COP 4531 Test 1 Each problem worth 10 points. Show ALL work for credit

1. Using only the operations in the text for the ADT "List", write a Pascal function *length* which returns the length of the parameter L: List.

2. Using only the operations in the text for the ADT "Binary Tree", write a Pascal function *hotel_space* which returns the number of nodes in the subtree rooted at the parameter n: node.

3. Assuming set_type is the Pascal data type set of 0..63 write implementations of the following operations from ADT "Set":
A. MAKENULL(var A: set_type) B. UNION(A, B: set_type; var result: set_type)
CD. function SIZE (A: set_type): integer;

4. A. Arrange in increasing order: $O(n!)$, $O(2^n)$, $O(30*n\log_2 n)$, & $O(10*n^2)$.
B. Arrange in increasing order: $4!$, 2^4, $30*4\log_2 4$, & $10*4^2$.

5. In the procedure do_nothing (below) write--in terms of big oh of n (and simplify--i.e. use $O(n^3)$ instead of $O(2*n^3 + 7)$.)--the number of times:
A. routine_one is called. B. routine_two is called. C. routine_three is called.

```
procedure do_nothing (n: integer);
var   i, j: integer; done: boolean;
BEGIN
for i := 1 to n - 1 do
      for j := i + 1 to n do
            routine_one;

i := 45; done := FALSE;
while not done do
      begin i := i + 3; done := ( i >= n ); routine_two; end;

i := 1; done := ( i >= n );
while not done do
      begin i := 2 * i; done := ( i >= n ); routine_three; end;
END;
```

6. Remove the recursion from:

```
procedure abc(p: position, L: list);
BEGIN
if p = NULL then return; printstuff@p; abc( NEXT(p, L), L);
END;
```

7. Implement the operations ENQUEUE and DEQUEUE for the ADT "Queue" using the data structure provided below. Write these operations as boolean functions which return TRUE if the operation was sucessful. Your queue needs to be able to hold MAXQ elements. All fields in the queue record should be correct between calls.

```
const LAST = MAXQ - 1;
type  queue = record
            count, front, rear: integer;
            full, empty: boolean;
            storage: array [0..LAST] of elementtype;
      end;
(* initially assume count, front, rear are zero, full is FALSE and
      empty is TRUE *)
```

8. A certain binary tree has each leaf labeled "0" and each internal node (i.e. non-leaf) labeled "1". Also each internal node has exactly two children. If we read the labels while traversing the tree in preorder, we get the sequence 11001101000. A. Draw the tree. B. List the labels while traversing the tree in inorder.

9. Using the data structures (below), implement the two operations on binary trees given by:

A. function IS_OLDER(n, m: node): updowntype; which returns equal if n = m, ancestor if n is an ancestor to m, descendant if n is a descendant of m and neither otherwise.

B. function WHICH_HAND(n, m: node): leftrighttype; which returns same if n = m, leftof if n is to the left of m, rightof if n is to the right of m and nope otherwise.

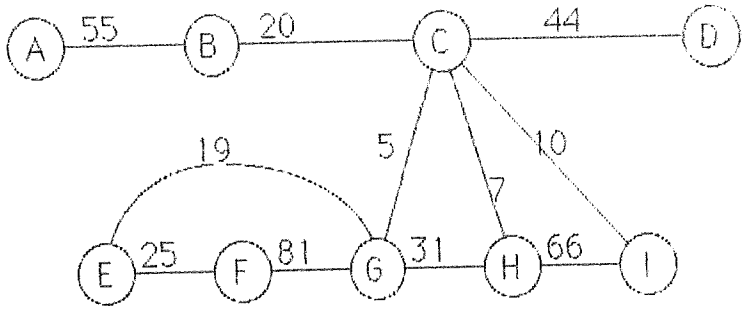
```
type  updowntype = (equal, ancestor, descandant, neither);
      leftrighttype = (same, leftof, rightof, nope);
      node = record
            left_child, right_child, parent: ^node;
            label: labeltype;
      end;
```

10. Show the run time of the function non_poly (below) is $O(2^n)$ by writing and solving the (non-homogenuous!) recurrence relation for T(n), the run time of non_poly(n). (Be sure to explain why $A \neq 0$ in the solution. $T(n) = A*2^n + \ldots$)

```
function non_poly(n: integer): integer;
var   a, b integer;
BEGIN
if n <= 1 then return 17;
a := non_poly(n - 1); b := non_poly(n - 1); return ( a - b + 35 );
END;
```

COP 4531 Test 2 All problems worth 10 points. Show ALL work for credit.

1. For the graph below, list the edges chosen for a minimum spanning tree (in the order chosen) by A. Prim's algorithm. B. Kruskal's algorithm.



2. For the graph above, draw the (unique) spanning tree obtained by doing A. Depth First Search. B. Breath First Search.

3. For the graph above, A. give the "cost" matrix A as it would be (after initialization, but before the main loop) at the start of Floyd's algorithm. BC. Note that this matrix A doesn't change for k = 1 (A) of the main loop. Show the matrix A after the k= 2 (B) and k= 3 (C) of the main loop.

4. For each of the ADT set implementations: A. bit vectors B. (sorted) linked list and C. hash table give "tight" big O estimates on the run time complexity of each of the following set operations: (i) MEMBER (ii) INSERT (iii) DELETEMIN (iv) UNION. Assume the set(s) have size is n and the hash table(s) have 3n elements.

5. A hash table (array[0..7]) currently contains (empty, deleted, 17, empty, 68, empty, deleted, 31). The following list of operations are to be done sequentially on this table. For each operation give the number of probes needed to preform the operation. Also show the hash table at the end of this sequence of operations. The hash function is x mod 8, and we are using linear resolution of collusions. The operations are: A. insert 35. B. insert 63. C. member 17. D. insert 45. E. delete 17. F. delete 31. G. member 47.

6. For a digraph G with n vertices and any vertex x in G define indegree(x) (respectively outdegree(x)) to be the number of edges of G pointing into x (respectively out of x). A. If G is represented as an adjacency matrix A[i,i], write O(n) functions to compute the indegree and outdegree. B. Suppose G is stored by adjacency lists, and we want to fill indegree, outdegree: array[1..n] of integer, write a O(e) algorithm to fill these two arrays in one pass throught the edges.

7. Heap big problems: figure out the (worst case) run times (in "tight" big O of m) of all three pascal procedures below. Swap takes O(1) time.

```
FixHeap(var big: array[1..n] of key; m: integer);
var i, j: integer;
begin
      for i := m downto 2 do begin
            j := i div 2;
            if big[i] < big[j] then swap(big[i], big[j]);
      end;
end;
AdjustUp(m: integer; var big: array[1..n] of key);
var i, j; integer;
begin
      i := m; j := i div 2;
      while j > 0 do begin
            if big[j] > big [i] then swap(big[i], big[j]) else j:=0;
            i := j; j := i div 2;
      end;
end;
FixHeap2(var big: array[1..n] of key; m: integer);
var i: integer;
begin
      for i := 2 to m do AdjustUp(i, big);
end;
```

8. A. Given the fields preordinal and postordinal have been constructed via a depth first search for some digraph G. Write the function edgekind (see below) which uses this info to determine what kind of edge is the edge from x to y in this DFS tree (or forest) digraph. (Noedge if there is an error or if x and y represent the same vertex.)

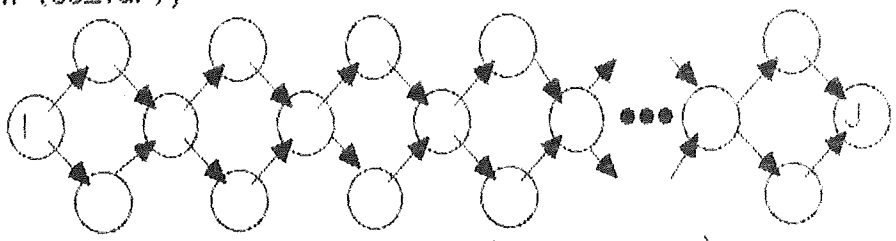
```
type vertex = record preordinal, postordinal: integer; ... end;
      edgetype = (noedge, forward, back, cross);
function edgekind (x, y: vertex): edgetype;
(* returns the edgetype of a edge from x to y in the DFS tree *)
```

B. Write an <u>algorithm</u> which will determine the fields preordinal and postordinal.

9. A digraph G is said to be unilaterally connected if for each pair of vertices x and y, either there is a path from x to y or a path from y to x. Design an algorithm to determine if G is unilaterally connected or not. (You may use any algorithms in the text as parts of your algorithm.) How do you store G? What is the run time of your algorithm?

10. Explain why the function below doesn't work on general digaraphs but does work on dag's. For the dag below compute both number_of_paths(i,j) and the runtime in terms of big O of n, where n is the number of vertices.

```
function number_of_paths(i, j: vertex): integer;
so_far := 0;
for each vertex k adjacent from i do
      so_far := so_far + number_of_paths(k, j);
return (so_far);
```

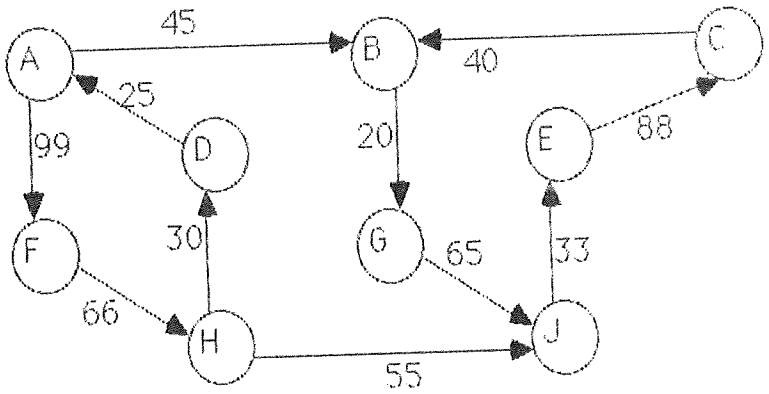


(Note there are about n/3 vertices in the center row.)

COP 4531 FINAL Each problem worth 10 points. Show ALL work for credit.

1. Using only the operations in the text for the ADT "List", write a Pascal function *length* which returns the length of the parameter L: List.

2. For the graph below, list the edges (ignore direction) chosen for a minimum spanning tree (in the order chosen) by A. Prim's algorithm. B. Kruskal's algorithm.



3. For the graph above, draw the (unique) spanning forest obtained by doing A. Depth First Search. B. Breath First Search.

4. For the graph above, find all the strongly connected components.

5. Solve for T(n), if $T(n) = 3\,T(n/2) + 5\,n^2$ and $T(1) = 120$.

6. An OPEN hash table (array[0..4] of list pointers) currently contains the numbers17, 68, and 31). The following list of operations are to be done sequentially on this table. For each operation give the number of probes needed to preform the operation. Also show the hash table at the end of this sequence of operations. The hash function is x mod 5, and we are sorting each list of elements in a bucket. The operations are: A. insert 35. B. insert 63. C. member 17. D. insert 45. E. delete 17. F. delete 31. G. member 47.

7. Write two efficient procedures for heaps: (Here the heap is stored in the first n entries of the array heap[1..maxheap] with its min at heap[1].) A. Decrease_j_by_delta (j, delta: integer); which does heap[j] := heap[j] - delta and rearranges to preserve the heap property of heap[1..n]. (Note that no knowledge of n is needed here). B. Increase_j_by_delta(j, delta, n: integer); which does heap[j] := heap[j] + delta and rearranges to preserve the heap property of heap[1..n].

8. In the procedure do_almost_nothing (below) write--in terms of big oh of n (and simplify--i.e. use $O(n^3)$ instead of $O(2*n^3 + 7)$.)--the number of times: A. check_one is called. B. check_two is called. C. check_three is called.

```
procedure do_almost_nothing (n: integer);
var   i, j, k: integer; done: boolean;
BEGIN
for i := 1 to n - 1 do
      for j := i + 1 to n do
            for k := 1 to 2 do check_one;

i := -5; done := FALSE;
while not done do
      begin i := i + 3; done := ( i >= n ); check_two; end;

i := 1; done := ( i >= n );
while not done do
      begin i := 2 * i; done := ( i >= n ); check_three; end;
END;
```

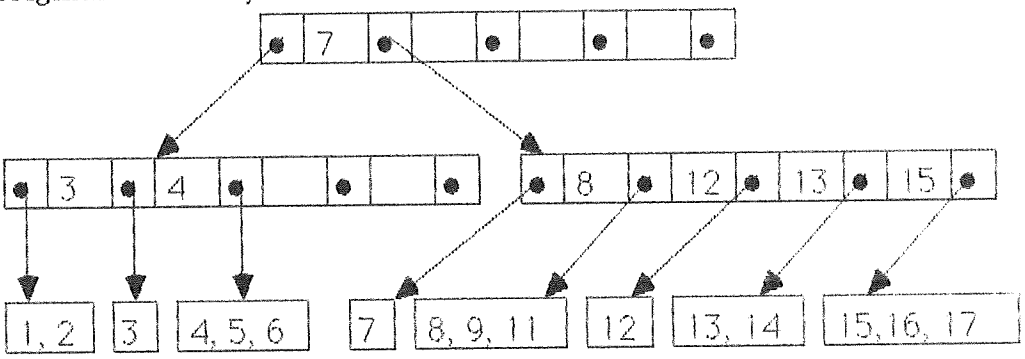
9. Implement the operations ENQUEUE and DEQUEUE for the ADT "Queue" using the data structure provided below. Write these operations as boolean functions which return TRUE if the operation was sucessful. Your queue needs to be able to hold MAXQ elements. All fields in the queue record should be correct between calls.

```
const LAST = MAXQ - 1;
type  queue = record
            count, front, rear: integer;
            full, empty: boolean;
            storage: array [0..LAST] of elementtype;
      end;
(* initially assume count, front, rear are zero, full is FALSE and
      empty is TRUE *)
```

10. For each of the ADT list implementations: A. (sorted) array implementation. B. (unsorted) linked list and C. (sorted) linked list give "tight" big O estimates on the run time complexity of each of the following list operations: (i) LOCATE (ii) INSERT (iii) DELETEMIN (iv) PRINTLIST. Assume the list(s) have size is n.

11. Use Dynamdic programing to improve the run time of Fun(n, i) which returns 1 if i <= 1 or i >= n and otherwise it returns Fun(n-1, i)+Fun(n, i-1). If m = n + i, show the old run time was $O(2^m)$ and the new run time is $O(m^2)$.

12. The picture below is of a B-tree, the leaves can hold up to three records. Show the resulting B-tree when A. 10 is inserted. B. 3 is deleted (to the original tree not your answer in A.)



13. Design an algorithm to determine the length of the longest cycle (or zero, if G is a dag) in the digraph G. (You may use any algorithms in the text as parts of your algorithm.) How do you store G? What is the run time of your algorithm?

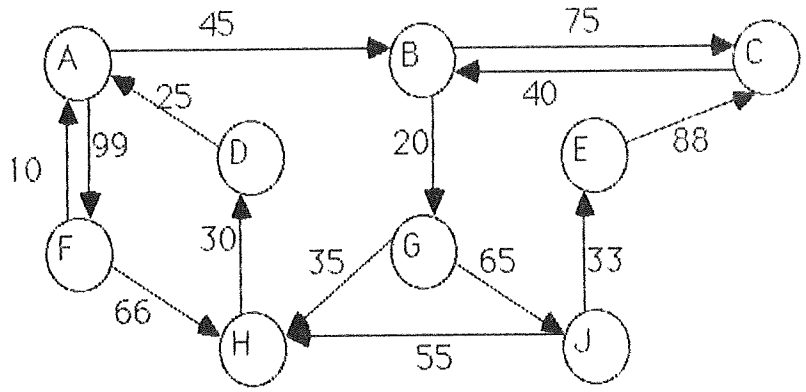
14. Your boss asks you to consider implementing the SIZE operation on the ADT set via a divide and conquer call to local_size (see below). Compute the run time of such a function (SHOW ALL WORK). What should you tell your boss? (You may assume n is a power of two.)

```
function local_size(A: set; first, last: integer): integer;
BEGIN
      mid := half_way_between( first, last); (* O(1) time *)
      if first = last then return (1 if last in A else 0)
      else return (local_size(A, first, mid)
                        + local_size(A, mid+1, last))
END;
```

15. Explain (in the context of algorithms) the terms: A. Greedy.
B. Backtracking. C. Local Search.

16. Define ADT and contrast it with "its implementations". Give several examples of ADT's, operations on same and implementations.

Practice Test 2 Cop 4531 Data struct & AA Spring 1987



For the graph above do everything, namely:
1. A. Write the adjacency cost matrix C[i, j]. B. Write the adjacency lists (with weights). C. What percent of the edges are present in the graph ($e = ? \% * n^2$). D. What is the transitive closure of the graph?
2. A Draw a spanning tree via DFS. Now ignore the direction of the edges, for the undirected graph find: B. a spanning tree via DFS. C. a spanning tree via BFS. D. The minimal spanning tree.
3. A. Use Floyd's algorithm and all your spare time to find the paths of minimal length from any vertex to any other vertex (all pairs shortest paths). B. Use Dijkstra's algorithm to find all shortest paths from vertex A.
4. Show how the test for Acyclity works on this graph. Show how the algorithm for finding strong components works here. Ignore direction and show how the algorithm for finding articulation points and biconnected components does it stuff.
5. Ignore direction and list the edges in the order that Prim's and in the order that Kruskal's algorithm would pick the edges. Figure out which algorithm i forgot and do that one too after deleting the edges AF, CB, and BG (top sort).
6. A graph with 4 or more vertices is said to be 3-connected if the removal of any 2 vertices does not disconnect the graph. Describe an algorithm which will test any undirected graph for 3-connectedness. You may use any algorithms in chapters 6 & 7 as needed. What is the speed of your algorithm?
7. Write a O(n) function which will take the array [1..n] of integer and rearrange it into a heap (partially ordered tree). Write a O(log n) routine decrease_key (j, delta) which decreases HEAP[j] by delta and then rearranges stuff to preserve the heapness. Also write deletemin.
8. Hashing open closed many collusion resolutions.
9. Sets (ADT) implemented as bit vectors. as linked lists.
10. How do you color the edges of a graph s.t. adjacent edges have dif.colors?

Bellenot's spring 1987
COP 4531-01 Data Structures and Algorithm Analysis
office 218 LOV office hours MWF 12:30 - 1:20 & by appointment

PREREQ'S: C- or better in both Pascal 2 & Discrete Math 2.
Text: Aho, Hopcroft & Ullman, Data Structures and Algorithms.
Topics: Chapters 1-4, 6, 9, 10 and 11 (More or less as time permits.)

Tests: Final 12:30-2:30 Tues 28 Apr -- counts 30% of your grade.
Test1 tentatively Wed 11 Feb -- counts 15% of your grade.
Test2 tentatively Wed 1 Apr -- counts 15% of your grade.

Grades: *Attendance is required* A student is "absent" if he or she doesn't turn in the assigned work at the beginning of class. Five such absences yields an automatic "F"! Otherwise grades are based on the classic 90%, 80%, 68%, 60% cut-offs.

Programs: 30% of your grade, due every other Wednesday starting 21 Jan. Programs may be graded on correctness, clarity, conciseness, the ease in which they could be modified, speed and their re-useability.

Homework: (The remaining 10% of your grade.) Problems mostly from the text. Attendance is taken by the "attempt" to do all of the problems. There will be homework due each day except when tests are given and days when programs are due.

Late Work is NOT accepted!

Program 1: (Due 21 Jan) Programs collected (filename=REVIEW) @high noon, hard copy due in class.

Homework 1: Problems 1.1, 1.3 & 1.16 due 12 Jan
2: Problems 1.6, 1.12 & 1.18 due 14 Jan
--19 Jan is MLK Day -- no classes. --

DS&AA Spring 1987 REVIEW program Due Wed 21 Jan.

Infinite precision postfix desk calculator

input line	meaning or action
number	push the number onto the stack
'A' or '+'	push(pop + pop)
'S' or '-'	a := pop; push(a - pop) ; note: '-' could be the start of number
'M' or '*'	push(pop * pop)
'X'	exit program
'T'	print top of stack
'P'	print all of the stack
'Q' or '/'	a := pop; push(a/pop) ; note: integer divide-no remainder
'R'	the remainder from divide (or modulus)

(save Q and R for last)

"number" is a string of chars in the range '0'..'9' which may be preceded by '-', if the number is negative. Since a number can be longer than one terminal line the char '&' is used indicate that the number continues on the next line. Printing of "number" types no more than 75 characters per line, again '&' is added to show the number continues on the next line.

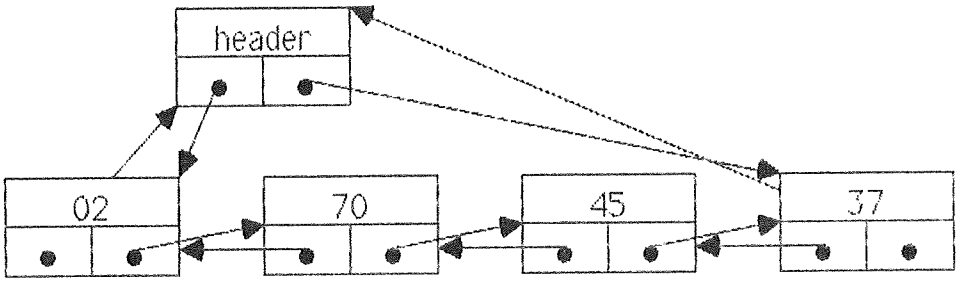
```
CONST   BIGGEST = 99 (* YOU MAY ASSUME IT'S POWER OF 10 - 1 *);
        MAXCOL = 75;
        MAXSTACK = 10;


TYPE    PLUSMINUS = (MINUS, PLUS);
        CHUNK = 0..BIGGEST;
        LONGINTPTR = ^LONGINT;

        LONGINT = RECORD
                HEADER:     BOOLEAN;
                SIGN:       PLUSMINUS;
                DIGITS:     CHUNK;
                POSITION:   INTEGER;
                UP, DOWN:   LONGINTPTR;
        END;
```

```
VAR         ZERO:  LONGINTPTR;
            SP:    0..MAXSTACK;
            STACK:       ARRAY[1..MAXSTACK] OF LONGINTPTR;
```

Thus a typical longint looks like the figure below. Only the digits and up, down fields of the cell longint are shown.



The number 2704537

27&
0&
453&
7

note: (one of) the alternate way(s) of inputing 2704537.
(the start of arrows are sometimes behind the cell)

Obviously the program should be written so that it still works when BIGGEST is any $10^n - 1$ (n would be chosen to be optimal for a given machine.) (Of course binary chunks might be optimal -- but then printing out the numbers becomes more complicated.)

Zero: there are too many ways to represent zero: (i) Just a header with sign=+, (i') Just a header with sign=-, (ii) One cell with digits=00, sign=+, (ii') One cell with digits=00, sign=-,...(n) n-cells all with digits=00, sign=+, (n') n-cells all with digits=00, sign=-,... .

What to do about leading zero's? (see above)

It is easier to deal with positive numbers. define a/b = (-a)/(-b) when b < 0, and define a/b = -(-a/b) when a < 0 < b. (Remainders are done similarly.)

Try to recover from errors. "unknown function", "but that would be dividing by zero", "the stack is empty" or "not enuff operands on stack". In particular, don't start an operation that can't be completely executed.

Q & R algorithm by example: (grammer school long division revisited)
12 34 56 78/22 33: since 12/22 < 1 consider 12 34/22 = 56.09... quess 56 (=56 00) times. 56 00 * 22 33 = 12 50 48 00.

12 34 56 78
less 12 50 48 00 which is negative so change to 55 00 and
add 00 22 33 00 which is positve!

now 00 06 41 78 is the partial remainder quess 29 (near 6 41/22) but instead we quess 28 to illustrate a point. 28 * 22 33 = 6 25 24. This is just right but the program doesn't know it.

06 41 78
less 06 25 24 which is positive so we change 28 to 29 and
sub 00 22 33 which is negative so we change 29 to 28 and restore
add 00 22 33

the quotient is 55 28 and remainder 16 54.

Program 2 due 4 Feb 1987

string is a sequence of 1..10 chars in 'A'..'z'
$\not{b}$ is a sequence of $0..10^{100}$ blanks

make a dictionary of words using 'qlib'

input	operation
+ $\not{b}$ string	add string to dictionary
- $\not{b}$ string	delete string from dictionary
? $\not{b}$ string	print position of string in dictionary -1 if absent
%	print dictionary

Also Answer:

1. List changes needed in qroutines so that your dictionary would hold at least 100 elements

Cop 4531 Spring 1987
Programing assignment: "Rehash" due Wed 4 Mar 1987

Besides the compiled listing and the file REHASH (collected electronically @ high noon), this assignment requires the drawing of a few graphs. In particular, those needed to fill the follwing table.

Table	Linear		Shift		Second Hash	
S or U	S	U	S	U	S	U
20%						
40%						
60%						
70%						
80%						
85%						
90%						
95%						
98%						

Average Number of Probes (S = sucessful MEMBER, U = unsucessful)
Percentage is that percent of the hash table which is full.
Linear = Linear resolution of collisions, Shift = a shift register sequence
The hash table size will be 128 (127 for the second hash function case), the things to hash will be random integers in the range $1..2^{16}$. The first hashing function will be mod 128 (127). (The second hash function will be h(x) = (x div 127) mod 127.) The shift register sequence will start with the smallest k which will produce a permutation of 1..128-1.

The numbers to hash will be generated via a random number generator. The Cyber's random number generator can be called via (*$I'random' *) and there after calls to the function "random" produces a random real in the interval [0,1). Collect statistics by method outlined in class.

Sucessful Member average = average number of probes to insert.

Unsuccessful Member average is from another random sample (throw away those which are ~~uns~~ successful.)

COP4531-01 Program 3 Huffman Coding due 18 Feb. 1987

On a file named HUFFMAN write a program which reads its input (or any textfile via lgo,textfoo) and then produces a best Huffman coding of the characters in the textfoo.

Output should contain a statement like:
"Input text contained x characters for a total of y bits,
encoded text would contain z bits, a saving of w percent."

followed by a collection of lines like:

character	appears	probability	huffman code
e	104	.1200	01
' '(blank)	100	.1153	001
...			
t	5	.0058	0000001

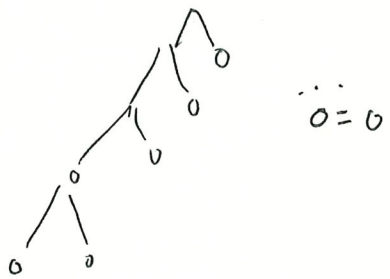
not real data!

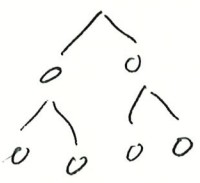
Hints and aids: 1 There is a file TOWERS/un=3374*** which all programs will be checked on. 2. A good part of chapter 3 of the text is devoted to this problem.

EOLN ?

missing characters? what should we do with these?

0 = e

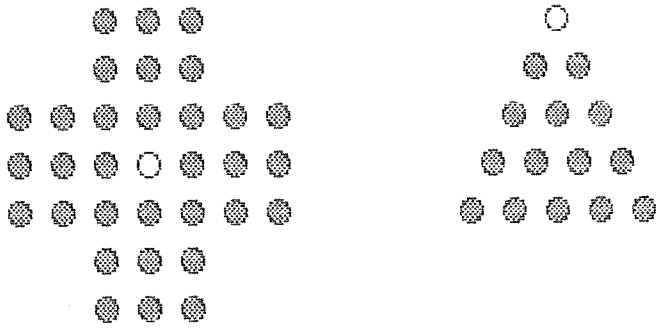




Cop 4531
Spring 1987 The Last Programming Project
Due: Wednesday 15 April 1987

DFS (depth first search) of a game tree.

Various dumb games consist of a peg board and pegs which can "jump over and take other pegs", the object being to end with exactly one peg left on the game board. These games come in various geometric shapes, a coupie are illustrated below.



The Main program (on the file "SEARCH") reads the input and writes the output as outlined below. The game tree which we are doing a DFS search on is never explicitly stored in the computer. Rather we will keep a stack like structure which stores the moves made so far and a Pascal set ("PEGS_AT"(?)) whose members are the vertices where there are currently pegs. Note that using this set structure limits us to boards of about 59 holes or less.

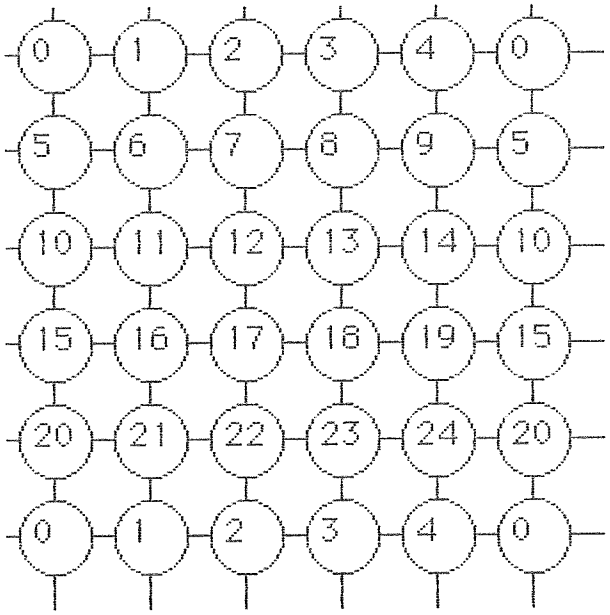
In addition to the main program, two small programs are also required. The first (on the file "G2JUMP") will translate a geometric board to a input file board and will be different for each student. The second (on the file "J2PICT") will take the output file and draw "pictures" of the peg board as it is being played (for the particular geometric board).

INPUT: The input textfile "board" consists of the following:
(I) A line containing the name of game (optional).
(II) Several lines (at least two) indicating the geometric legal jumps. The first line contains the string "JUMPS", the last line contains the string "ENUFF" and the middle lines (if any) are of the form $vertex_1$ space(s) $vertex_2$ space(s) $vertex_3$, where space = ' ' and the three vertices are in the range 0..maxvertex. The meaning of such a line is that if there is a peg at both $vertex_1$ and $vertex_2$ and no peg at $vertex_3$, then a jump from $vertex_1$ over

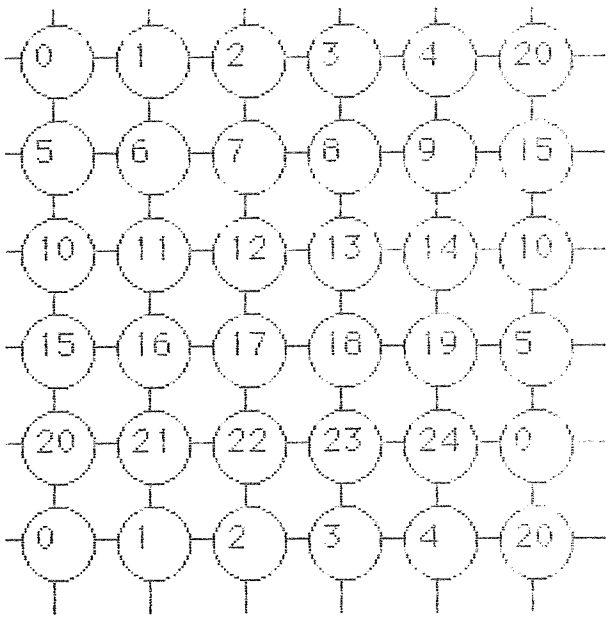
vertex$_2$ to vertex$_3$ is legal, leaving no pegs at vertex$_1$ and vertex$_2$ but vertex$_3$ now has a peg.
(III) A line with either the string "PEGS AT" or the string "NO PEGS AT". The following lines consist of vertices separated by either spaces or new lines. In the case of "NO PEGS AT", pegs are to be put at each vertex of a geometric legal jump except for the vertices in the following lines.
(IV) An optional line with the string "SUCCESS IS LESS THAN OR EQUAL TO" followed by an integer which is one or more.
(V) A line with the string "START HOPPING".

OUTPUT: Either "THE GAME (optional name (in quotes) if it exists) CAN'T BE PLAYED TO SUCCESS.(new line)AT BEST WE CAN ONLY GET DOWN TO (k the fewest pegs) PEGS" or "THE GAME (optional name (again in quotes) if defined) CAN BE SUCCESSFULLY PLAYED AS FOLLOWS:", followed by lines containing the sequence of moves which was found to be successful, which is followed by "PEGS REMAIN AT ONLY THE FOLLOWING HOLES:", and finally a line listing where there are still pegs.

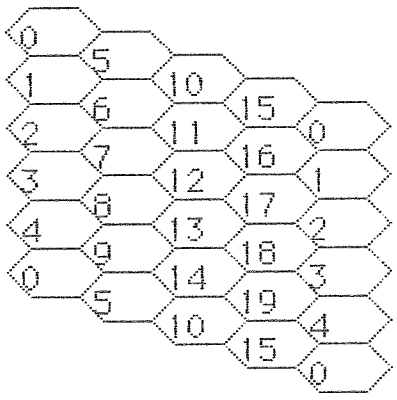
```
SAMPLE GAME          |     1   2   3
JUMPS                |     4   5   6
1 2 3                |     7   8   9
1 4 7                |
2 5 8                |     O   X   X
3 2 1                |     X   X   X
3 6 9                |     X   X   X
4 5 6                |
6 5 4                |     X   O   O
7 4 1                |     X   X   X
7 8 9                |     X   X   X
8 5 2                |
9 8 7                |     X   X   O
9 6 3                |     X   O   X
ENUFF                |     X   O   X
NO PEGS AT           |
1                    |     O   O   X
START HOPPING        |     X   O   X
                     |     X   O   X
                     |
                     |     X   O   X
                     |     O   O   X
                     |     O   O   X
```

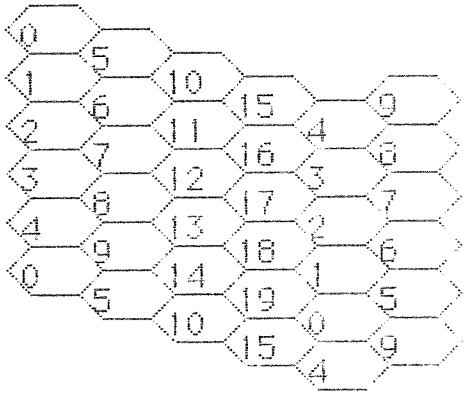


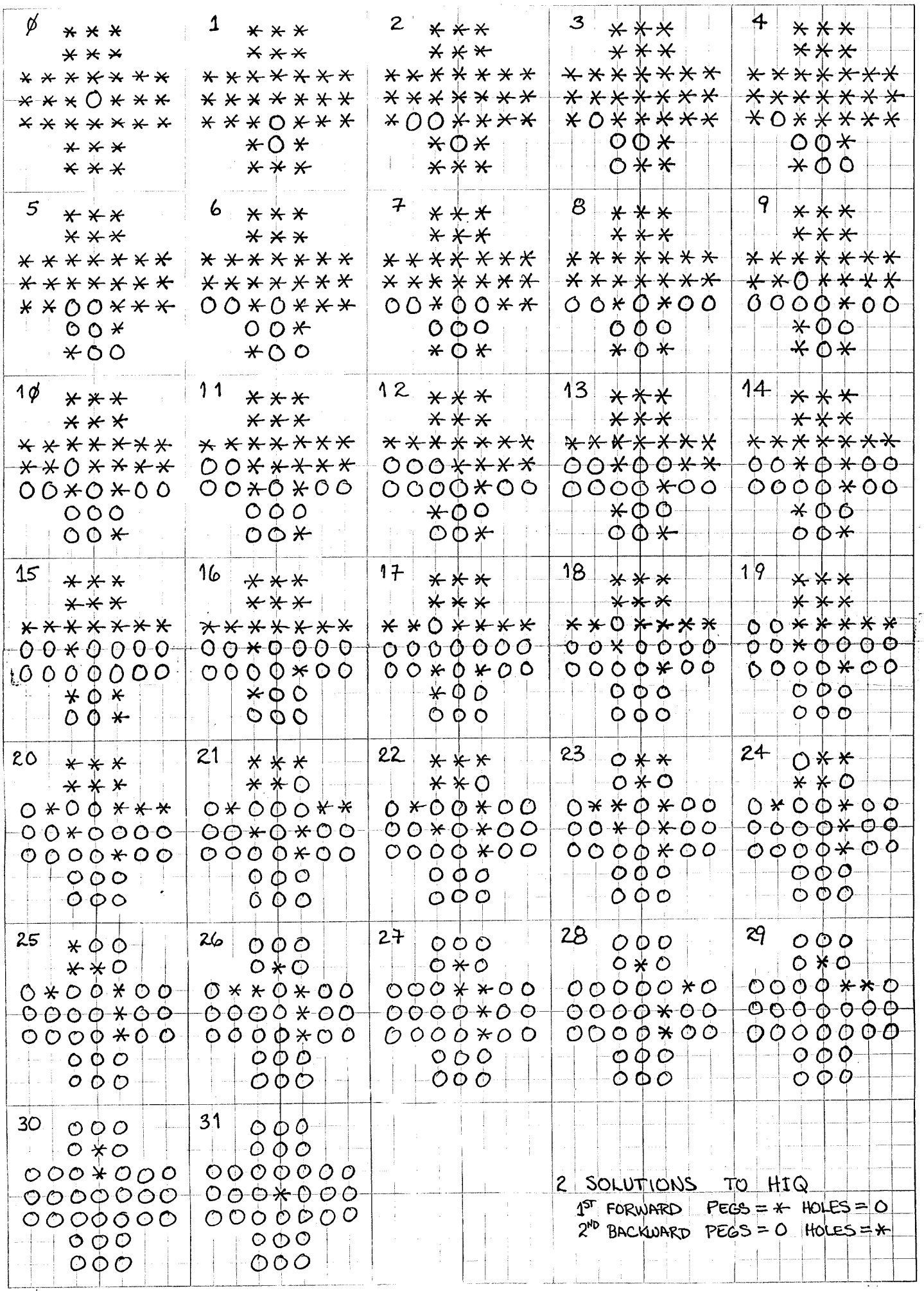
TORUS



KLEIN BOTTLE






0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
2 SOLUTIONS TO HIQ
1ST FORWARD PEGS = * HOLES = O
2ND BACKWARD PEGS = O HOLES = *