Computer Science 3711
Winter 2005
Final Exam Answers

1. (14 marks)

a) (8 marks) Circle the letters associated with the appropriate answers in the following multiple choice questions. Note that some of these questions may have more than one answer whose letter needs to be circled.

i) (4 marks) Which of the following is true of the function $T(n) = n^2 2^{2\log_2 n}$?

a) $T(n)$ is $O(n2^{\log_2 n^6})$ b) $T(n)$ is $\Omega(2^{3\log_2 n})$

c) $T(n)$ is $\Omega(n^5)$ d) $T(n)$ is $\Theta(2^{30}n^4)$

Answer: Parts (a), (b), and (d) are circled.

ii) (4 marks) Which of the following is true of the function $T(G) = |V||E|\log_2 |E|^4$ relative to undirected graph $G = (V, E)$?

a) $T(G)$ is $\Theta(|V|^2 \log_2 |V|^4)$ for complete graphs

b) $T(G)$ is $\Omega(|E|^2 \log_2 |V|^6)$ for complete graphs

c) $T(G)$ is $O(|V|^2 \log_2 |V|^2)$ for acyclic graphs

d) $T(G)$ is $\Omega(|E|^3 \log_2 |E|)$ for acyclic graphs

Answer: Part (c) is circled.

b) (6 marks) Give a recurrence for the worst-case time complexity of the following recursive algorithm:

```
procedure FUNKY-REC(m, n)
    mult = m
    if (n < 3)
        return(m)
    else if (n > 10)
        for (i = 1; i <= n * log2(m); i++)
            for (j = 1; j <= n; j++)
                mult = FUNKY-ITR(i * j) * FUNKY-REC(m, n/4)
    else if (n < 20)
        for i = 1 to log2(m) do
            mult = FUNKY-ITR(i * i * m) * FUNKY-REC(m, n/5)
    else if (n > 5)
        for i = 1 to n * m do
            mult = FUNKY-ITR(m) * FUNKY-REC(m, n/2)
    else
        for (i = 1; i <= n * n * n * n; i++)
            for (j = 1; j <= m; j++)
            mult = mult * FUNKY-ITR(i) * FUNKY-REC(m, n/3)
    return(mult)
```

Assume that procedure `FUNKY-ITR(x)` runs in $O(x^3)$ time.

$$T(m,n) = \begin{cases} O(1) & \text{if n} < 3 \\ O(m^3(\log_2 m)^7) & \text{if!} 3 \leq \text{n} \leq 10 \\ n^2 \log_2 mT(m, n/.4) + O(n^6(\log_2 m)^3) & \text{otherwise} \end{cases}$$

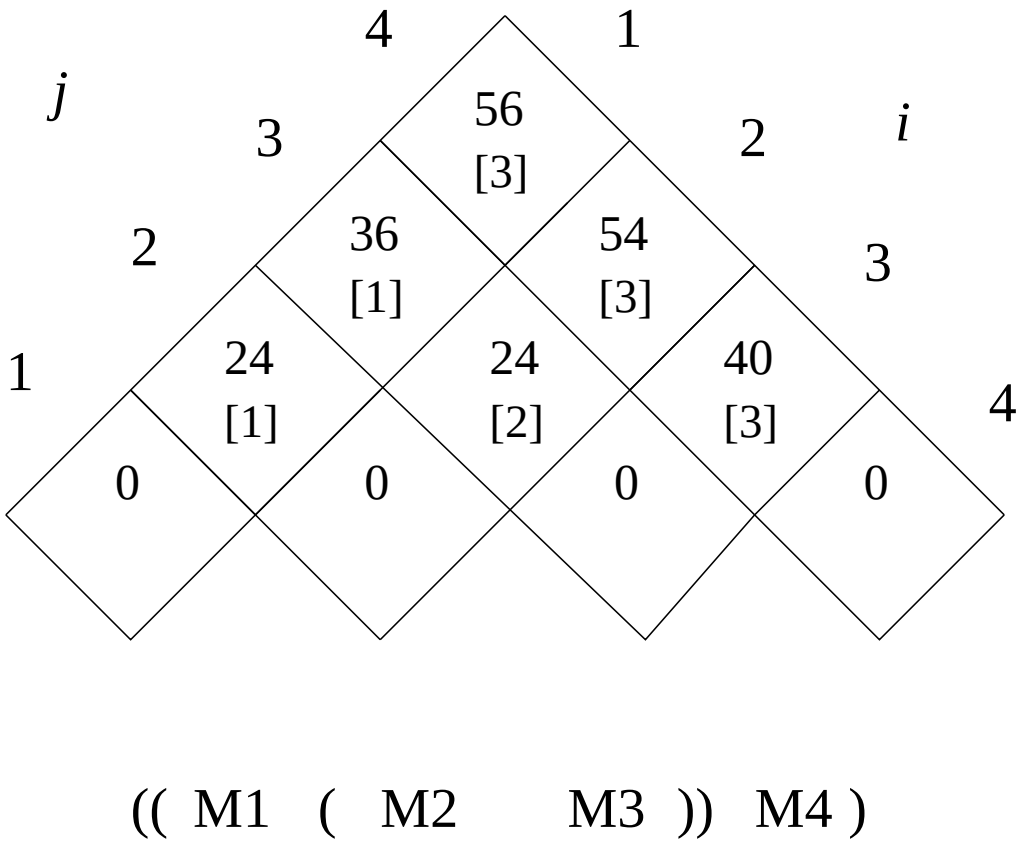
2. (26 marks)

a) (14 marks) Consider the the dynamic programming algorithm for the Matrix Chain Parenthesization problem defined by the recurrence

$$m[i,j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} m[i,k] + m[k+1,j] + p_{i-1}p_j p_k & i < j \end{cases}$$

where the input is a matrix-chain dimension list $p = \langle p_0, p_1, \ldots, p_n \rangle$ specifying a matrix-chain $M_1 M_2 \ldots M_n$ with dimensions $(p_0 \times p_1), (p_1 \times p_2), \ldots, (p_{n-1} \times p_n)$. Given matrix-chain dimension list $p = \langle 2, 3, 4, 2, 5 \rangle$, compute and output an optimal parenthesization of the corresponding matrix-chain relative to the recurrence above (that is, fill in the given dynamic programming table (including one backpointer per table-cell), show the backpointer "path" that gives an optimal

parenthesization, and give the parenthesization associated with that backpointer-"path").



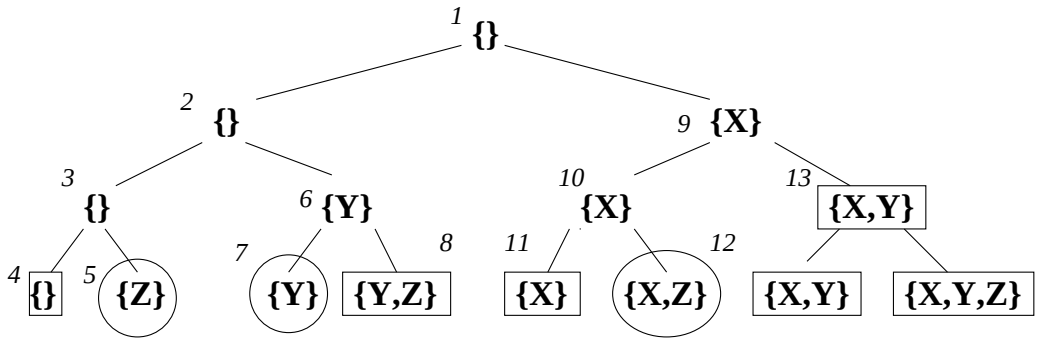
b) (12 marks) Consider the following algorithm:

```
DFS-V(i, sol, U, BL, BU, val)
    if (SIZE(sol) > BU)
        print("pruned: solution too big for knapsack")
    else if (i == n)
        if (SIZE(sol) < BL)
            print("pruned: solution too small for knapsack")
        else if (VALUE(sol) < val)
            print("pruned: solution too cheap for knapsack")
        else
            print(sol)
    else
        DFS-V(i + 1, sol, U, BL, BU, val)
        DFS-V(i + 1, UNION-COPY(sol, U[i]), U, BL, BU, val)
```

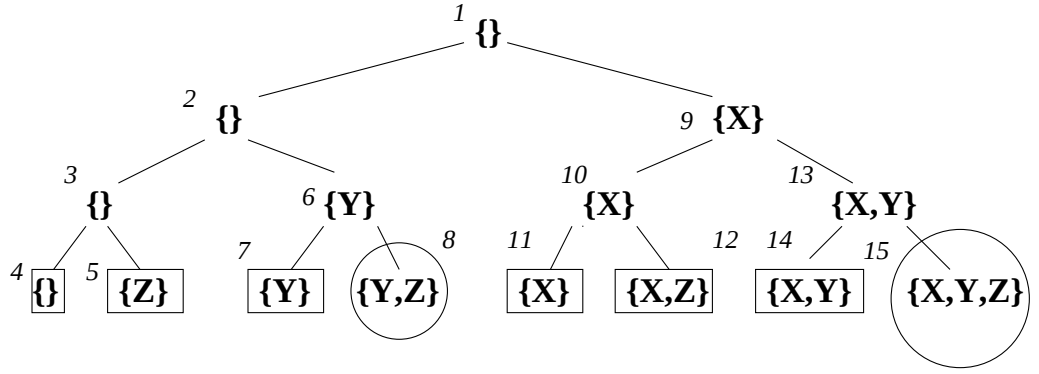
The problem solved here is a variant of 0/1 KNAPSACK that prints all knapsack-loads of value at least *val* whose sizes are between BU and BL inclusive. In this

algorithm, U is the set of items, n is the number of items in U, BL and BU are the lower and upper knapsack size bounds, sol is a subset of U, SIZE(sol) returns the sum of the sizes of the items in sol, VALUE(sol) returns the sum of the values of the items in sol, and UNION-COPY($sol, U[i]$) returns a copy of sol to which the ith item in U has been added. To answer this question, sketch the implicit tree of solution-nodes generated by the algorithm above (marking leaf-nodes which are printed by a circle and nodes that are pruned with a square) with the call DFS-V($0, sol, U, BL, BU, val$) when $U = \{X, Y, Z\}$ such that $size(X) = 2$, $size(Y) = 3$, $size(Z) = 2$, $value(X) = 1$, $value(Y) = 2$, and $value(Z) = 2$, sol is the empty set, and BL, BU, and val have the following values:

i) (6 marks) $BL = 2$, $BU = 4$, and $val = 2$:

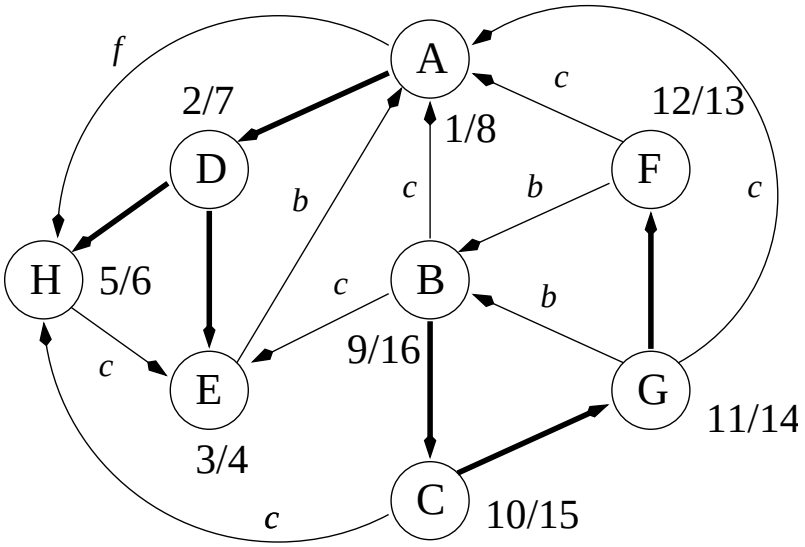


ii) (6 marks) $BL = 4$, $BU = 7$, and $val = 4$:



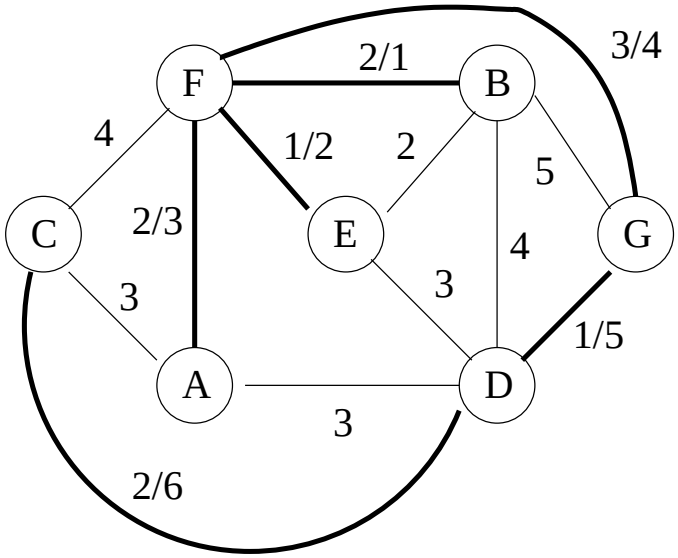
3. (24 marks)

a) (8 marks) Consider the following directed graph:

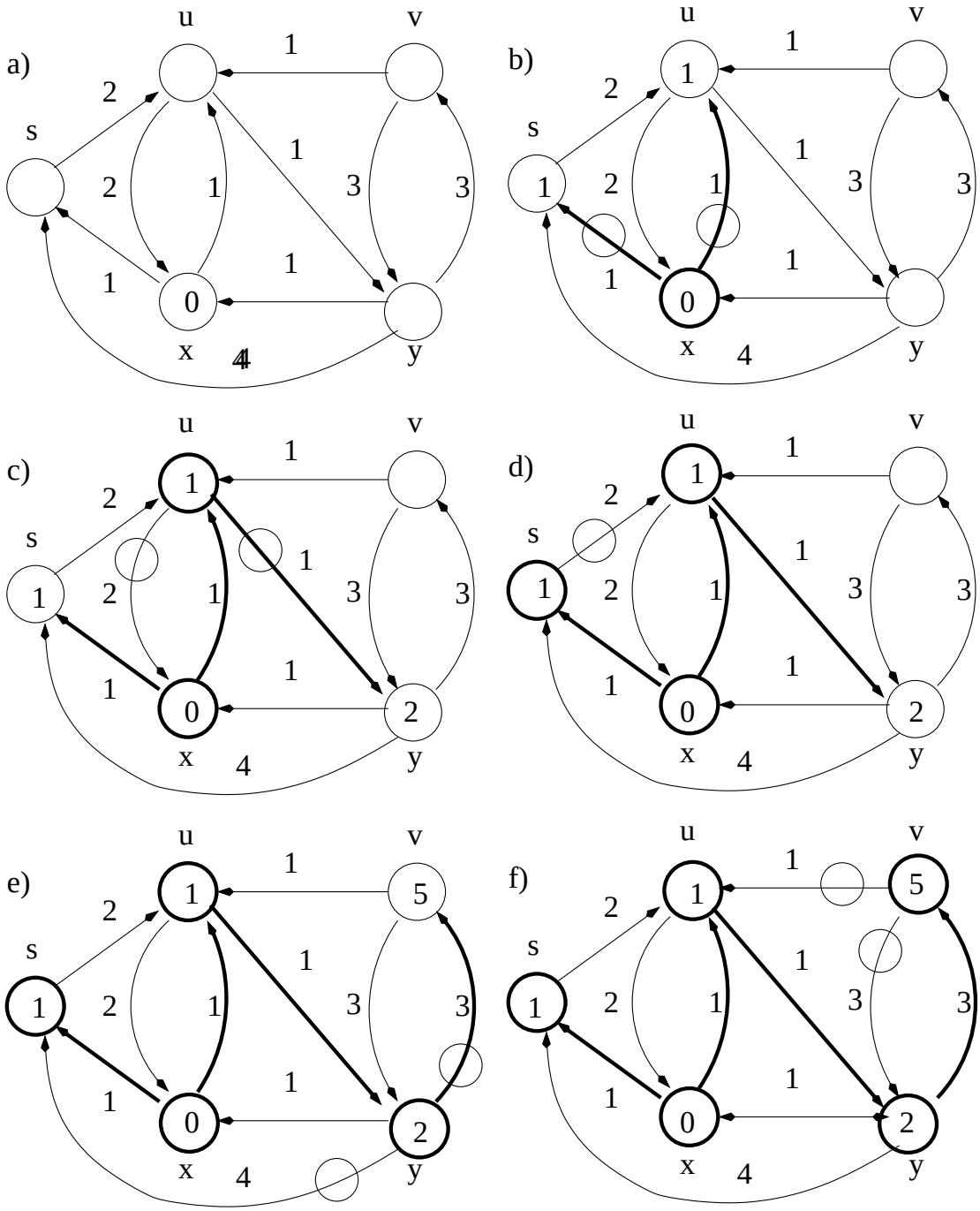


Give the graph at the end of the execution of the DFS algorithm, with the d- and f-values of all vertices as well as the types of all edges clearly marked. Assume that the algorithm considers vertices in alphabetical order and that each adjacency list is ordered alphabetically.

b) (6 marks) Show how Prim's minimum spanning tree algorithm works on the undirected edge-weighted graph below relative to root-vertex $r = B$. Mark all tree-edges and the order in which each tree-edge was added.



c) (10 marks) Run Dijkstra's algorithm on the directed graph below using vertex x as the source vertex. Show the d and π values and the vertices in set S after each iteration of the **while** loop by filling in the appropriate values on the spare copies of the graph given in this table.



4. (36 marks)

a) (18 marks) Consider the following problem: You have been asked to help schedule the stunts in movie scripts for a major Hollywood studio. Each movie script consists of a sequence of scenes, where each scene has an associated set-type (Urban / Country / Outer Space / Underwater) and a sequence of stunt-activities (Fall / Crash / Explosion). For example, Scene 3 may take place in Outer Space (O for short) and has the stunt-sequence Fall / Fall / Crash / Explosion / Explosion (or FFCEE for short). The studio lot consists of a group of movie sets of various types, each of which can support a particular sequence of stunts, and two-way roads connecting these sets together. A scene in a movie script can be scheduled for a particular set if the scene and the set are of the same type and the stunt-sequence of the scene is a subsequence of the stunt-sequence for that set. A movie script of n scenes is viable if there is a path of n road-connected sets on the studio lot such that scene i can be scheduled for set i on this path for $1 \leq i \leq n$.

Give pseudocode for an algorithm that solves the scheduling problem above – namely, given a scene-sequence $S = \langle (t_1, s_1), (t_2, s_2), \ldots, (t_n, s_n) \rangle$ of set-type / stunt-sequence pairs such that $t_i \in \{U, C, O, U\}$ and s_i is a string over the alphabet $\{F, C, E\}$ and an undirected graph $L = (V, E)$ where each vertex v has an associated set-type $t(v)$ and stunt-sequence $s(v)$, determine whether or not there is a path $\langle v_1, v_2, \ldots, v_n \rangle$ of n vertices in L such that $t_i = t(v_i)$ and s_i is a subsequence of $s(v_i)$ for $1 \leq i \leq n$ and $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq (n-1)$. You may assume that no two adjacent scenes in scene-sequence S have the same set-type. Please give the asymptotic worst-case time complexity of your algorithm.

Answer: One possible pseudocode for this problem is as follows:

```
isValidSched(S, L = (V, E, s, t))
    isPath = false
    FOR each v in V DO
        isPath = isPath OR isPathDFS(S, L, v, 1)
    RETURN(isPath)

isPathDFS(S, L, v, i)
    IF ((t(v) == ti) AND
        (LCS(s(v), si) == |si|)) THEN
        IF (i == |S|) THEN RETURN(true)
        ELSE
            res = false
            FOR each vertex u in adj[v] DO
                res = res OR isPathDFS(S, L, u, i + 1)
            RETURN(res)
    ELSE
        RETURN(false)
```

In this pseudocode, method `LCS(x, y)` returns the length of the longest common subsequence of strings x and y. One can visualize the recursion tree of each of the depth-first searches starting from an initial vertex v in `isValidSched` as a $(|V|-1)$-ary tree of depth $|S|$. Let l be the length of the longest stunt-sequence string over all elements of S and vertices of L. As there are $O(|V|^{|S|})$ nodes in each vertex search tree and the time required to execute the recursive call associated with each node is $O(l^2)$, *i.e.*, the time required to execute one call to method `LCS`, the algorithm as a whole runs in $O(|V|(|V|^{|S|})l^2) = O(|V|^{|S|+1}l^2)$ time.

b) (18 marks) Let $G = (V, E, l)$ be an edge-labeled undirected graph such that each edge e has label $l(e) \in \{s, d\}$. Give pseudocode for a polynomial-time algorithm that computes a subset $E' \subseteq E$ of edges in G (if such an E' exists) such that in graph $G' = (V, E')$, there is a path between each pair of vertices in V and the number of edges with label d in E' is the smallest possible. Please give the asymptotic worst-case time complexity of your algorithm.

Answer: Observe that the requested set of edges E' forms a spanning tree of G which has the fewest number of d-labeled edges possible. Given this observation, one possible pseudocode for this problem is as follows:

```
Create an edge-weighted graph G' = (V, E, w)
 with the same vertex- and edge-sets as G
FOR each edge e in E DO
    IF l(e) = d THEN
        w(e) = 1
    ELSE
        w(e) = 0
E' = MinSpanningTree(G')
RETURN(E')
```

As minimum spanning trees can be computed in $O(|E| \log_2 |V|)$ time by either Kruskal's or Prim's algorithm implemented using appropriate data structures, the pseudocode given above runs in $O(|V| + |E| + |E| \log_2 |V|) = O(|E| \log_2 |V|)$ time.

5. (20 marks) Circle the letters associated with the appropriate answers in the the multiple choice questions in parts (i–v). Note that some of these questions may have more than one answer whose letter needs to be circled.

Suppose we have seven decision problems A, B, C, D, E, F, and G such that $A \leq_p B$, $A \leq_p C$, $A \leq_p D$, $B \leq_e C$, $C \leq_p E$, $C \leq_e G$, $D \leq_p B$, $E \leq_p B$, and $F \leq_p C$, where $X \leq_p Y$ ($X \leq_e Y$) means that there is a polynomial-time (exponential-time) many-one reduction from X to Y, *i.e.*, X reduces to Y.

i) (4 marks) What is implied if we know that B is solvable in polynomial time?

a) G is not solvable in polynomial time.
b) if G is solvable in polynomial time then $P = NP$.
c) C is solvable in exponential time.
d) F is solvable in polynomial time.

ii) (4 marks) What is implied if we know that A is NP-complete?

a) G is NP-hard.
b) B is NP-complete.
c) D is NP-hard.
d) C is not solvable in polynomial time.

iii) (4 marks) What is implied if we know that C is NP-hard and B is solvable in polynomial time?

a) D may be solvable in polynomial time.
b) $P = NP$.
c) G is solvable in polynomial time.
d) E is NP-complete.

iv) (4 marks) What is implied if we know that D is NP-hard and C is solvable in polynomial time?

a) F is solvable in polynomial time.
b) $P \neq NP$.
c) A is solvable in exponential time.
d) B may be solvable in polynomial time.

v) (4 marks) What is implied if we know that B is NP-hard, C is solvable in polynomial time, and F is in class NP?

a) F is solvable in polynomial time.
b) $P = NP$.
c) D is solvable in exponential time.
d) C may be solvable in polynomial time.

Answer: In this question, polynomial-time tractability (intractability) results are propagating backwards (forwards) along chains of polynomial-time many-one reductions. The easiest way to untangle what is going on is to draw a directed graph of the given reductions, where the vertices of this graph are problems and there is an arc from X to Y if there is a reduction from problem X to problem Y. Given such a graph, we can then label the vertices with the appropriate results and propagate these results accordingly.

The reduction-graph corresponding to the situation described in this question is given in Part (a) of Figure 1. Given the results described in parts (i–v), we can then deduce that the following implications are true:

Part (i): (c) and (d) (see part (b) of Figure 1)

Part (ii): (c) (see part (c) of Figure 1)

Part (iii): (a), (b), and (d) (see part (d) of Figure 1)

Part (iv): (a), (c), and (d) (see part (e) of Figure 1)

Part (iv): (a), (c) and (d) (see part (f) of Figure 1)

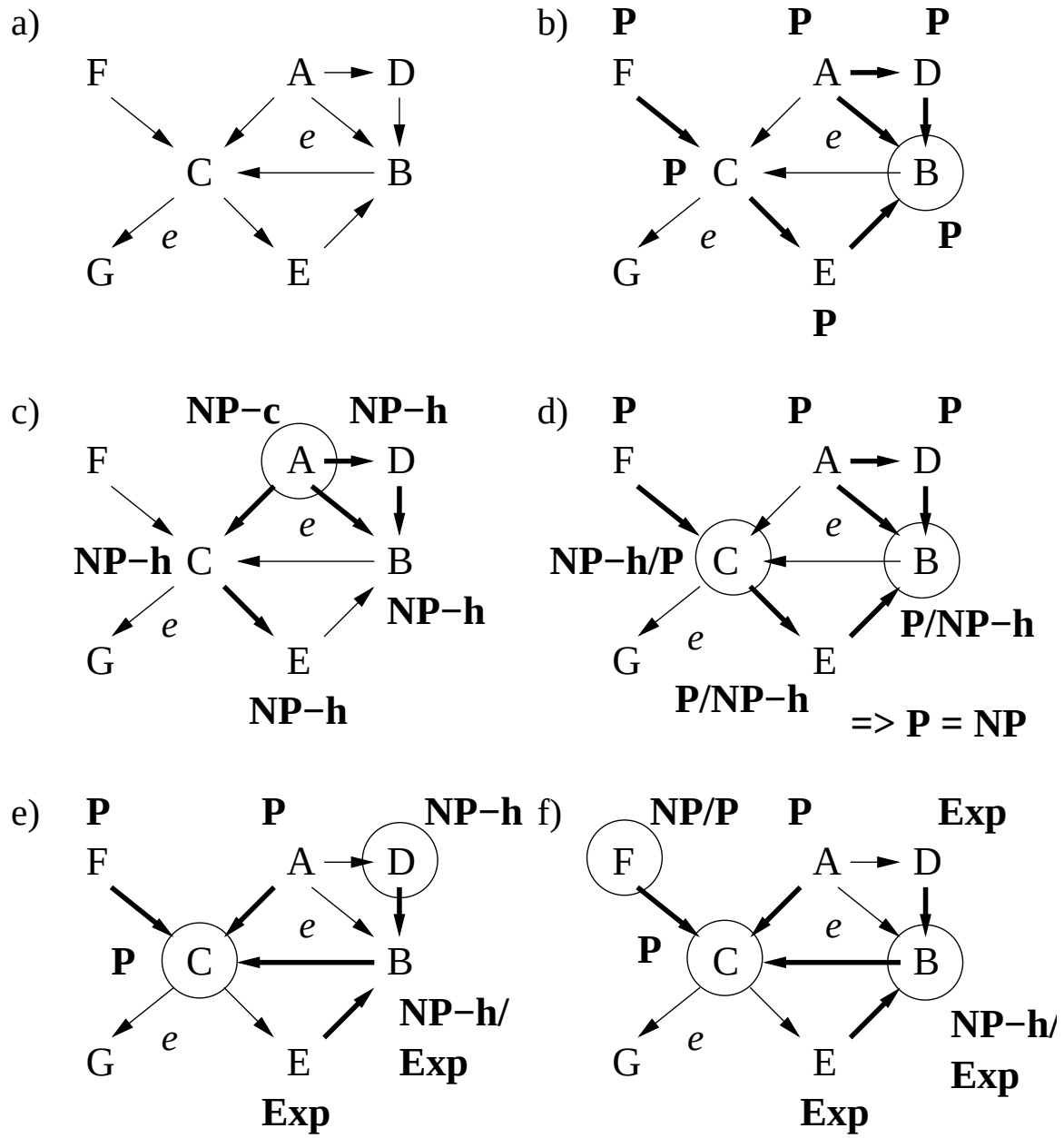


Figure 1: Answers for Question #5. **(a)** Reduction-graph for question. **(b – f)** Reduction-graphs relative to results described in subparts (i – v). Reductions along which results propagate are highlighted.