Exercises
Chapter 1 (Fundamentals)

1. Given the following procedure **hanoi**:

```
procedure hanoi(n, beg, aux, end);
begin
   if n = 1 then
   writeln(beg, end)
   else
   begin
      hanoi(n-1, beg, end, aux) ;
      writeln(beg, end);
      hanoi(n-1, aux, beg, end);
   end
end;
```

Let C(n) be the number of disk moves from a peg to another peg. Find the recurrence relation for the above program. And prove that
$C(n) = 2^n - 1$.

2. Consider the following recursive algorithm for computing the sum of the first *n* cubes: $S(n) = 1^3 + 2^3 + \ldots + n^3$.

```
Algorithm S(n)
// input: a positive integer n
if n = 1 then 1
else return S(n-1) + n*n*n
```

Assume that the multiplication is the basic operation in the above algorithm. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

3. Given the following procedure that finds the maximum and minimum elements in an array.

```
procedure MAXMIN(A, n, max, min)
/* Set max to the maximum and min to the minimum of A(1:n)  */
begin
  integer i, n;
  max := A[1]; min:= A[1];
  for i:= 2 to n do
    if A[i] > max then max := A[i]
    else if A[i] < min then min := A[i];
end
```

Let C(n) be the complexity function of the above algorithm, which measures the number of element comparisons.

(a) Describe and find C(n) for the worst-case.
(b) Describe and find C(n) for the best-case.
(c) Find C(n) for the average-case when n=3.

4. Suppose Module A requires M units of time to be executed, where M is a constant. Find the complexity C(n) of the given algorithm, where n is the size of the input data and b is a positive integer greater than 1.

```
j:= 1;
while j <= n do
begin
    call A;
    j := j*b
end
```

5. Suppose module A requires M units of time to be executed, where M is a constant. Find the complexity C(n) of the following algorithm, where n is the size of the input data.

```
for i:= 1 to n do
  for j:= 1 to i do
    for k:= 1 to j do
        moâ ñun A
```

6. Given a recursive program with the following recurrence relation:

$C_N = 4C_{N/2} + N$, for $N \geq 2$ with $C_1 = 1$

when N is a power of two.
Solve the recurrence.

7. Given a recursive program with the following recurrence relation:

$C_N = 2C_{N/2} + N + 1$, for $N \geq 2$ with $C_1 = 0$

when N is a power of two.
Solve the recurrence.

8. Given a recursive program with the following recurrence relation:

$C(n) = c + C(n-1)$ for $n > 1$ with $C(1) = d$

where c,d are two constants.
Solve the recurrence

9. Given a recursive program with the following recurrence relation:

$C(n) = 2C(n/2) + 2$ for $n > 1$ with $C(2) = 1$

Solve the recurrence

10. Given a recursive program with the following recurrence relation:

$C(n) = 2C(n/2) + 3$ for $n > 1$ with $C(2) = 1$

Solve the recurrence

11. Given a recursive program with the following recurrence relation:

$C_N = 4C_{N/2} + N^2$, for $N \geq 2$ with $C_1 = 1$

when N is a power of two.

Solve the recurrence.

12. Given a recursive program with the following recurrence relation:

$C_N = 2C_{N/2} + N^2$,

a. Draw recursive tree for the above recurrence relation.

b. Prove that C(N) = O(N)

13. Given the selection sort algorithm as follows:

```
procedure selection;
var i, j, min, t: integer;
begin
        for i :=1 to N-1 do
        begin
                min :=i;
                for j :=i+1 to N do
                  if a[j]<a[min] then min :=j;
                t :=a[min]; a[min]  :=a[i];
        a[i] :=t;
        end;
end;
```

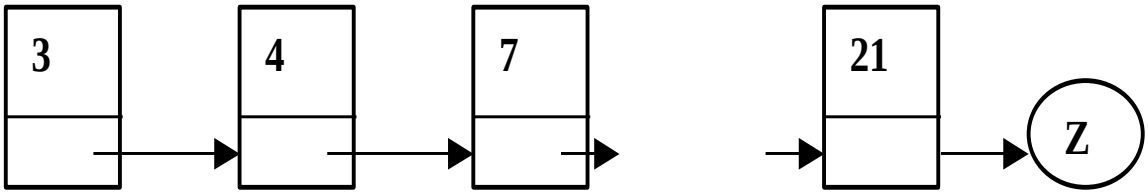
Prove that selection sort uses about N exchanges and $N^2/2$ comparisons.

14. Given the bubble sort algorithm as follows:

```
procedure bubble;
var j,j, t: integer;
begin
        for i :=N downto 1 do
                for j := 2 to i do
                if a[j-1] > a[j] then swap(a[j],a[j-1]);
end;
```

Prove that bubble sort uses about $N^2/2$ comparisons and $N^2/2$ exchanges in the worst case and used about $N^2/2$ comparisons and $N^2/4$ exchanges in the average case.

15. Sequential search on a sorted singly linked list. Given a sorted singly linked list as in the following figure.



And given the pseudocode of the procedure that can search for a key value *v* in the linked list as follows. Notice that we use a null node *z* at the end of the linked list.

```
type link = ↑ node
  node = record key, info: integer;
  next: link
end;
var head, t, z: link;
     i: integer;

procedure initialize;
begin
  new(z); z↑.next: = z;
  new(head); head↑.next:= z
end;

function listsearch (v: integer; t: link): link;
begin
  z↑.key: = v;
  repeat
      t:= t↑.next
  until v < = t↑.key;
  if v = t↑.key then listsearch:= t
  else listsearch: = z
end;
```

a) Prove the following property: Sequential search (sorted linked list implementation) uses about N/2 comparisons for both successful and unsuccessful search (on the average).
b) Analyse the complexity in average case for sequential search on an unsorted linked list.

16. Let M be the size of the hash table. In open hashing with separate chaining, keys are stored in linked lists attached to cells of a hash table. Each list contains all the keys hashed to its cell.

What is the time complexity for inserting a key into a hash table that has been created from N keys from an initially empty hash table. For collision resolution, this hash table

applies separate chaining with ordered lists? Answer the same question for the case of unsorted lists.

Exercises
Chapter 2 (Divide-and-Conquer)

1. Given a recursive program with the following recurrence relation:

$C(n) = 2C(n/3) + 1$ for $n > 1$ with $C(1) = 1$

Solve the recurrence relation to find the complexity of the program.

2. Write the Quicksort algorithm that uses the *rightmost* element as the pivot (by modifying the *quicksort2* procedure). And trace by hand the algorithm when it works on the following keys: A S O R T I N G E X A M P L E.

3. Given the following list of integers 66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30. Trace by hand the Quicksort algorithm that uses the leftmost element as the pivot to sort these integers.

4. If the array is already in descending order, estimate the total number of comparisons when we apply *Quicksort* on that array. Derive the worst-case complexity of the Quicksort.

5. For the version of QuickSort given in this chapter, are arrays made up of all equal elements the worst-case or best-case, or neither of QuickSort?

6.

a. Show the merges done when the recursive Mergesort is used to sort the keys E A S Y Q U E S T I O N.

b. Given the following algorithm:

```
procedure mergesort2(l, r: integer);
var i, j, k, m : integer;
begin
   if r-l > 0 then
      begin
        m:=(r+l)/2; mergesort(l, m); mergesort(m+1, r);
        for i := m downto l do b[i] := a[i];
        for j := m+1 to r do b[r+m+1-j] := a[j];
        for k := l to r do
          if b[i] < b[j] then
          begin
1             a[k] := b[i] ; i := i+1;
```

```
 2              if i = m+1 then break
              end
               else
               begin
3                a[k] := b[j]; j:= j-1
4                 if j = m then break
               end;
5           if i = m+1 then
6              for k1 = k to r do begin a[k1] := b[j]; j:= j-1 end
7           if j = m then
8              for k1 = k to r do begin a[k1] := a[i]; i:= i+1 end
         end;
end;
```

Explain the purpose of the statements 1,2, 3, 4, 5, 6, 7, 8.

c. State the time complexity of merge-sort.

7. Given the data file of 23 records with the following keys: 28, 3, 93, 10, 54, 65, 30, 90, 10, 69, 8, 22, 31, 5, 96, 40, 85, 9, 39, 13, 8, 77, 10.
Assume that one record fits in a block and memory buffer holds at most three page frames. During the merge stage, two page frames are used for input and one for output. Trace by hand the *external sorting* (external sort-merge) for the above data file.

8. Draw the binary search tree that results from inserting into an initially empty tree records with the keys: E A S Y Q U E S T I O N, and then delete Q.
In the average case, how many comparisons can a search in a binary search tree with N keys require?

9. Draw the binary search tree that results from inserting into an initially empty tree records with the keys: 5, 10, 30, 22, 15, 20 31. And then delete 10 from the tree.
In the worst case, how many comparisons can a search in a binary search tree with N keys require?

10. Given a recursive program to compute the height of a binary tree (the longest distance from the root to an external node) as follows.

```
function height(x: link): integer;
begin
  if x = nil then return -1
  else return max(height(x↑.l), height(x↑.r)) + 1
end;
```

Assume that the key operation in the above algorithm is checking whether the tree is empty. Analyze the time complexity of the algorithm.

11. Write a recursive program to compute the number of levels in a binary tree. (In particular, the algorithm should return 0 and 1 for the empty and single-node trees, respectively). Analyze the time complexity of the algorithm.

12. Give the recursive implementation of binary search. Analyze the time complexity of binary search.

13. Give the following algorithm:

```
Algorithm closest-pair(X[1..n])
// An array X[1..n] of n real numbers
Quicksort(X[1..n]);
for j := 2 to n do
  D[j-1]:= X[j] - X[j-1];
min := 1;
for j := 2 to n-1 do
  if D[j] < D[min] then min := j;
writeln(X[min], X[min+1])
```

a) State the meaning of the above algorithm.
b) State the complexity of the above algorithm.

Exercises
Chapter 3 (Decrease-and-Conquer)

1. Design a decrease-by-one algorithm for finding the position of the smallest element in an array of *n* real numbers. Determine the time efficiency of this algorithm and compare it with that of the brute-force algorithm for the same problem.

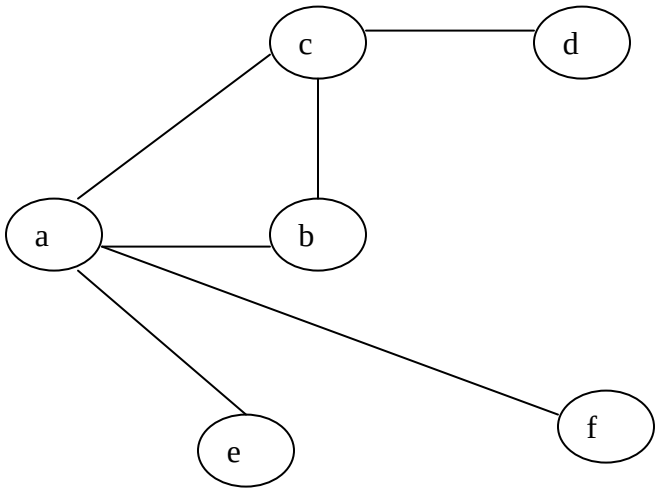
2. Given the insertion-sort algorithm as follows:

```
procedure insertion;
var i; j; v:integer;
begin
    a[0]:= intmin;
        for i:=2 to N do
        begin
                v:=a[i]; j:= i;
                while a[j-1]> v do
                begin a[j] := a[j-1]; j:= j-1 end;
                a[j]:=v;
        end;
end;
```

a) By hand, trace the action of the algorithm on the following list of keys:
44, 30, 50, 22, 60, 55, 77, 55
b) In the best case (the array is already in ascending order), how many comparisons and moves can the insertion-sort algorithm require for sorting an array of N keys?
c) In the worst case (the array is in reverse order), how many comparisons and moves can the insertion-sort algorithm require for sorting an array of N keys?

3. Is it possible to implement insertion sort for sorting linked lists? Will it has the same $O(n^2)$ efficiency as the array version?

4. Given an undirected graph as follows:

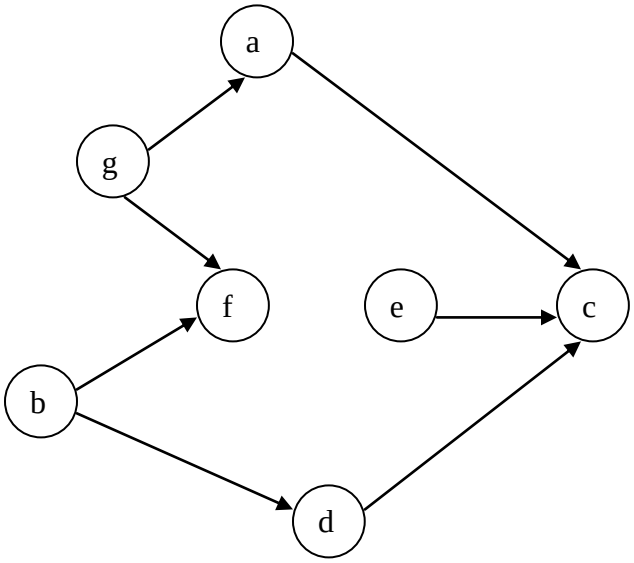


a. Construct the adjacency list representation of the above graph.
b. Construct the adjacency matrix that represents the graph.
c. By hand, trace step by step the status of the stack when you use it in a *depth-first-search* on the above graph (starting from vertice *a*). Then show the corresponding order in which the vertices might be processed during the *depth-first-search*.
d. State the time complexity of depth-first-search.
e. By hand, trace step by step the status of the queue when you use it in a *breadth-first- search* on the above graph (starting from vertice *a*). Then show the corresponding order in which the vertices might be processed during the *breadth-first-search*.

5. Modify the depth-first-search algorithm in order that it can be used to check whether a graph G has a cycle.

6. Explain how we can identify connected components of a graph by using
a. a depth-first-search
b. a breath-first-search

7. Given the directed graph



a. Construct an adjacency list representation for the above directed graph.
b. Using method 1, find two different topological sorts for the above directed graph.
c. Using method 2, find two different topological sorts.

8. a. Prove that a directed acyclic graph must have at least one source vertex.
b. How would you find a source vertex in a directed graph represented by its adjacency matrix? What is the time efficiency of this operation.
c. How would you find a source vertex in a directed graph represented by its adjacency linked lists? What is the time efficiency of this operation.
Note: Assume that we don't have *indegree* or *outdegree* information for each vertex

9. True/false: Topological sorting can be used to check if there is a cycle in a directed graph. Explain your answer.

10. Generate all permutations of {1,2,3,4} by tracing by hand the algorithm PERM given in the text.

Exercises
Chapter 4 (Transform-and-Conquer)

1. Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.
Algorithm Mindistance(A[1..n])

```
// Input: An array A[1..n] of numbers
//Output: The minimum distance d between two of its elements
dmin := max
for i := 1 to n-1 do
  for j:= i+1 to n do
     temp := |A[i] – A[j]|
     if temp < dmin then
        dmin = temp
return dmin
```

a. Analyse the worst case complexity of this brute-force algorithm.
b. Design a sorting-based algorithm for solving the above problem and analyze its complexity. Compare the complexity of this algorithm to the complexity of the brute-force algorithm.

2. Solve the following system by Gaussian elimination

$$x_1 + x_2 + x_3 = 2$$
$$2x_1 + x_2 + x_3 = 1$$
$$x_1 - x_2 + 3x_3 = 8$$

3. Write an algorithm for the back-substitution stage of Gaussian elimination and show that its running time is in $O(n^2)$.

4. a. By hand, build the heap (using top-down method) from the following list of keys read from the keyboard:
23, 7, 92, 6, 12, 24, 40, 44, 20, 21
b. By hand, build the heap (using bottom-up method) from the list of keys given in question a).
c. Is it always true that the bottom-up and top-down algorithms yield the same heap for the same input.

5. Design an algorithm for checking whether an array H[1..n] is a heap and analyze its complexity.

6. Given the heap-sort algorithm:

```
N:=0;
for k:= 1 to M do
 insert(a[k]); /* construct the heap */
for k:= M downto 1 do
 a[k]:= remove;
```

By hand, trace the action of heap-sort on the following list of keys:
44, 30, 50, 22, 60, 55, 77, 55
State the time complexity of heap-sort.

7. Rewrite the *upheap* procedure to build a minimum heap.

8. Given the algorithm that can sort an array of numbers by creating a binary search three from the array of numbers and then traverse the binary tree using in-order traversal.

```
procedure Tree-sort(T)
let T be an empty binary search tree
for i := 1 to n do
   TreeInsert(T, A[i]);
InOrder-Tree-Traversal(T);
```

Analyze the complexity of the algorithm.

9. Consider the following brute-force algorithm for evaluating a polynomial.

```
// P[0..n] is the array that stores the coefficients of a polynomial of degree n.
p := 0;
for i:= n downto 0 do
   power := 1;
   for j:= 1 to i do
      power := power*x;
   p := p + P[i]*power
return p;
```

Find the total number of multiplications and the number of additions made by this algorithm.

10. Apply Horner's algorithm to evaluate the polynomial

$P(x) = 2x^4 – x^3 + 3x^2 + x - 5$ at $x = 3$.

Is Horner's method more time efficient at the expense of being less space efficient than the brute-force algorithm?

11. Working modulo q = 11, how many spurious hits does the Rabin-Karp matcher encounter in the text T = "3141592653589793" when looking for the pattern P = "26"?

12. Given a text *T* which is a string of hexadecimal digits as follows:

T = "31ABC926DEF897A".

Given the pattern *P* = "BC". Working modulo *q* = 17 how many spurious hits does the Rabin-Karp matcher encounter in the text T when looking for the pattern P?

Exercises
Chapter 5 (Dynamic Programming & Greedy Algorithms)

1. Consider the problem of finding the *n*th Fibonacci number, as defined by the recurrence equation

F(0) = 0

F(1) = 1
F(n) = F(n-1) + F(n-2)

Develop a dynamic programming algorithm for finding the nth Fibonacci number.

2. a. Trace by hand the application of the dynamic programming algorithm to the following instance of the 0-1 knapsack problem:

item	weight	value
A	3	25
B	2	20
C	1	15
D	4	40
E	5	50

Assume that capacity W = 6

b. Modify the dynamic programming algorithm for 0-1 knapsack problem to take into account another constraint defined by an array *num*[1..N] which contains the number of available items of each type.

3. Given the following algorithm that computes the tables *m* and *s* when applying dynamic programming to solve the matrix chain multiplication problem.

```
procedure MATRIX-CHAIN-ORDER(p, m, s);
begin
   n:= length[p] - 1;
   for i: = 1 to n do
   m[i, i] := 0;
   for l:= 2 to n do   /* l: length of the chain */
     for i:= 1 to n – l + 1 do
     begin
        j:= i + l – 1;
        m[i, j]:= ∞;                    /* initialization   */
        for k:= i to j-1 do
        begin
           q:= m[i, k] + m[k + 1, j] + p_{i-1}p_k p_j;
           if q < m[i, j] then
           begin
              m[i, j]: = q;  s[i, j]: = k
           end
        end
     end
end
```

Compute the table *m* and *s* when we apply the above algorithm to solve the matrix chain multiplication problem
n (the number of matrices) = 4, $p_0 = 2$, $p_1 = 5$, $p_2 = 4$, $p_3 = 1$, $p_4 = 10$.

4. We can recursively define the number of combinations of m things out of n, denoted C(m,n), for $n \geq 1$ and $0 \leq m \leq n$, by

C(m, n) = 1 if m = 0 or m = n
C(m, n) = C(m, n-1) + C(m -1, n -1) if $0 < m < n$

a) Give a recursive function to compute C(m, n).
b) Give a dynamic programming algorithm to compute C(m, n). *Hint*: The algorithm builds a table generally known as Pascal's triangle.

5. Given a directed graph whose adjacency-matrix is as follows:

$$A = \begin{matrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

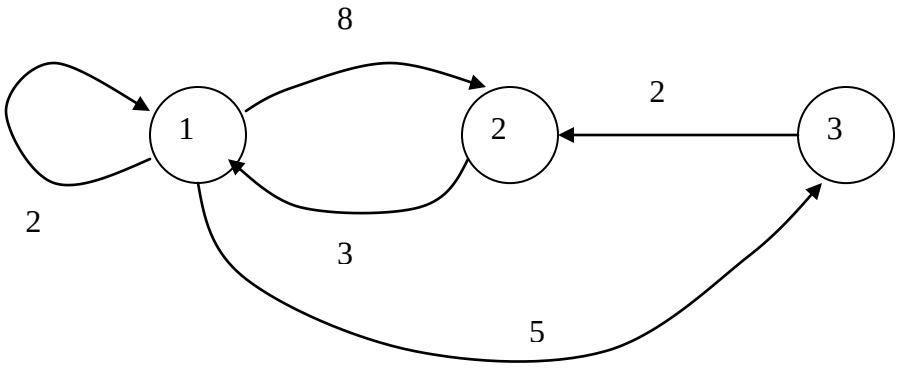
a. Show its adjacency-list representation.
b. Apply Warshall algorithm to find the transitive closure of the above directed graph (You have to show the matrices of 4 stages: y = 1, y=2, y = 3, y = 4).

6. Given a weighted, directed graph whose adjacency-matrix is as follows:

$$A = \begin{matrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{matrix}$$

Apply Floyd's algorithm to solve the all-pairs shortest path problem of the above directed graph (You have to show the matrices of 4 stages: y = 1, y=2, y = 3, y = 4).

7. Given the following directed graph.



Apply the modified Floyd algorithm which can recover the shortest path from one vertice to another. (You have to show the matrix *a* in 3 stages: y = 1, y =2, and y = 3 and the matrix *P* in the last stage, for the given graph.)

8. Given the following characters and their occurrence frequencies in the text file:

Character	Frequency
A	12
B	40
C	15
D	8
E	25

Find the Huffman codes for these above characters. What is the average code length?

9. Given the following greedy algorithm that solves the fractional knapsack problem (assume that the quantity of each item is 1):

```
procedure GREEDY_KNAPSACK(V, W, M, X, n);
/* V, W are the arrays contain the values and weights respectively of the n objects ordered so that
V_i/W_i ≥ V_{i+1}/W_{i+1}. M is the knapsack capacity and X is the solution vector */
var rc: real;
    i: integer;
begin
   for i:= 1 to n do  X[i]:= 0;
   rc := M ;  // rc = remaining knapsack capacity  //
   for i := 1 to n do
   begin
      if W[i] > rc  then exit;
      X[i] := 1;  rc := rc – W[i]
   end;
   if i ≤ n then X[i] := rc/W[i]
end
```

Improve the above algorithm in order that it can solve the the fractional knapsack problem in which the quantity of item *i* is *num[i]* (the array *num* keeps the information about the quantities of items).

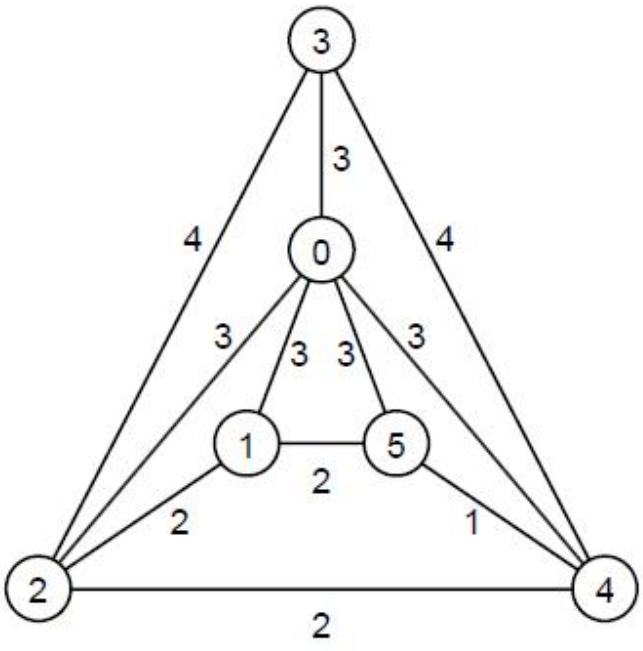
10. Consider the problem of making change for n cents using the least number of coins. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels and pennies. (quarter = 25 cents, dime = 10 cents, nickel = 5 cents, penny = 1 cent).

11. Given Prim's algorithm that constructs minimum spanning tree as follows.

```
 procedure MST-PRIM (G, w, r);
/* G = (V,E) is weighted graph with the weight function w, and r is an arbitrary root
vertex */
begin
   Q: = V[G];  /* Q is a priority queue  */
   for each u ∈ Q do key[u]: = ∞;
   key[r]: = 0; p[r]: = NIL;
   while Q is not empty do
   begin
      u: = EXTRACT-MIN(Q);
      for each v ∈ Q and w(u, v) < key[v] then
      / * update the key field of vertice v */
      begin
         p[v] := u;  key[v]: = w(u, v)
      end
   end
end;
```

Note: For each vertex *v*, *key[v]* is the minimum weight of any edge connecting *v* to a vertex in the growing minimum spanning tree. By convention, *key[v]* = ∞ if there is no such edge. The field *p[v]* names the "parent" of *v* in the growing minimum spanning tree.

Given the following weighted graph



a. Trace the actions of finding a minimum spanning tree, using Prim's algorithm. (Assume that 3 is the starting vertex).

b. If heap is used to implement the priority queue in the Prim's algorithm, analyze the worst-case complexity of the algorithm (assume that *adjacency list* representation is used for the undirected graph).

c. If array is used to implement the priority queue in the Prim's algorithm, analyze the worst-case complexity of the algorithm (assume that *adjacency list* representation is used for the undirected graph).

12. Given Dijkstra's algorithm that finds a shortest path from a given source vertex *s* in a weighted directed graph to every vertex *v* in the graph.

```
procedure dijkstra(G, w, s);
/* G is a graph, w is a weight function and s is the source node */
begin
   for each vertex v ∈ V[G] do /* initialization */
   begin  d[v]: = ∞; p[s]: = NIL end;
   d[s]: = 0; S: = ∅; Q: = V[G]
   while Q is not empty do
   begin
      u: = EXTRACT-MIN (Q); S: = S ∪ {u};
      for each vertex v ∈ Adj [u] do /* relaxation */
        if d[v] > d [u] + w(u, v) then
        begin d[v]: = d[u] + w(u, v); p[v]: = u end
```

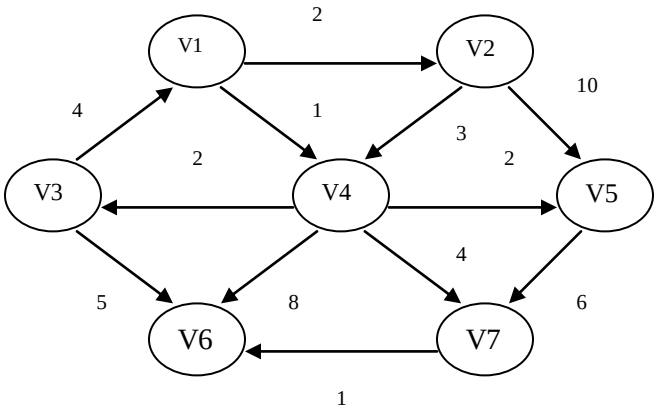
end
end

Note: for all vertice *v* in the graph, we have

d[v] = min (shortest-path-estimate from *s* to *v*)

and *p[v]* names the "parent" of *v* in the path.

Given the following weighted directed graph:



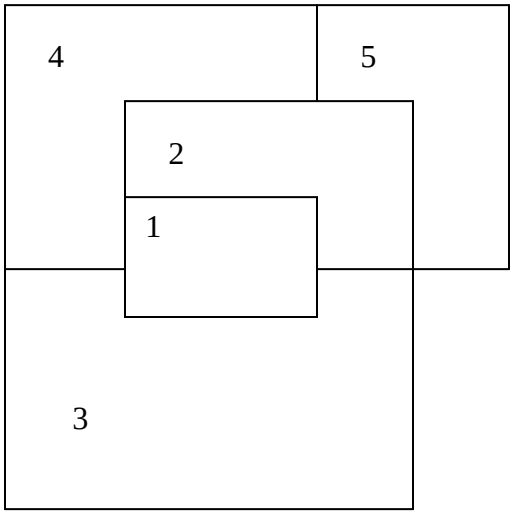
a. Trace by hand the working of the Dijkstra algorithm to solve the single-source shortest path problem for the above graph (the initial vertex is v_1). State all the arrays *p, d* at each iteration of the algorithm.

b. If heap is used to implement the priority queue in the Dijkstra's algorithm, analyze the worst-case complexity of the algorithm (assume that *adjacency list* representation is used for the directed graph).

c. If array is used to implement the priority queue in the Dijkstra's algorithm, analyze the worst-case complexity of the algorithm (assume that *adjacency list* representation is used for the directed graph).

13. Analyze the time complexity of the greedy algorithm for the graph coloring problem in the case the graph G = (V,E) is a complete graph.

14. Given the following map, color the regions in the map in such a way that no two adjacent regions have the same color. Transform the map coloring problem to a graph coloring problem and apply the greedy algorithm to solve it.



Exercises
Chapter 6 (Backtracking Algorithms)

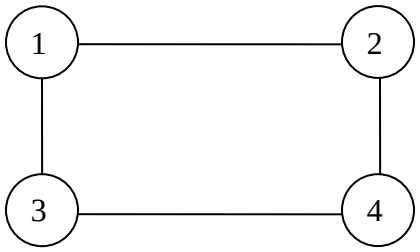
1. A *coloring* of a graph is an assignment of a color to each vertex of the graph so that no two vertices connected by an edge have the same color. We are interested in determining **all** the different ways in which a given graph may be colored using at most *m* colors. Assume that the graph is represented by adjacency-matrix GRAHP[1..n,1..n]. The colors will be represented by the integers 1, 2,…,*m* and the solutions will be given by the *n*-tuple <X[1], X[2],…, X[n]> where X[i] is the color of node *i*. The algorithm is given as follows in a form of two procedures.

```
procedure MCOLORING(k)
/* this procedure is to assign color the vertex k . It is a backtracking procedure */
begin
  int k;
  repeat  // generate all legal assignments for X(k)
     ASSIGN_COLOR(k);  // assign to X(k) a legal color
      if X(k) = 0 then exit;   // no new color possible
      if k = n then print(X)
      else
         MCOLORING(k+1);
   until false;
end
procedure ASSIGN_COLOR(k)
begin
   int j,k;
   repeat
```

```
    X(k) := (X(k) + 1) mod (m+1);   // next color
    if X(k) = 0 then return;   // all colors have been exhausted
    for j:= 1 to n do
      if GRAPH[k, j] and X(k) = X(j) then
        exit;
    if j = n+1 then return;
  until false;
end;
```

Procedure MCOLORING is begun by first assignning the graph to its adjacency matrix, setting the array X to zero, and then invoking the statement MCOLORING(1).

a. Explain how the above backtracking algorithm can solve the m-colorability optimization problem.
b. Draw the state space tree for MCOLORING when n = 3 and m = 3.
c. Analyze the time complexity of the above algorithm.
d. Find all the solutions when applying MCOLORING for the following graph and with at most three colors. (Hint: Draw the state space tree.)



2. Given the graph coloring problem with the graph given in Figure 3. In Figure 3, the set of legal colors that can be assigned to each vertex is given inside the vertex itself.

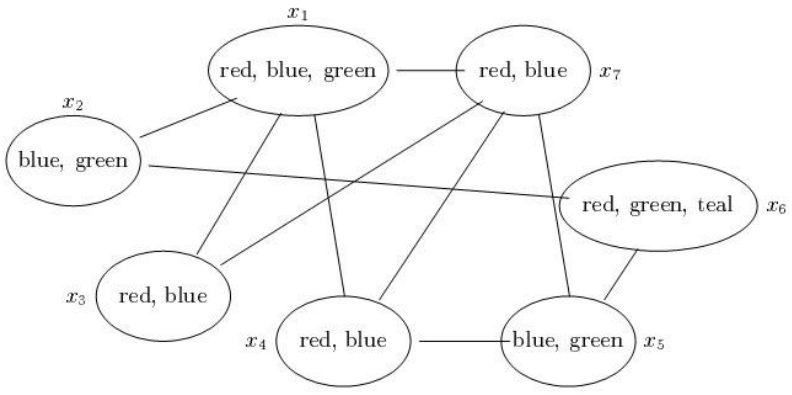


Figure 3: A modified coloring problem.

a. Draw the search tree that illustrates the solving of the graph coloring problem using backtracking. Assume that the vertex ordering for color assignment is as follows: x_1, x_7, x_4, x_5, x_6, x_3, x_2. Notice that the search tree should show the exhaustive search of all the solutions.
b. Draw the search tree in the case that the vertex ordering for color assignment is as follows: x_1, x_2, x_3, x_4, x_5, x_6, x_7.

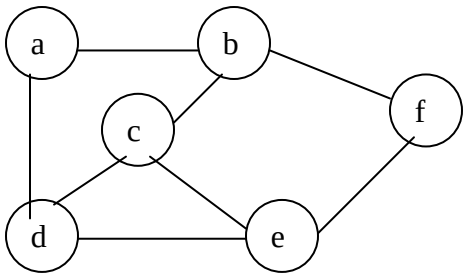
3. A complete graph is a graph in which there exists at least one edge between any pair of vertices.
Assume that a complete undirected graph is represented by adjacency matrix. A *simple path* between two vertices in the graph is a path on which each vertex is visited only once.
Explain how the following backtracking algorithm can generate all the simple paths starting from a given vertex in the graph (assume that the starting vertex is with the index 1).

```
procedure visit(k:integer);
var t: integer;
begin
  id:= id + 1; val[k] := id;
  for t:= 1 to V do        /* V laø soá ñænh trong ñoà thò */
    if a[k,t] = 1 then
      if val[t]=0 then visit(t);
  id:= id – 1; val[k]:= 0
end
```

The above procedure is invoked from the main program as follows.

```
for i:= 1 to V do val[i]:= 0;
id:= 0; visit(1).
```

4. *Hamiltonian cycle* in a graph is a simple path that starts from a vertex, visits each vertex in the graph only once and then returns back to the starting vertex. Given the following graph.



<a, b, f, e, c, d, a> is an example of a Hamilton cycle in the above graph.

Draw a search tree that shows the process of finding a Hamilton cycle in the above graph with the starting vertex *a*, using a backtracking algorithm.

5. Suppose the first solution for the 4 Queens problem is as follows:

1			X	
2	X			
3				X
4		X		
	1	2	3	4

Draw a search tree that shows the process of finding that solution using a backtracking algorithm.

6. Let G = (V, E) be a connected graph with *n* vertices. Develop a backtracking algorithm that can generate all Hamiltonian cycles in G starting from a given starting vertex.

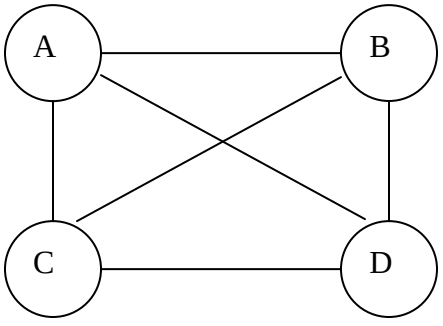
Hint: Modify the DFS algorithm into the algorithm that can generate all the simple paths originating from the starting vertex and then modify this algorithm into the algorithm for generating Hamilton cycles.

7. Is there is any relationship between backtracking and branch-and-bound algorithm design strategies? Explain your answer.

8. Given a complete and weighted graph consisting of 5 vertices A, B, C, D, E. The weights on the edges in the graph are as follows: AB = 3, AC = 4, AD = 2, AE = 7, BC = 4, BD = 6, BE = 3, CD = 5, CE = 8, DE = 6.

Assume that vertex A denotes the starting city of a Traveling Salesman Problem (TSP) with the above weighted graph. Solve the TSP using *branch-and-bound* algorithm. Give the optimal solution and the total cost of this tour.

9. Given a complete and weighted graph consisting of 4 vertices A, B, C, D as in the following figure.



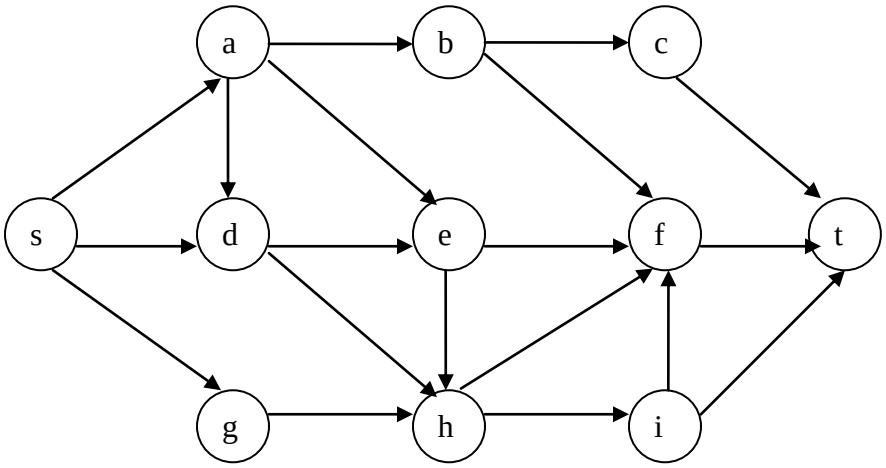
The weights on the edges in the graph are as follows: AB = 2, AC = 5, AD = 7, BC = 8, BD = 3, CD = 1,.

Assume that vertex A denotes the starting city of a Traveling Salesman Problem (TSP) with the above weighted graph. Solve the TSP using *branch-and-bound* algorithm. Give the optimal solution and the total cost of this tour.

10. Given the directed and weighted graph as in the following figure.

The weights on the edges are: sa = 2, ab = 7, bc = 1, ad = 3, ae = 2, bf = 2, ct = 4, sd = 3, de = 9, ef = 3, ft = 4, sg = 4, dh = 2, eh = 1, hf = 5, if = 2, it = 2, gh = 2, hi = 1.

Apply the branch-and-bound algorithm to find the shortest path from the vertex *s* to the vertex *t*. Assume that the *lower bound* of each partial solution is the sum of distances in the path from the source vertex to the current vertex.



Review Questions
Chapter 7 (NP-Completeness)

1. Use a known NP-complete problem, prove that the following problem is also NP-complete:

LONGEST PATH
INSTANCE: Graph G = (V,E), and a positive integer $k \leq |V|$.
QUESTION: whether G has a simple path with length $\geq k$ or not.

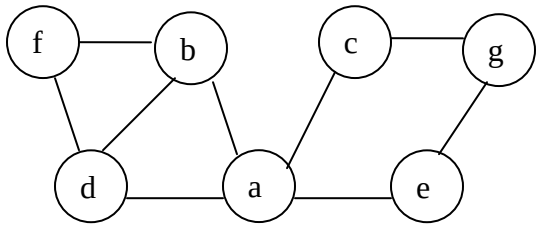
2. Can combining backtracking and heuristics be a method to solve a NP-complete problem? Explain your answer.

3. Can greedy algorithm be a method to solve an NP-complete problem? Explain your answer.

4. What are metaheuristics? Can we use a metaheuristics to solve a NP-complete problem?

Exercises
Chapter 8 (Approximation Algorithms)

1. Consider the graph



Apply the approximate algorithm for vertex-covering problems to find the a vertex cover of minimum size for the above instance.

2. Given an instance {X, F} of the set covering problem, where X consists of the 11 elements $x_1, x_2,..,x_{11}$ and F = { S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11} where
S1 = {x_1, x_2, x_3, x_4}
S2 = { x_1, x_2, x_3, x_4, x_5}
S3 = { $x_1, x_2, x_3, x_4, x_5, x_6$}
S4 = { x_1, x_3, x_4, x_6, x_7}
S5 = { $x_2, x_3, x_5, x_6, x_8, x_9$}
S6 = { $x_3, x_4, x_5, x_6, x_7, x_8$}
S7 = { x_4, x_6, x_7, x_8 }
S8 = { $x_5, x_6, x_7, x_8, x_9, x_{10}$}
S9 = { $x_5, x_8, x_9, x_{10}, x_{11}$}
S10 = { x_8, x_9, x_{10}, x_{11}}
S11 = { x_9, x_{10}, x_{11}}
Apply the approximate algorithm for set-covering problems to solve the above instance.

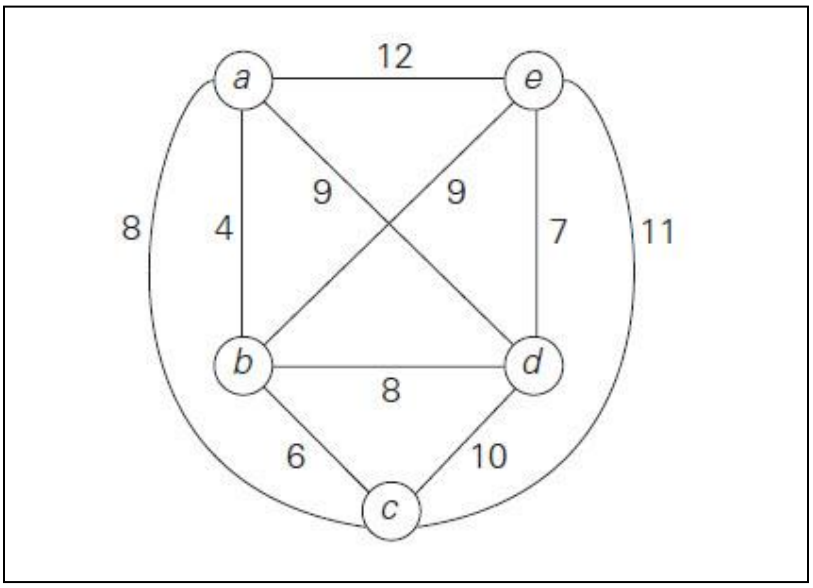
3. Given an instance {X, F} of the set covering problem, where X consists of the 7 elements X = {a, b, c, d, e, f, g} and F = { S1, S2, S3, S4, S5, S6, S7} where
S1 = {a, b, f, g}
S2 = {a, b, g}
S3 = { a, b, c}
S4 = { e, f, g}
S5 = { f, g}
S6 = { d, f}

S7 = { d }
Apply the approximate algorithm for set-covering problems to solve the above instance.

4. Given a complete and weighted graph consisting of 5 vertices A, B, C, D, E. The weights on the edges in the graph are as follows: AB = 3, AC = 4, AD = 2, AE = 7, BC = 4, BD = 6, BE = 3, CD = 5, CE = 8, DE = 6.

Assume that vertex A denotes the starting city of a Traveling Salesman Problem (TSP) with the above weighted graph. Solve the TSP using approximation algorithm. Give the near optimal solution and the total cost of this tour.

5. Given a complete and weighted graph as in the following figure.



Assume that vertex *a* denotes the starting city of a Traveling Salesman Problem (TSP) with the above weighted graph. Solve the TSP using approximation algorithm. Give the near optimal solution and the total cost of this tour.

6. Given the problem of scheduling independent tasks which is defined as follows:

The number of processors $m = 3$, the set of 6 tasks with $(t_1, t_2, t_3, t_4, t_5, t_6) = (2, 5, 8, 1, 5, 1)$.

Apply the *LPT* rule to solve the above scheduling problem.

7. Given the bin packing problem in which the capacity of each bin is 13, the number of objects is 8 and the capacity of each object given the array $L = (7, 9, 7, 1, 6, 2, 4, 3)$.

a) If the heuristic First Fit is used to solve the problem, what is the result?

b) If the object with capacity 1 is removed, and the heuristic First Fit is used, what will be the result?

8. Given the bin packing problem in which the capacity of each bin is 1, the number of objects is 7 and the capacity of each object given the array L = (0.2, 0.6, 0.5, 0.2, 0.8, 0.3, 0.2).

a) If the heuristic First Fit is used to solve the problem, what is the result?

b) If the heuristic First Fit Decreasing is used to solve the problem, what is the result?