91.503 Algorithms **Name: SOLUTION**
Spring 2002 Midterm Examination
Open Book (CLRS), open notes, time period: 3 hours.
Record answers in exam book. Turn in exam book(s) and examination question sheet.

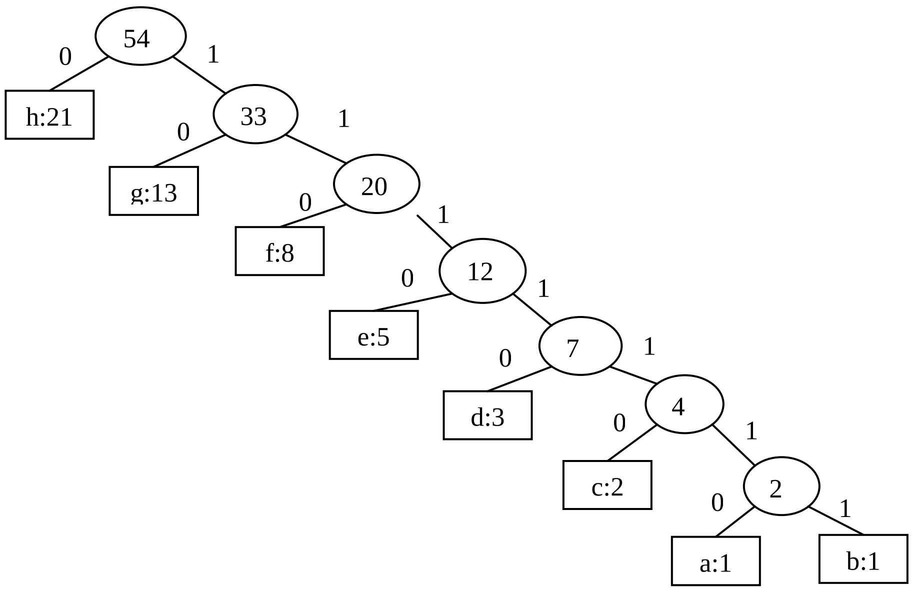
1. (Ch 16 Greedy Algorithms)

    a. (16.3-2) What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

10 pts

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Use the Huffman algorithm p. 388, build the following binary tree:



Code table:

Symbol	Code
a	1111110
b	1111111
c	111110
d	11110
e	1110
f	110
g	10
h	0

    b. (16.3-2) Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

10 pts

Symbol	Code
a	$1^{n-2}0$
b	$1^{n-2}1$
c	$1^{n-3}0$
…	…
nth symbol	0

Alternately, if Fib(i) is the frequency of the ith character, then the Huffman code hc(i) is:

hc(i) = $1^{n-2}0$ and $1^{n-2}1$ for i = 1 and 2, $1^{n-i-1}0$ for 2 < i <= n, where n is the size of the set of characters.

Note: It can be shown that the worst case for lengths of Huffman codes are reached when the distribution frequencies follow a Fibonacci sequence.

2. (Ch 17 Amortized Analysis)

10 pts a. (17.2-1) A sequence of stack operations is performed on a stack whose size never exceeds k. After every k operations, a copy of the entire stack is made for backup purposes. Show that the cost of n stack operations, including copying the stack is O(n) by assigning suitable amortized costs to the various stack operations.

The stack operations push(), pop(), stack-empty() are defined in section 10.1 of the text. Each of these operations takes O(1) time.

(i) aggregate analysis

The aggregate analysis for this problem is similar to that of section 17.1 for multi-pop().

```
COPY(S1, S2)
1 for i = 1 to top[S1]
2     S2[i] = S1[i]
3 top[S2] = top[S1]
```

The total cost of COPY is k = top[S1]. A worst case of a sequence of n stack operations followed by a COPY operation results when the n stack operations are PUSH operations, so that the cost of the COPY operation is O(n). Since each PUSH operation is O(1), any sequence of stack operations followed by a COPY is O(n + n) = O(n). The average cost of an operation is O(n)/n = O(1).

In aggregate analysis, the amortized cost of each operation is the average cost or O(1).

(ii) accounting method

Assign the following amortized costs

Push	3
Pop	0
Copy	0

Every Push costs 1, leaving 2 credits. A pop takes 1 credit. A Copy of a stack of size k takes k credits. After a sequence of n Push operations, the remaining credit is 2*n. The Copy costs n credits, leaving n credits for subsequent Pop operations. The amortized cost is 3*n or O(1) per operation.

10 pts b. (17.3-6) Show how to implement a queue with two ordinary stacks so that the amortized cost of each Enqueue and Dequeue is O(1).

The queue data structure and associated ENQUEUE and DEQUEUE operations are described in section 10.1 of the CLRS text.

Define two stacks of size n each, called E and D. ENQUEUE is implemented as a push onto the E stack. DEQUEUE is implemented as a pop from the D stack. The queue items are moved from E to D whenever D is empty and DEQUEUE is called.

```
E-to-D(E, D)
1   while TRUE
2        x = POP[E]
2        if NOT STACK-EMPTY[E]   PUSH[D,x]
3        else return x
```

```
ENQUEUE(E, x))
1   PUSH(E,x)
```

```
DEQUEUE(E,D)
1  if (STACK-EMPTY(D)) return E-TO-D(E,D,Direction)
2  else return POP(D)
```

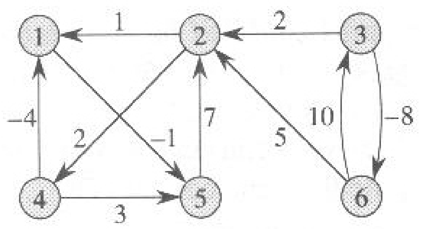
Analysis using accounting method:

Enqueue is assigned an amortized cost of 4, Dequeue is assigned amortized cost of 0. A sequence of k enqueue operations leaves a credit of 3*k. The first Dequeue uses 2*k credits. The subsequent dequeues use 1 credit each. So amortized cost of 4 per operation, or O(1).

3. (Ch 25 All Pairs Shortest Paths)

15 pts

a. (25.1-1) Run FASTER-ALL-PAIRS-SHORTEST-PATHS on the directed graph below, showing the matrices that result for each iteration of the respective loops.



L1	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

L2	1	2	3	4	5	6
1	0	6	∞	∞	-1	∞
2	-2	0	∞	2	0	∞
3	3	-3	0	4	∞	-8
4	-4	10	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	∞	0

L4	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-3	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

L8	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

10 pts

b. (25.2-4) The Floyd-Warshal algorithm (CLRS, p. 630) requires $\Theta(n^3)$ space. Show that the following procedure, which simply drops all superscripts, is correct, and only requires $\Theta(n^2)$ space:

```
FLOYD-WARSHALL'(W)
1  n ← rows[W]
2  D ← W
3  for k ← 1 to n
4      do for i ← 1 to n
5              do for j ← n
6                      do $d_{ij} \leftarrow \min(d_{ij}, d_{ik} + d_{kj})$
```

(i) Correctness:

Line 6 – if d_{ij} is min, then there is no change to the array, so the algorithm is unaffected

Line 6 - if $d_{ik} + d_{kj}$ is min, then a new shortest path via node k is found, and the ij entry is updated and potentially accessed in the same iteration. This makes a better shorter path available for reference earlier in an iteration than in the original algorithm. The new path can be no shorter than δ_{ij}, since if it were, δ_{ij} would not be the shortest path, a contradiction. Any other path which uses this path can be no worse off than in the original algorithm.

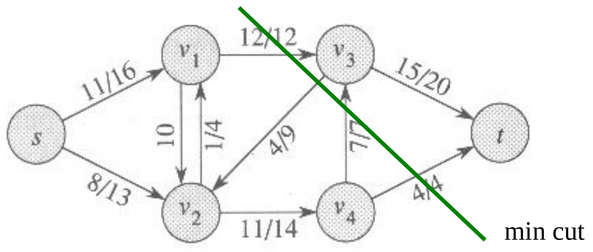
(2) Only requires $\Theta(n^2)$ space

The space used by the matrix D is n^2, as all references are in place.

4. (Ch 26 Maximum Flow)

5 pts

a. (26.2-3) What is the min-cut corresponding to the maximum flow for the following network flow graph? Draw the cut. Which two augmenting paths cancel flow that was previously shipped?



(b)

Augmenting paths which cancel flows:

path [s, v2, v3, t] cancels a flow of 4 from v2 to v3.

path [s, v1, v2, v3, t] cancels a flow of 1 from v1 to v2, and a flow of 4 from v2 to v3.

10 pts

b. (26-4.a) Updating maximum flow. Let G = (V, E) be a flow network with source s, sink t, and integer capacities. Suppose that we are given a maximum flow in G. Supppose that the capacity of a single edge (u,v) ∊ E is increased by 1. Give an O(V+E) time algorithm to update the maximum flow.

Let (u,v) be the edge whose capacity is increased by one.

Use BFS to find an augmenting path in G from s to t, taking time O(V + E). This path must contain (u,v). Augment the path, as in lines 6 – 8 in FORD-FULKERSON. This takes as most O(E) time. Total time is O(V+E).

```
MAX-FLOW-UPDATE(G)
1  p = a path in residual network Gf found by BFS
2  for each edge (u,v) in p
3        f[u,v] = f[u,v] + 1
4        f[v,u] = -f[u,v]
```

5. (Ch 26 Maximum Flow)

(26-5.a,b,c) Let G = (V,E) be a flow network with source s, sink t, and an integer capacity c(u,v) on each edge (u,v) $\in$ E. Let C = max $_{(u,v) \in E}$ c(u,v).

5 pts

a. Argue that a minimum cut of G has capacity at most C |E|

The maximum number of edges leaving s and entering t can be no more than |E|. The capacity of each edge can be no more than C. The flow from s to t can be no more than C |E|, which is an upper bound on the min cut.

5 pts

b. For a given number K, show that an augmenting path of capacity at least K can be found in O(E) time, if such a path exists.

(1) Form a graph G' from G in which each edge has capacity K or greater. This takes time O(E). If there is an augmenting path of capacity K in E, it is also the same path in E'. Find any path from s to t, since any path from s to t in E' is an augmenting path of capacity K. Finding this path takes at most |E'| edge traversals, where |E'| <= |E|. So the total time is O(E). See p. 660 of text.

(2) The capacity of an augmenting path is the minimum capacity of any edge on the path, so look for an augmentingpath whose edges all have capacity at least K. Do a BFS search to find the path, considering only edges with residual capacity of at least K. This takes O(V+E) = O(E) time, since |V| = O(E) is a flow network.

10 pts

c. The following modification of Ford-Fulkerson-Method can be used to compute a maximum flow in G.

```
MAX-FLOW-BY-SCALING(G,s,t)
1  C ← max_{(u,v) ∈ E} c(u,v)
2  initialize flow f to 0
3  K ← 2^{⌊lg C⌋}
4  while K ≥ 1
5      do while there exists an augmenting path p of capacity at least K
6              do augment flow f along p
7         K ← K / 2
8  return f
```

Argue that MAX-FLOW-BY-SCALING returns a maximum flow.

MAX-FLOW-BY-SCALING uses the FORD-FULKERSON method. It repeatedly augments the flow along an augmenting path until there are no augmenting paths of capacity >= 1. Since all capacities are integers, and the capacity of an augmenting path is positivie, this means that there are no augmenting paths whatsoever in the residual graph Gf. Thus, by the max-flow min-cut theorem, case 2 no augmenting paths, MAX-FLOW-BY-SCALING returns a maximum flow.