

The answer is $\Theta(n)$.

We now explain the solutions to the last two problems. Let $k = \log j$ and $m = \log n$. Substituting, Problem (viii) becomes

```
for(int i = 1; i < n; i++)
  for(int k = 0; k < log i; k++)
```

while Problem (ix) becomes

```
for(int i = 1; i < n; i++)
  for(int k = log i; k < m; k++)
```

During the i^{th} iteration of the outer loop, The inner loop of Problem (viii) iterates $\log i$ times, while The inner loop of Problem (ix) iterates $\log n - \log i$ times. The time complexity of (viii) is thus approximately $\int_1^n \ln x dx = [x \ln x - x]_1^n = \Theta(n \log n)$, while The time complexity of (ix) approximately $\int_1^n (\ln n - \ln x) dx = [x \ln n - x \ln x + x]_1^n = -\ln n + n \ln n - n \ln n + n - 1 = \Theta(n)$.

(x)
```
for(int i = 1; i*i < n; i++)
  cout << "Hello world" << endl;
```
This is a duplicate of Problem (iv).

(xi)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
    cout << "Hello world" << endl;
```
This is a duplicate of Problem (viii).

(xii)
```
for(int i = 0; i < n; i++)
  for(int j = 0; j*j < n; j++)
    cout << "Hello world" << endl;
```
The two loops are independent. The other loop iterates n times and the inner loop iterates $\sqrt{n}$ time for each iteration of the outer loop. Thus the answer is $\Theta(n\sqrt{n})$.

(xiii)
```
for(int i = n; i > 1; i = i/2)
  for(int j = 0; j < i; j++)
    cout << "Hello world" << endl;
```
The answer is $\Theta(n)$. First notice that each line of output occurs when $0 \le j < i$ and $i = n/2^k$ for some k. We reverse the nesting of the loops, maintaining those bounds on i and j.

```
for(int j = 0; j < n; j++)
  for(int i = n; i > j; i = i/2)
    cout << "Hello world" << endl;
```

We can start with $j = 1$ without changing the asymptotic complexity. During the j^{th} iteration of the outer loop, the inner loop iterates $\log n - \log j$ times. Approximate by integration, where y is a real variable approximating j. We have $\int_1^n (\ln n - \ln y) dy = [y \ln n - y \ln y + y]_{y=1}^{y=n} = \Theta(n)$.

(xiv)
```
for(int i = 1; i < n; i++)
  for(int j = 2; j < i; j=j*j)
    cout << "Hello world" << endl;
```
The answer is $\Theta(n \log \log n)$. During the i^{th} iteration of the outer loop, the inner loop iterates approximately $\log \log i$ times. We could use integration and get $\int_1^n \ln \ln x \, dx$, but the anti-derivative of $\ln \ln x$ is not covered in first year calculus, so we will use summation.

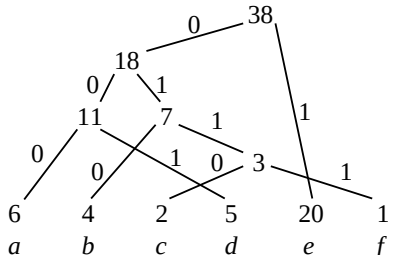
We first note that $\log \log(n/2) = \log(\log n - 1) = \Theta(\log \log n)$. The time complexity of our code is approximately $\sum_{i=1}^{n} \log \log i$ which we split into two parts:

$$\sum_{i=1}^{n} \log \log i = \sum_{i=1}^{n/2-1} \log \log i + \sum_{i=n/2}^{n} \log \log i$$

The second "half" is obviously larger than the first, and each term of the second half is greater than or equal to $\log \log(n/2)$; thus the total is at least $(n/2) \log \log(n/2) = \Theta(n \log \log n)$.

5. Find an optimal prefix code for the alphabet $\{a, b, c, d, e, f\}$ where the frequencies are given in the following array.

a	6	000
b	4	010
c	2	0110
d	5	001
e	20	1
f	1	0111



6. Fill in the blanks.

   In problems (i) and (ii), let n be the number of vertices, m the number of arcs, and p the maximum number of arcs in the shortest path between any two vertices.

   (i) The asymptotic complexity of the Floyd/Warshall algorithm is $\Theta(n^3)$.

   (ii) The asymptotic complexity of Dijkstra's algorithm algorithm is $O(m \log n)$

   (iii) A **perfect** hash function fills the hash table exactly with no collisions.

   (iv) **Huffman's** algorithm finds a binary code so that the code for one symbol is never a prefix of the code for another symbol.

   (v) **Huffman's** and **Kruskal's** are greedy algorithms that we've studied this semester.

   (vi) **Mergesort, quicksort and binary search** are divide-and-conquer algorithms that we've studied this semester.

   (vii) In **separate chaining** there can be any number of items at a given index of the hash table.

   (viii) The asymptotic expected time to find the median item in an unordered array of size n, using a randomized selection algorithm, is $\Theta(n)$.

   (ix) If $h(x)$ is already occupied for some data item x, a **probe sequence** is used to find an unoccupied position in the hash table.

   (x) If a directed acyclic graph has n vertices, it must have n strong components.

   (xi) If a planar graph has 10 edges, it must have at least **6** vertices.

   (xii) If G is a weighted graph, then it is impossible to solve the all pairs shortest path problem for G if G has a **negative cycle**

   (xiii) Fill in the blank with one letter. If all arc weights are equal, then Dijkstra's algorithm visits the vertices in same order as **B**FS.

   (xiv) If a planar graph has 7 edges, it must have at least **5** vertices. (You must give the best possible answer, exactly. No partial credit.)

   (xv) The height of a binary tree with 17 nodes is at least **4**. (You must give the best possible answer, exactly. No partial credit.)

   (xvi) The following is pseudo-code for what algorithm?

   bubblesort.

```
int x[n];
obtain values of x;
for(int i = n-1; i > 0; i--)
 for(int j = 0; j < i; j++)
  if(x[j] > x[j+1])
   swap(x[j],x[j+1]);
```

(xvii) **Dijkstra's** algorithm does not allow the weight of any arc to be negative.

(xviii) The asymptotic time complexity of Johnson's algorithm on a weighted directed graph of n vertices and m arcs is $O(nm \log n)$. (Your answer should use O notation.)

(xix) The time complexity of every comparison-based sorting algorithm is $\Omega(n \log n)$. (Your answer should use Ω notation.)

(xx) The postfix expression $zw + x \sim y - *$ is equivalent to the infix expression $(z + w) * (-x - y)$

(xxi) The items stored in a priority queue (that includes stacks, queues, and heaps) represent **unfulfilled obligations**.

(xxii) A directed graph has a topological order if and only if it is **acyclic**.

(xxiii) **stack, queue** and **heap** are three examples of priority queues.

(xxiv) The operators of the ADT **stack** are *pop* and *push*.

(xxv) The operators of the ADT **array** are *fetch* and *store*.

(xxvi) In order to solve a shortest path problem on a weighted directed graph, there must be no **negative cycle**.

(xxvii) If a planar graph G has n vertices, where n is at least 3, then G can have no more than $3n - 6$ edges. (Exact formula, please.)

7. Compute the Levenshtein distance between *abcdafg* and *agbccdfc*. Show the matrix.

		a	g	b	c	c	d	f	c
	0	1	2	3	4	5	6	7	8
a	1	0	1	2	3	4	5	6	7
b	2	1	1	1	2	3	4	5	6
c	3	2	2	2	1	2	3	4	5
d	4	3	3	3	2	3	2	3	4
a	5	4	4	4	3	3	3	3	4
f	6	5	5	5	4	4	4	3	4
g	7	6	5	6	5	5	5	4	4

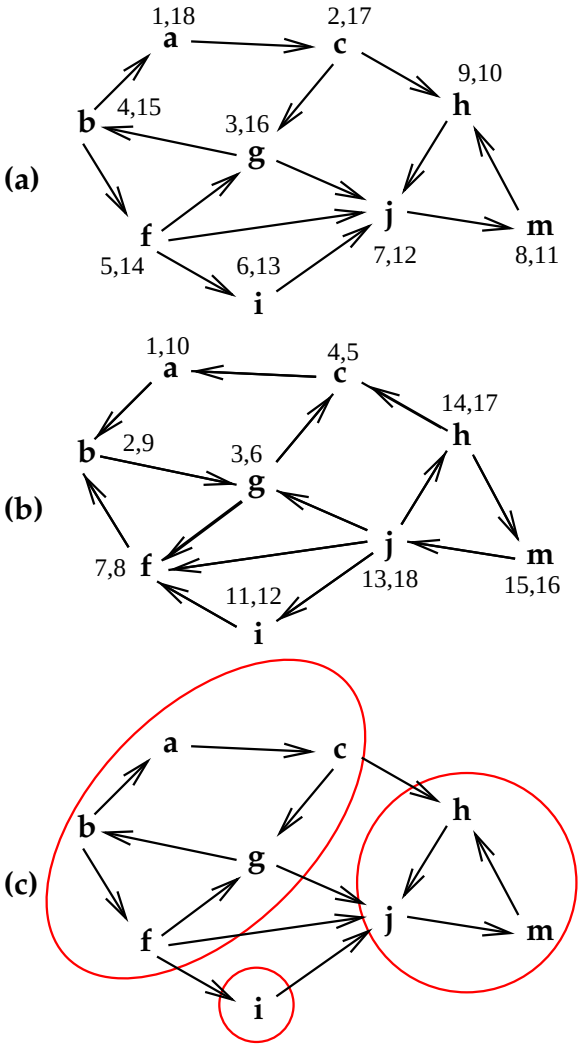
The Levenstein distance is 4.

8. You need to store Pascal's triangle in row-major order into a 1-dimensional array P whose indices start at 0. The triangle is infinite, but you will only store $\binom{n}{k}$ for $n < N$. Write a function I such that $P[I(n,k)] = \binom{n}{k}$ for $0 \le k \le n < N$. For example, $I(3,2) = 8$.

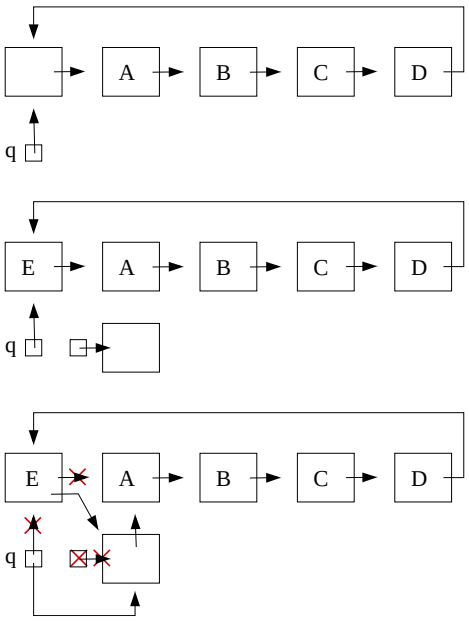
```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

```
int I(int n, int k)
 {
  // the position of n choose k in the linear array
  assert(k >= 0 and n >= k and n < N);
  int indx = // fill in formula here
   n*(n+1)/2 + k;
  return indx;
 }
```

9. Use the DFS method to find the strong components of the digraph shown below as (a). Use the other figures to show your steps.



10. Sketch a circular linked list with dummy node which implements a queue. The queue has four items. From front to rear, these are A, B, C, D, and show the insertion of E into the queue. Show the steps. Don't erase deleted objects; instead, simply cross them out.



11. In class, we implemented a minheap as an almost complete binary tree, implemented as an array.

    (a) Suppose the minheap is initialized as shown in the first line of the array shown below. Show the evolution of the structure when deletemin is executed.

A	C	F	D	Q	H	L	R	Z
Z	C	F	D	Q	H	L	R	
C	Z	F	D	Q	H	L	R	
C	D	F	Z	Q	H	L	R	
C	D	F	R	Q	H	L	Z	

    (b) Starting from the final configuration above, show the evolution of the structure when B is inserted.

C	D	F	R	G	H	L	Z	B
C	D	F	B	G	H	L	Z	R
C	B	F	D	G	H	L	Z	R
B	C	F	D	G	H	L	Z	R

12. What is the loop invariant of the loop in the following function?

    $x * n = y * m + z$

```
float product(float x, int n)
 {
  // assert(n >= 0);
  float z = 0.0;
  float y = x;
  int m = n;
  while(m > 0)
   {
    if(m%2) z = z+y;
    m = m/2;
    y = y+y;
   }
  return z;
 }
```

13. The usual recurrence for Fibonacci numbers is:
$F[1] = F[2] = 1$
$F[n] = F[n-1]$ for $n > 2$

However, there is another recurrence:
$F[1] = F[2] = 1$
$F[n] = F\left[\frac{n-1}{2}\right] * F\left[\frac{n}{2}\right] + F\left[\frac{n+1}{2}\right] * F\left[\frac{n+2}{2}\right]$ for $n > 2$
where integer division is truncated as in C++.

Using that recurrence, Describe a $\Theta(\log n)$-time memoization algorithm which reads a value of n and computes $F[n]$, but computes only $O(\log n)$ intermediate values.

A search structure consisting of *memos*, each of which is an ordered pair of the form $(i, F(i))$. If the the program requests the value $F(i)$, the structure first searches for a memo whose first term is i. If such a memo is not found, the value $F[i]$ is computed using the function and then the memo $(i, F(i))$ is stored. The value $F(i)$ is then returned. The actual function is then:

```
int F(int n)
     {
      if (n <= 2) return 1;
      else return F((n-1)/2)+F(n/2)+F((n+1)/2)+F((n+2)/2);
     }
```

The number of memos that are stored during the computation of $F(n)$ is $\Theta(\log n)$.

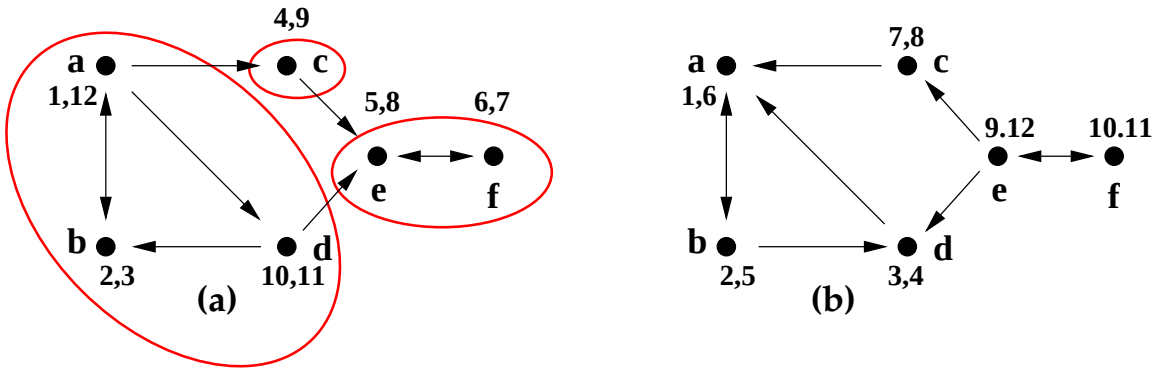
14. Write pseudocode for the Bellman-Ford algorithm. Be sure to include the shortcut that ends the program when the final values have been found.

We assume there are n vertices, named 1 through n, and the source vertex is 1. We assume there are m arcs. The j^{th} arc is the ordered pair $(x\{j\}, y[j])$, which has weight $w[j]$, which we assume is an integer. The output is the array V where $V[i]$ is the length of the shortest path from 1 to i, as well as the array $\{back[i]\}$.

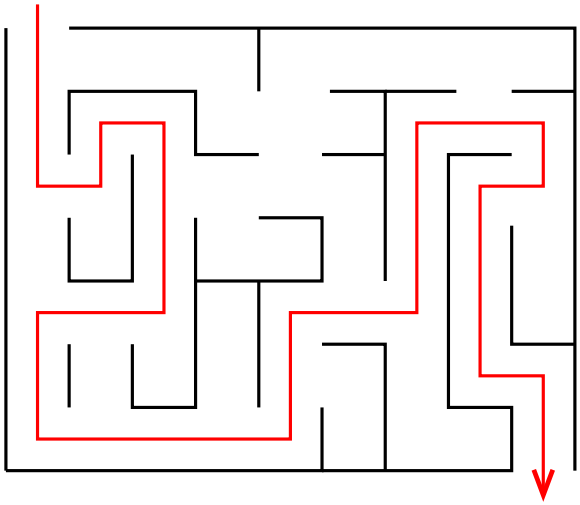
```
for(int i = 1; i < n; i++) V[i] = infinity;
V[0] = 0;
bool changed = true;
while(changed)
  {
   changed = false;
   for(int j = 1; j <= m; j++)
    {
     int temp = V[x[j]] + w[j];
     if(temp < V[y[j]])
      {
       V[y[j]] = temp;
       back[y[j]] = x[j];
       changed = true;
      }
    }
  }
```

In this code, we assume there is no negative cycle. Otherwise, the while loop will never halt.

15. Find the strong components of graph (a) below, using DFS search. Use (b) for your work. Circle the strong components.



16. The figure below shows an example maze. The black lines are walls. You need to find the shortest path, avoiding the walls, from the entrance at the upper left and the exit at the lower right. The red path shows one such path, although it is not the shortest. Describe a program to find the shortest path from the entrance of such a maze, not necessarily this one, to the exit. You do not need to write pseudocode. Your answer should contain the word, "graph," and should state which search method and which data structure(s) you need to use.



We reduce the problem to the single pair shortest path problem in an undirected graph where all edge weights are 1. The graph has 63 vertices, arranged in a 7×9 grid. Each vertex is one square of the figure, and there is an edge between adjacent squares if there is no "wall" separting them. The source vertex is the upper left square and the target vertex is the lower right.
Breadth first search, using a queue, will identify the shortest path, which is not the red path shown in the figure.

17. Walk through polyphase mergesort, where the input file is as given below.

VJANLDQMFSP

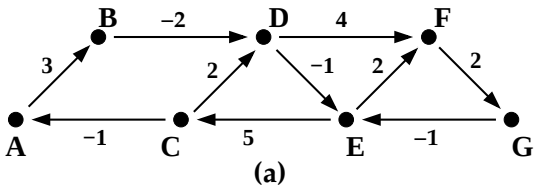
VANDQFS
JLMP

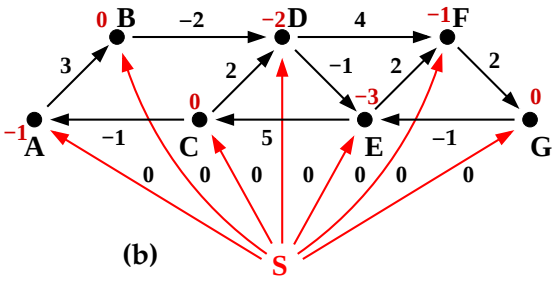
JLMPVDQ
ANFS

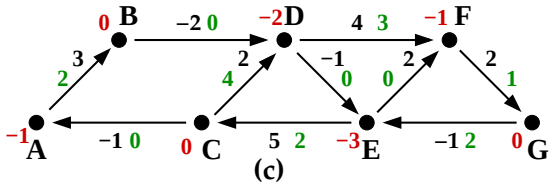
AJLMNPV
DFQS

ADFJLMNPQSV

18. Suppose we wish to solve the all-pairs shortest path problem for the weighted digraph shown in (a) below, using Johnson's algorithm. First solve the single source minpath problem using a fictitious source to give each vertex a non-positive number. In (b), label each vertex with that number. We then compute an adusted non-negative weight for each edge. In (c), label each edge with the correct adjusted weight. Do not finish Johnson's algorithm.







19.
```
float power(float x, int n)
{
 assert(n >= 0);
 float y = x;
 int m = n;
 float z = 1.0;
 while(m > 0)
  {
   if(m%2) z = z*y;
   y = y*y;
   m = m/2;
  }
 return z;
}
```

(a) What does this function do?

It outputs x^n.

(b) What is the loop invariant of the while loop?

$x^n = y^m * z$