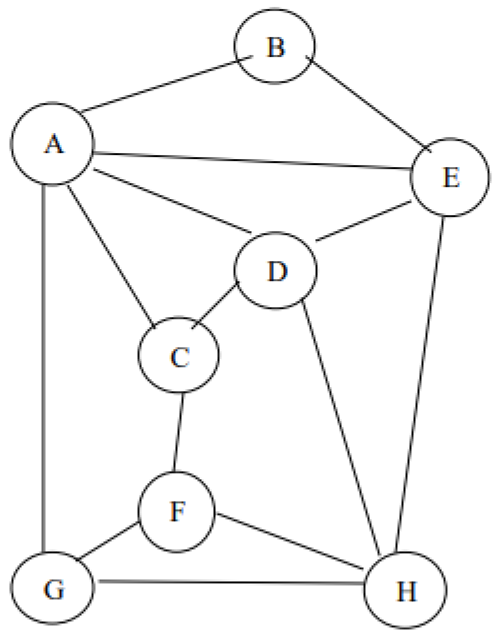
1. Consider the following graph with edge weights listed in the table.



AB	2
AC	3
AD	5
AE	1
AG	2
BE	3
CD	6
CF	3
DE	7
DH	2
EH	4
FG	8
FH	1
GH	5

(a) What is the cost of its minimum spanning tree (MST)?

$3+5+1+2+2+4+1=18$

(b) How many minimum spanning trees does it have?

2

(c) Suppose Kruskal's algorithm is run on the graph.

1. In what order are the edges added to the MST (in case of ties add the edge which is lexicographically first, that is, comes first in the table)?
2. For each edge in this sequence, give a cut that justifies its addition.
3. Show how the disjoint-sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is not used.

1) Edges added in this order: AE, FH, AB, AG, AC, CF, AD

2) Edges AE: S={A} V-S={E, B, C, D, F, G, H}

Edges FH: S={A, C, G, B, F} V-S={H, D, E}

Edges AB: S={G, H, F, C, D, A} V-S={B, E}

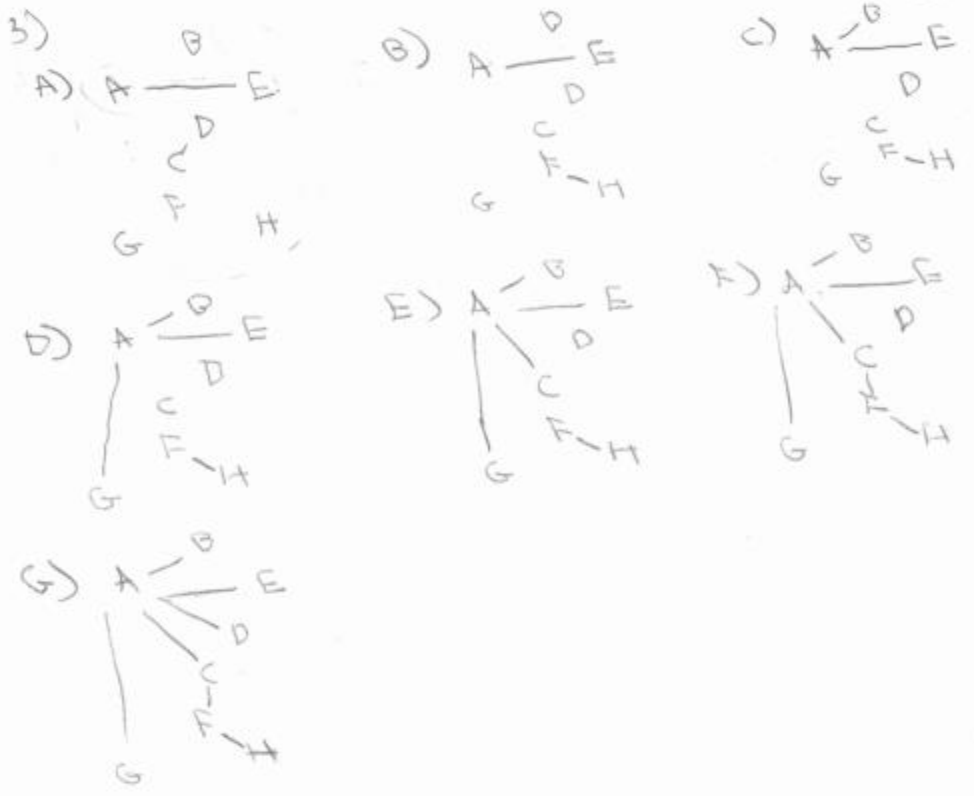
Edges AG: S={B, F, C, D, A} V-S={G, H, F}

Edges AC: S={B, G, E, D, A} V-S={C, F, G}

Edges AD: S={B, G, C, F, H, E} V-S={D}

Edge CF: S={A, B, E, D, G} V-S={F, H}

3)



(d) Suppose Prim's algorithm is run on the graph. Whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node A). In what order are the edges added to the MST?

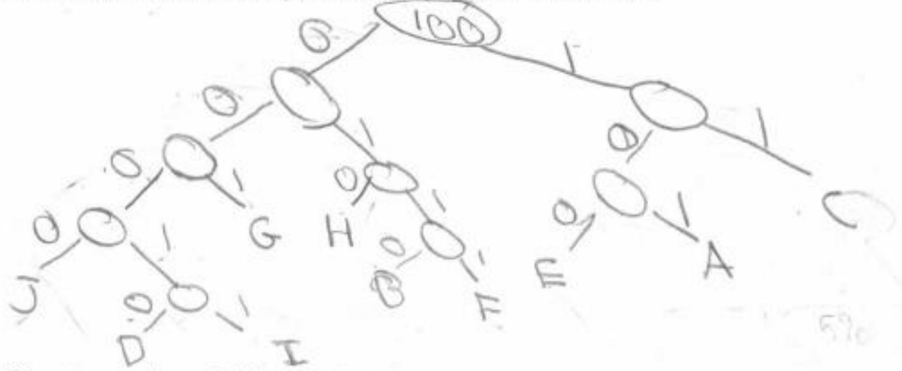
phabetic ordering (e.g., start from node A). In what

AE, AB, AG, AC, CF, AD

The following table gives the frequency of t

2. The following table gives the frequencies of the symbols in a particular document.

a	10%
b	7%
c	20%
d	5%
e	12%
f	6%
g	15%
h	14%
i	3%
j	8%



(a) What is the optimum Huffman encoding of this alphabet?

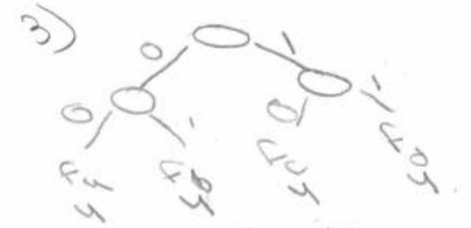
a=101 d=00010 g=001 j=0000
b=0110 e=100 h=010
c=11 f=0111 i=00011

(b) If this encoding is applied to a file consisting of 1,000,000 characters with the given frequencies, what is the length of the encoded file in bits?

1,000,000 (10·3 + 7·4 + 20·2 + 5·5 + 12·3 + 6·4 + 15·3 + 14·3 + 3·5 + 8·4) = 317600000 bits

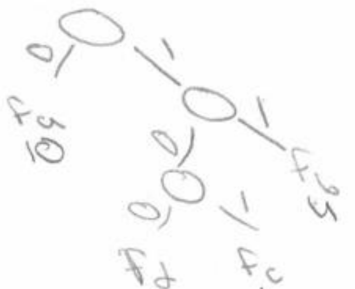
3. We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c, d\}$ with frequencies f_a, f_b, f_c, f_d. Assume that $f_a \geq f_b \geq f_c \geq f_d$. List all possible different coding trees that could be obtained (consider two trees to be the same if one tree can be obtained from the other one by changing the order of children of nodes and/or exchanging 0/1 labels on edges connecting nodes to their children) . For each coding tree give an example of frequencies (f_a, f_b, f_c, f_d) that would yield the coding tree.

3)



fa=00
fb=01
fc=10
fd=11

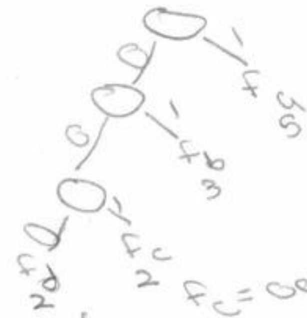
-or-



fa=0
fb=11
fd=100
fc=101

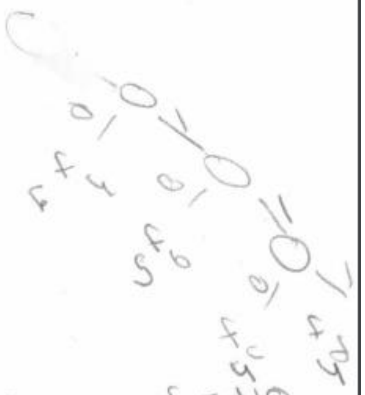
key

f letter
frequency



fa = 1
fb = 01
fc = 001
fd = 000

-or-



fa=0
fb=10
fc=110
fd=111

4. Use dynamic programming algorithm to find edit distance between strings "CGAACCG" and "CATCA". Show your work - draw a table and fill it in. (The algorithm is described in Section 6.3)

4) y = CGAACCG x = CATCA

	y	0	1	2	3	4	5
x			C	A	T	C	A
0		0	1	2	3	4	5
1	C	1	0	1	2	3	4
2	G	2	1	1	2	3	4
3	A	3	2	1	2	3	3
4	A	4	3	2	2	3	3
5	C	5	4	3	3	2	3
6	C	6	5	4	4	3	3
7	G	7	6	5	5	4	4

Edit distance 4.

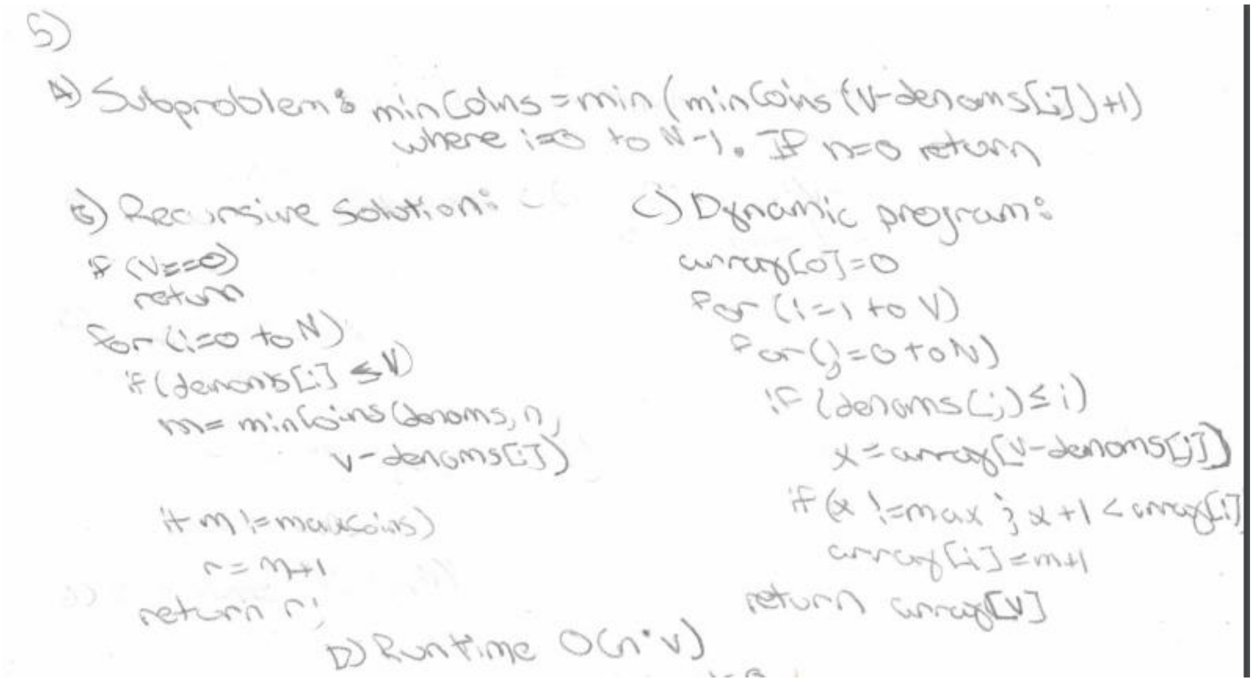
5. Assume that you have an unlimited supply of coins with integer denominations d_1, d_2, d_3, …, d_N. We wish to make change for a value V; that is, we wish to find a set of coins whose total value is V. Assume that d_1=1, so it is always possible to make change for any value V. We want to make change for V with as few coins as possible. Design a dynamic programming algorithm that given an integer V computes the smallest number of coins necessary to make change for V.

(a) Define a suitable subproblem.

(b) Give a recursive solution to the subproblem.

(c) Give a pseudocode for a dynamic programming algorithm that solves the problem.

(d) Analyze the running time of your algorithm.



5)

A) Subproblem: minCoins = min(minCoins(V-denoms[i])+1)
where i=0 to N-1. If n=0 return

B) Recursive Solution:

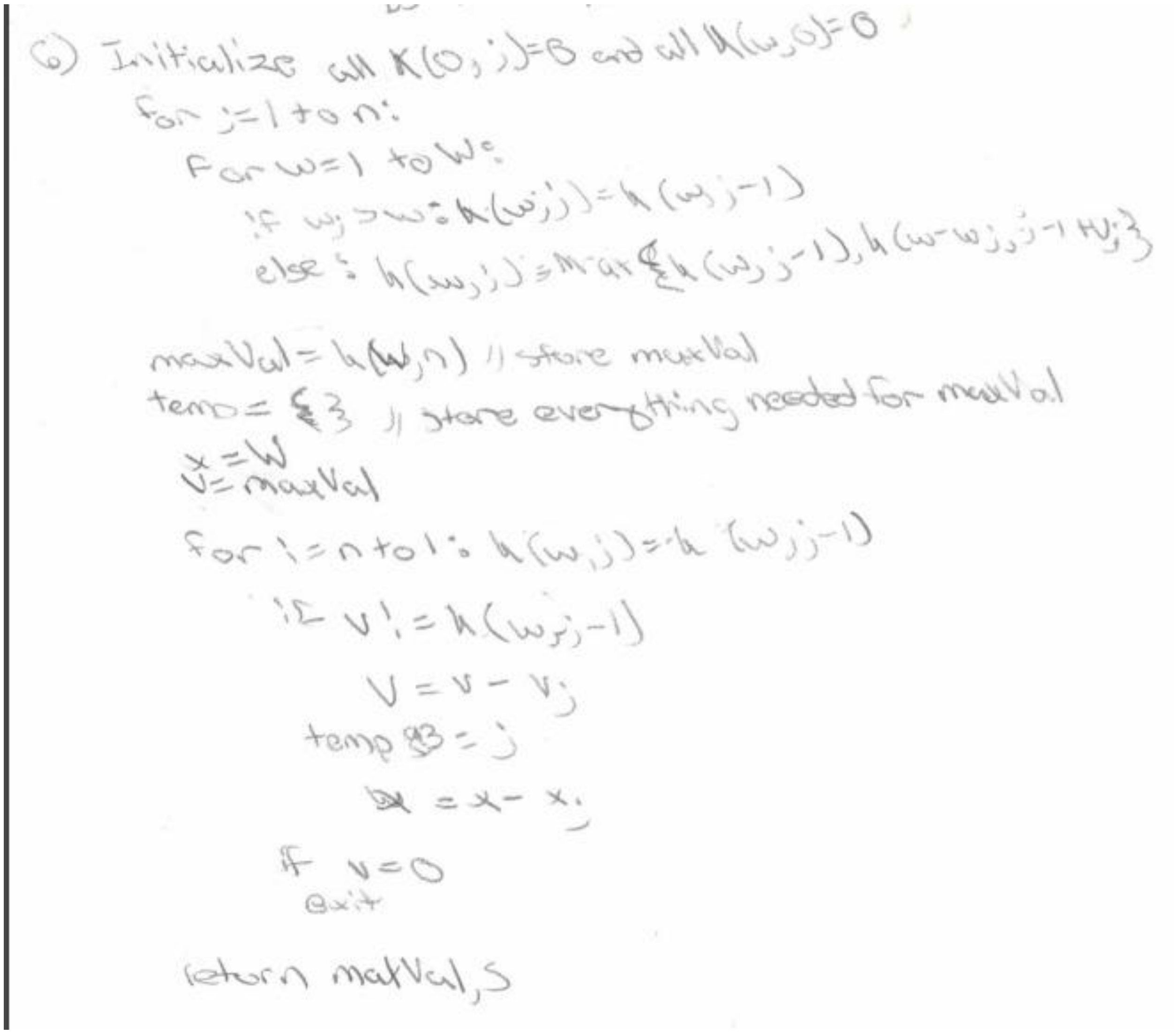
```
if (V==0)
  return
for (i=0 to N)
  if (denoms[i] ≤ V)
    m= minCoins(denoms, n,
            v-denoms[i])

    if m != maxCoins)
      r = m+1
  return r;
```

C) Dynamic program:

```
array[0]=0
for (i=1 to V)
  for(j=0 to N)
    if (denoms(j)≤i)
      x=array[V-denoms[j]]
    if (x !=max ; x+1 < array[i]
      array[i]=m+1
return array[V]
```

D) Runtime O(n*v)

6. Consider the code for the Knapsack without repetition problem at the end of Section 6.4 (page 168) of the textbook. The code returns the maximum value achievable using a knapsack of capacity W and items 1, …, n. Modify the code in such a way that it outputs not just the maximum value but the subset of objects that achieves the maximum value, subject to the weight constraint. You need to actually construct the subset of objects in the knapsack. Provide a pseudocode that does it.



```
6) Initialize all $K(0,j)=0$ and all $k(w,0)=0$
   for $j=1$ to $n$:
      for $w=1$ to $W$:
         if $w_j > w$: $k(w,j) = k(w,j-1)$
         else: $k(w,j) = \max\{k(w,j-1), k(w-w_j, j-1)+v_j\}$

   maxVal = $k(W,n)$ // store maxVal
   temp = {} // store everything needed for maxVal
    $x = W$
    $v$ = maxVal
    for $i=n$ to $1$: $k(w,j) = k(w,j-1)$
       if $v \neq k(w,j-1)$
            $V = v - v_j$
          temp{} = $j$
            $x = x - x_j$
       if $v=0$
        exit

   return maxVal, S
```