CS 325 Final Practice Solutions

1) Consider the following code for driving a robot in a spiral-like pattern:

SPIRAL(n):
1. If n ≤ 1, return (without doing any driving).
2. Drive north n miles.
3. Drive west n miles.
4. Drive south n miles.
5. Drive east n-1 miles.
6. Drive north 1 mile.
7. Call SPIRAL(n-2).

Let M(n) denote the number of miles that the robot drives (in total) if we call SPIRAL(n).

(a) M(1) = ***0 .***

(b) M(2) = ***8***

(c) Write a recurrence relation for M(n):

M(n) = M(n-2)+4n .

(d) Solve for M(n), as a function of n. Write your answer using O( ) notation.

M(n) = O(n^2) .

CS 325 Final Practice Solutions

2. Consider the following game. A "dealer" produces a sequence $s_1, \ldots, s_n$ of cards, face up, where each card s_i has a value v_i. Then two players take turns picking a card from sequence, but can only pick the first or the last card of the remaining sequence. The goal is to collect cards of the largest value. Assume n is even.

a) Show a sequence of cards such that it is not optimal for the first player to start by picking up the available card of larger value. That is, the greedy strategy is suboptimal.

Consider the sequence (5, 100, 1, 1).

b) Give an $O(n^2)$ algorithm to compute an optimal strategy for the first player. Given the initial sequence, your algorithm should precompute in $O(n^2)$ time some information and then the first player should be able to make each move optimally in O(1) time by looking up the precomputed information.

Let OPT(I,j) be the difference between: the largest total the first player can obtain and the corresponding score of the second player when playing on the sequence $s_i, \ldots, s_j$

Recursive formulation. At any stage of the game, there are 2 possible moves for the first player:

- either choose the first card, in which case he will gain s_i and score $-\mathbf{OPT}(i+1, j)$ in the rest of the game,
- or the last card, gaining s_j and $-\mathbf{OPT}(i, j-1)$ from the remaining cards.

Because we are interested in the largest total the first player can gain over the second, we take the maximum of these 2 values (for $i < j$):

$$\mathbf{OPT}(i,j) = \max\{s_i - \mathbf{OPT}(i+1,j), s_j - \mathbf{OPT}(i,j-1)\}$$

And the base cases are: $\forall i \in \{1,\ldots,n\},\ \mathbf{OPT}(i,i) = s_i$.

Pseudo-code.

```
// base cases
for i = 1...n:
  OPT[i, i] = s[i]

// main loop
for j = 1...n:
  for i = j...1:
    OPT[i, j] = max(s[i] - OPT(i+1, j), s[j] - OPT(i,j-1))

// final result
return OPT[1, n]
```

Complexity.

- We have $O(n^2)$ subproblems,
- each update takes time $O(1)$.

Therefore, the overall complexity is $O(n^2)$.

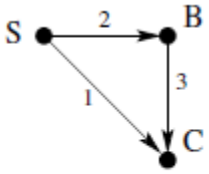
CS 325 Final Practice Solutions

3. Let G = (V, E) be a dag, where each edge is annotated with some positive length. Let s be a source vertex in G. Suppose we run Dijkstra's algorithm to compute the distance from s to each vertex v $\in$ V, and then order the vertices in increasing order of their distance from s. Are we guaranteed that this is a valid topological sort of G?

Circle YES or No .

Justify your answer as follows: If you circled YES, then give one sentence that explains the main idea in a proof of this fact. If you circled NO, then give a small counterexample (a graph with at most 4 vertices) that disproves it.

Answer:



Comment: Sorting by increasing distance gives S, C, B; but B must precede C in any valid topological sort.

CS 325 Final Practice Solutions

4. Suppose we have an alphabet with only five letters A, B, C, D, E which occur with the following frequencies.

Letter	A	B	C	D	E
frequency	0.35	0.12	0.18	0.05	0.30

Use Huffman coding to find the optimal prefix-free variable-length binary encoding of the alphabet.

(a) Draw a binary tree that represents the optimal encoding.

(b) Fill in the table below with the binary encoding of each letter.

Letter	encoding
A	
B	
C	
D	
E	

CS 325 Final Practice Solutions

5. Show that the Hamiltonian Cycle problem for directed graphs in NP-Complete.

Solution

1) Show you can verify a solution in Polynomial time

2) Show that Ham-Cycle (undirected) which is in NP-Complete can be reduced to Ham-Cycle-Directed.

Transform an undirected graph G into a directed graph H by replacing each edge vw of G by edges in each direction, vw and wv.

Then if there is a Hamiltonian cycle in G, there is one in H. For each edge in the cycle in G, the cycle in H uses the edge from a vw and wv pair that corresponds to the direction the edge was traversed in G.

If there is a Hamiltonian cycle in H, then there is one in G; simply replace either edge vw or wv in the cycle in H by the corresponding undirected edge in G.

6. Consider a connected weighted directed graph G = (V, E, w). Define the *fatness* of a path P to be the maximum weight of any edge in P. Give an efficient algorithm that, given such a graph and two vertices u,v $\in$V, finds the minimum possible fatness of a path from u to v in G.

Solution: There are two good solutions to this problem.

We can see that that fatness must be the weight of one of the edges, so we sort all edge weights and perform a binary search. To test whether or not a path with a fatness no more than x exists, we perform a breadth-first search that only walks edges with weight less than or equal to x. If we reach v, such a path exists.

If such a path exists, we recurse on the lower part of the range we are searching, to see if a tighter fatness bound also applies. If such a path does not exist, we recurse on the upper part of the range we are searching, to relax that bound. When we find two neighboring values, one of which works and one of which doesn't, we have our answer. This takes $O((V + E) \lg E)$ time.

Another good solution is to modify Dijkstra's algorithm. We use "fatness" instead of the sum of edge weights to score paths, and the only change to Dijkstra itself that is necessary is to change the relaxation operation so that it compares the destination node's existing min-fatness with the max of the weight of the incoming edge (i, j) and the min-fatness of any path to i (the source of the incoming edge).

The correctness argument is almost precisely the same as that for Dijkstra's algorithm. A correct solution also had to note that negative-weight edges, which could be present here and normally break Dijkstra, don't do that here; adding negative numbers produces ever-more-negative path weights, but taking their max doesn't. This solution has the same time complexity as Dijkstra's algorithm.

Two less-efficient solutions were to use Bellman-Ford instead of Dijkstra (with the same modified relaxation step and an analagous correctnes argument) and to perform the iterative search linearly instead of in a binary fashion. These recieved partial credit.

CS 325 Final Practice Solutions

7. Solve the recurrences by giving tight bounds.

a) $T(n) = 4T(n/2) + n\log n$, $T(1) = 1$

By the master method $a=4$, $b=2$, $\log_2 4 = 2$,
$f(n)=n\log n = O(n^{2-\epsilon})$
Case 1:
$T(n) = \Theta(n^2)$

b) $T(n) = 2T(n/2) + n^2$, $T(1) = 1$.

By the master method $a=2$, $b=2$, $\log_2 2 = 1$,
$f(n) = n^2 = \Omega(n^{1+\epsilon})$
Case 3:
Check regularity:
$2f(n/2) \le f(n)$
$2(n/2)^2 = n^2/2 <= c\, n^2$, $c = ½$
Therefore, $T(n) = \Theta(n^2)$

c) $T(n) = T(n/2) + n^3$, $T(1) = 1$.

By the master method $a=1$, $b=2$, $\log_2 1 = 0$,
$f(n) = n^3 = \Omega(n^{0+\epsilon})$
Case 3:
Check regularity:
$f(n/2) \le f(n)$
$(n/2)^3 = n^3/8 <= c\, n^3$, $c = 1/8$
Therefore, $T(n) = \Theta(n^3)$

d) $T(n) = 3\,T(n-1) + 1$, $T(1) = 1$

$T(n) = 3\,T(n-1) + 1$
$= 3(3T(n-2)+1) + 1 = 3^2T(n-2) + 3 + 1$
$= 3^2(3T(n-3) + 1) + 3 + 1 = 3^3T(n-3) + 9 + 3 + 1$
…
$= 3^kT(n-k) + 3^{k-1} + \ldots + 9 + 3 + 1$ Stop when $n-k = 1$ or $k = n-1$
$= 3^{n-1} + 3^{n-2} + \ldots + 9 + 3 + 1$
$= \Theta(3^n)$

CS 325 Final Practice Solutions

8. In the bottleneck-path problem, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W. Describe an efficient algorithm to solve this problem.

Solution: *Run BFS ignoring any edges of weight less than W. This will take O(V+E) time.*

9. Consider the modified binary search algorithm so that it splits the input not into two sets of almost-equal sizes, but into four sets of sizes approximately one-fourth. Write the recurrence for this quarternary search algorithm and find the asymptotic complexity of the algorithm.

$T(n) = T(n/4) + 3$

By the master method $T(n) = \Theta(\log_4 n)$

CS 325 Final Practice Solutions

10. Macrosoft has a 24-hour-a-day, 7-days-a-week toll free hotline that is being set up to answer questions regarding a new product. The following table summarizes the number of full-time equivalent employees (FTEs) that must be on duty in each time block. Macrosoft may hire both full-time and part-time employees.

- Full-timers work 8-hour shifts with hourly wages of $15.20
- Part-timers work 4-hour shifts with hourly wages of $12.95.
- Employees may start work only at the beginning of 1 of the 6 intervals.
- Part-time employees can only answer 5 calls in the time a full-time employee can answer 6 calls. (i.e., a part-time employee is only 5/6 of a full-time employee.)
- At least two-thirds of the employees working at any one time must be full-time employees.

Formulate an LP to determine how to staff the hotline at minimum cost.

Interval	Time	FTEs
1	0-4	15
2	4-8	10
3	8-12	40
4	12-16	70
5	16-20	40
6	20-0	35

Decision Variables

x_t = # of full-time employees that begin the day at the start of interval t and work for 8 hours

y_t = # of part-time employees that are assigned interval t

$$\min \quad \underbrace{121.6(x_1 + \cdots + x_6)}_{(8 \times 15.20)} + \underbrace{51.8(y_1 + \cdots + y_6)}_{(4 \times 12.95)}$$

s.t.

$$\begin{aligned}
x_1 + x_6 + \tfrac{5}{6}y_1 &\geq 15 \\
x_1 + x_2 + \tfrac{5}{6}y_2 &\geq 10 \\
x_2 + x_3 + \tfrac{5}{6}y_3 &\geq 40 \\
x_3 + x_4 + \tfrac{5}{6}y_4 &\geq 70 \\
x_4 + x_5 + \tfrac{5}{6}y_5 &\geq 40 \\
x_5 + x_6 + \tfrac{5}{6}y_6 &\geq 35
\end{aligned}$$

All shifts must be covered

PT employee is 5/6 FT employee

$$\begin{aligned}
x_1 + x_6 &\geq \tfrac{2}{3}(x_6 + x_1 + y_1) \\
x_1 + x_2 &\geq \tfrac{2}{3}(x_1 + x_2 + y_2) \\
&\vdots \\
x_5 + x_6 &\geq \tfrac{2}{3}(x_5 + x_6 + y_6)
\end{aligned}$$

At least 2/3 workers must be full time

$x_t \geq 0, \; y_t \geq 0 \quad t = 1,2,\ldots,6$ Nonnegativity

CS 325 Final Practice Solutions

11. Complete the following table

	$T_1(n)$	$T_2(n)$	Is $T_1(n) = O(T_2(n))$?	Is $T_1(n) = \Omega(T_2(n))$?	Is $T_1(n) = \Theta(T_2(n))$?
a	$25n \ln n + 5n$	$\frac{1}{2} n \log_2 n$	yes	yes	yes
b	$\frac{1}{2} n^2 + n \log_2 n$	$5n \log_2 n$	no	yes	no
c	$\sqrt{n}(\log_2 n)$	n	yes	no	no
d	$2^{\log_2 n}$	$2n^2$	yes	no	no
e	$n\sqrt{n}$	$n^{1.4}$	no	yes	no

(a) Recall that $\log_2 n = \frac{\ln n}{\ln 2}$. Hence $\lim_{n\to\infty} \frac{25n \ln n + 5n}{\frac{1}{2} n \log_2 n} = \lim_{n\to\infty} \left(\frac{25n \ln n}{n \ln n/(2 \ln 2)} + \frac{5n}{n \ln n/(2 \ln 2)} \right) = \lim_{n\to\infty} \left(50 \ln 2 + \frac{10 \ln 2}{\ln n} \right) = 50 \ln 2 \approx 35$. They grow at the same asymptotic growth rate, however A_2 is almost 35 times faster and hence is the preferred algorithm.

(b) $\lim_{n\to\infty} \frac{1/2n^2 + n \log_2 n}{5n \log_2 n} = \lim_{n\to\infty} \left(\frac{n}{10n \log_2 n} + \frac{1}{5} \right) = \infty$ and hence T_1 is asymptotically faster growing and so A_2 is the preferred algorithm.

(c) $\lim_{n\to\infty} \frac{\sqrt{n} \log_2 n}{n} = \lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = 0$ and hence T_2 is asymptotically faster growing and so A_1 is the preferred algorithm.

(d) $2^{\log_2 n} = n$. Hence $\lim_{n\to\infty} T_1(n)/T_2(n) = \lim_{n\to\infty} 1/(2n) = 0$. Hence $T_1(n)$ is asymptotically slower growing and thus A_1 is the preferred algorithm.

(e) Note that $n\sqrt{n} = n^{1.5}$. So $\lim_{n\to\infty} \frac{n^{1.5}}{n^{1.4}} = \infty$ and hence T_1 is asymptotically faster growing.

12. Prove or disprove the following $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

DISPROVE: Many counter examples. Let f(n) = n^3 and g(n) = n. min(f(n), g(n)) = n,

n^3+n is not Θ(n) .

CS 325 Final Practice Solutions

13. For each of the following give a tight Θ( ) bound on the number of times the z ← z + 1 statement is executed and justify your solution.

```
j ← 0
while (j < n) do
  j ← j + 2
  z ← z + 1
```

AN ANSWER. Since j goes through the values 0, 2, 4, 6, ... until j reaches n (if n is even) or $n+1$ (if n is odd), the while loop goes through at most $\lceil n/2 \rceil$ many iterations. Hence, z is in $\Theta(n)$.

```
for k ← 0 to n do
  for j ← 0 to k do
    z ← z + 1
```

AN ANSWER. *Inner loop:* Since j goes from 0 to k, the inner loop has $(k+1)$-many iterations. So, z is increased by $(k+1)$. *Outer loop:* k goes from 0 to n. So z is increased by

$$\sum_{k=0}^{n}(k+1) = 1+2+\cdots+n+(n+1) = \frac{(n+1)(n+2)}{2} \in \Theta(n^2).$$

```
i ← n
while (i > 1) do
  i ← ⌊i/2⌋
  z ← z + 1
```

AN ANSWER. Since i takes on the sequence of values $n = n/2^0$, $\lfloor n/2 \rfloor = \lfloor n/2^1 \rfloor$, $\lfloor n/4 \rfloor = \lfloor n/2^2 \rfloor$, ..., $\lfloor n/2^j \rfloor$ until i is ≤ 1. The smallest value of i such that $n/2^i \leq 1$ is $\lceil \log_2 n \rceil$. So there are $(1 + \lceil \log_2 n \rceil)$-many iterations and $z \in \Theta(\log_2 n)$.

14.

Ans: To show that 4-SAT is NP-complete, we prove that 4-SAT is in NP and NP-hard.

First, 4-SAT is in NP, we can write a nondeterministic polynomial-time algorithm which takes a 4-SAT instance and a proposed truth assignment as input. This algorithm evaluates the 4-SAT instance with the truth assignment. If the 4-SAT instance evaluates to true, the algorithm outputs *yes*; otherwise, the algorithm outputs *no*. This runs in polynomial time.

To prove that 4-SAT is NP-hard, we reduce 3-SAT to 4-SAT as follows. Let ϕ denote an instance of 3-SAT. We convert ϕ to a 4-SAT instance ϕ' by turning each clause $(x \vee y \vee z)$ in ϕ to $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$, where h is a new variable. Clearly this is polynomial-time doable.

$\Rightarrow$ If a given clause $(x \vee y \vee z)$ is satisfied by a truth assignment, then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ is satisfied by the same truth assignment with h arbitrarily set. Thus if ϕ is satisfiable, ϕ' is satisfiable.

$\Leftarrow$ Suppose ϕ' is satisfied by a truth assignment T. Then $(x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \neg h)$ must be true under T. As h and $\neg h$ assume different truth values, $x \vee y \vee z$ must be true under T as well. Thus ϕ is satisfiable.

15. O(n)