HOMEWORK #3
SELECTED SOLUTIONS
CSCI-2300 INTRO TO ALGORITHMS (SPRING 2014)
DASGUPTA 6.4

a) Subproblems: Define an array of subproblems $S(i)$ for $0 \leq i \leq n$ where $S(i)$ is 1 if $s[1 \cdots i]$ is a sequence of valid words and is 0 otherwise.

Algorithm and Recursion: It is sufficient to initialize $S(0) = 1$ and update the values $S(i)$ in ascending order according to the recursion

$$S(i) = \max_{0 \leq j < i} \{S(j) : \text{dict}(s[j+1 \cdots i]) = \text{true}\}$$

Then, the string $s$ can be reconstructed as a sequence of valid words if and only if $S(n) = 1$.

Correctness and Running Time: Consider $s[1 \cdots i]$. If it is a sequence of valid words, there is a last word $s[j \cdots i]$, which is valid, and such that $S(j) = 1$ and the update will cause $S(i)$ to be set to 1. Otherwise, for any valid word $s[j \cdots i]$, $S(j)$ must be 0 and $S(i)$ will also be set to 0. This runs in time $O(n^2)$ as there are $n$ subproblems, each of which takes time $O(n)$ to be updated with the solution obtained from smaller subproblems.

b) Every time a $S(i)$ is updated to 1 keep track of the previous item $S(j)$ which caused the update of $S(i)$ because $s[j+1 \cdots i]$ was a valid word. At termination, if $S(n) = 1$, trace back the series of updates to recover the partition in words. This only adds a constant amount of work at each subproblem and a $O(n)$ time pass over the array at the end. Hence, the running time remains $O(n^2)$.

---

DASGUPTA 6.4

• [8pts total] Grading notes for 6.4(a):
  • [3pts] Correct definition of subproblem
  • [3pts] Correct recursive relation between subproblem and larger problem
  • [2pts] Algorithm’s runtime satisfies $O(n^2)$
  • If a correct algorithm was given without using dynamic programming, 4pt penalty

• [8pts total] Grading notes for 6.4(b):
  • Different feasible solutions are acceptable, as long as a correct means of keeping track of the position where word partitioning occurs
DASGUPTA 6.5

a) There are 8 possible patterns: the empty pattern, the 4 patterns which each have exactly one pebble, and the 3 patterns that have exactly two pebbles (on the first and fourth squares, the first and third squares, and the second and fourth squares).

b) Number the 8 patterns 1 through 8, and define $S \subseteq \{1, 2, \ldots, 8\} \times \{1, 2, \ldots, 8\}$ to be all $(n, k)$ such that pattern $a$ is compatible with pattern $b$. For each pattern, there are a constant number of patterns that are compatible (for example, every pattern is compatible with the empty pattern).

**Sub-problems and Recursion:** We consider the sub-problem $L[i, j]$, $i = 0, 1, 2, \ldots, n$ and $j \in \{1, 2, \ldots, 8\}$ to be the maximal value achievable by pebbling columns 1, 2, ..., $t$ such that the final column has pattern $j$. It is easy to see that:

$$L[i+1, j] = \max_{(k,j) \in S} L[i, k]$$

The base case is $L[0, j] = 0$ for all $j$. In order to recover the optimal placement, we should also maintain a back-pointer: $P[i+1, j] = k$ is the value of $k$ such that $(k, j) \in S$ and $L[i, k]$ is maximal.

**Algorithm and Running Time:** For $i = 0, 1, \ldots, n$, for $j = 1, 2, \ldots, 8$, compute $L[i, j]$ and $P[i, j]$ using the recurrence. Note that $L[i, j]$ and $P[i, j]$ can be computed using the recursion in constant time since we only need to check a constant number of possible $k$. The value of the optimal placement is given by $\max_j L[n, j]$.

To recover the optimal placement, let $j^*$ be the value of $j$ for which $L[n, j]$ is maximal. Then, column $n$ should be pebbled using pattern $j^*$. Then, column $n-1$ should be pebbled using pattern $P[n, j^*]$, column $n-2$ with pattern $P[n-1, P[n, j^*]]$, and so on.

The running time of this algorithm is $O(n)$ because compute the recurrence takes $O(n)$ time and backtracking takes $O(1)$ time per column.

---

DASGUPTA 6.5

- **[8pts total] Grading notes for 6.5(a):**
  - One point for each legal pattern
  - If only the number of patterns is given without descriptions, 4pt penalty

- **[8pts total] Grading notes for 6.5(b):**
  - **[3pts]** Correct definition of subproblem
  - **[3pts]** Correct recursive relation between subproblem and larger problem
  - **[2pts]** Algorithm's runtime satisfies $O(n)$
  - If a correct algorithm was given without using dynamic programming, 4pt penalty
One way to solve this problem is by dynamic programming.

**Subproblems:** Define $L(i, j)$ to be the probability of obtaining exactly $j$ heads amongst the first $i$ coin tosses.

**Algorithm and Recursion:** By the definition of $L$ and the independence of the tosses, it is clear that:

$$L(i, j) = p_j L(i - 1, j - 1) + (1 - p_j) L(i - 1, j) \quad j = 0, 1, \ldots, i$$

We can then compute all $L(i, j)$ by initializing $L(0, 0) = 1$, $L(i, j) = 0$ for $j < 0$, and proceeding incrementally (in the order $i = 1, 2, \ldots, n$, with inner loop $j = 0, 1, \ldots, i$). The final answer is given by $L(n, k)$.

**Correctness and Running Time:** The recursion is correct as we can get $j$ heads in $i$ coin tosses either by obtaining $j - 1$ heads in the first $i - 1$ coin tosses and throwing a head on the last coin, which takes place with probability $p_j L(i - 1, j - 1)$, or by having already $j$ heads after $i - 1$ tosses and throwing a tail last, which has probability $(1 - p_j) L(i - 1, j)$. Besides, these two events are disjoint, so the sum of their probabilities equals $L(i, j)$. Finally, computing each subproblem takes constant time, so the algorithm runs in $O(n^2)$ time.

**Grading notes for 6.10:**

- [6pts total] Grading notes for 6.10:
  - [6pts] Correct definition of subproblem
  - [6pts] Correct recursive relation between subproblem and larger problem
  - [4pts] Algorithm's runtime satisfies $O(n^2)$
  - If a correct algorithm was given without using dynamic programming, 4pt penalty
DASGUPTA 6.17

• [16pts total] Grading notes for 6.17:
  • [6pts] Correct definition of subproblem
  • [6pts] Correct recursive relation between subproblem and larger problem
  • [4pts] Algorithm’s runtime satisfies $O(nv)$
  • If a correct algorithm was given without using dynamic programming, 4pt penalty

DASGUPTA 8.3

It’s a generalization of SAT. Given a SAT formula $\varphi$ with $n$ variables, $(\varphi, n)$ is an instance of STINGY SAT which has a solution if and only if the original SAT formula has a satisfying assignment.

• [16pts total] Grading notes for 8.3:
  • If incorrect attempt was made, 10pt penalty
DASGUPTA 8.4

(a) Given a clique in the graph, it is easy to verify in polynomial time that there is an edge between every pair of vertices. Hence a solution to CLIQUE-3 can be checked in polynomial time.

(b) The reduction is in the wrong direction. We must reduce CLIQUE to CLIQUE-3, if we intend to show that CLIQUE-3 is at least as hard as CLIQUE.

(c) The statement “a subset \( C \subseteq V \) is a vertex cover in \( G \) if and only if the complimentary set \( V - C \) is a clique in \( G \)” used in the reduction is false. \( C \) is a vertex cover if and only if \( V - C \) is an \textit{independent set} in \( G \).

(d) The largest clique in the graph can be of size at most 4, since every vertex in a clique of size \( k \) must have degree at least \( k - 1 \). Thus, there is no solution for \( k > 4 \), and for \( k \leq 4 \) we can check every \( k \)-tuple of vertices, which takes \( O(|V|^k) = O(|V|^4) \) time.

---

DASGUPTA 8.4

- [4pts total] Grading notes for 8.4(a):
  - If incorrect attempt was made, 3pt penalty; for minor mistake(s), 1pt penalty

- [4pts total] Grading notes for 8.4(b):
  - If incorrect attempt was made, 3pt penalty; for minor mistake(s), 1pt penalty

- [4pts total] Grading notes for 8.4(c):
  - If incorrect attempt was made, 3pt penalty; for minor mistake(s), 1pt penalty

- [4pts total] Grading notes for 8.4(d):
  - If incorrect attempt was made, 3pt penalty; for minor mistake(s), 1pt penalty