Solutions to Homework 1 Practice Problems

CS6515 Spring 2026

[DPV] Problem 6.4 – Dictionary Lookup

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
Let T(i) = TRUE if s[1], s[2], ..., s[i] can be reconstituted as a sequence
of valid words, otherwise FALSE.
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(0) = TRUE

T(i) = TRUE     if OR{T(j-1) and dict(s[j..i]) : for 1 <= j <= i}
     = FALSE    otherwise
where 1 <= i <= n
```

c. Implementation Analysis

(1) State the number of subproblems: $O(n)$
(2) State the runtime for table fill: $O(n^2)$
(3) State how the return is extracted: `return T(n)`
(4) State the runtime of that return extraction: $O(1)$

Note

For purposes of this problem, we're given the runtime of `dict()` as $O(1)$. We will similarly consider the slicing of `s[]` to be part of that same operation for purposes of runtime analysis, for this problem only.

[DPV] Problem 6.8 – Longest Common Substring

Note

Here we are doing the longest common **substring**, as opposed to the longest common **subsequence**.

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
T(i,j) = length of the longest common substring for x and y, ending with x[i] = y[j]
        (which is zero when the characters do not match).
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(i, 0) = 0 where 0 <= i <= n
T(0, j) = 0 where 1 <= j <= m

T(i, j) = 1 + T(i-1, j-1)   if x[i] = y[j]
        = 0                 if x[i] != y[j]
where 1 <= i <= n
where 1 <= j <= m
```

c. Implementation Analysis

(1) State the number of subproblems: $O(n * m)$
(2) State the runtime for table fill: $O(n * m)$
(3) State how the return is extracted: `return max(T(*, *))`
(4) State the runtime of that return extraction: $O(n * m)$

[DPV] Problem 6.18 – Making Change II

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
T(i,j) = max sum at most j using a subset of coins x[1], x[2], ..., x[i].
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(i, 0) = 0 for 0 <= i <= n
T(0, j) = 0 for 1 <= j <= v

T(i, j) = max{T(i-1, j), x[i] + T(i-1, j-x[i])}  if x[i] <= j
        = T(i-1, j)                              if x[i] > j
where 1 <= i <= n
where 1 <= j <= v
```

c. Implementation Analysis

(1) State the number of subproblems: $O(n * v)$
(2) State the runtime for table fill: $O(n * v)$
(3) State how the return is extracted: `return TRUE if v = T(n,v), else FALSE`
(4) State the runtime of that return extraction: $O(1)$

Note

It is also possible to solve this using a boolean table:

```
Subproblem:
  D(i,j) = TRUE if there is a subset of coins x[1], x[2], ..., x[i]
           to form the value j.

Base Cases:
  D(0,0) = TRUE
  D(0,j) = FALSE : 1 <= j <= v

Recurrence:
  D(i,j) = D(i-1, j) OR D(i-1, j-x[i])   if x[i] <= j
         = D(i-1, j)                     if x[i] > j
  where 1 <= i <= n
  where 0 <= j <= v

Return: D(n,v)
```

[DPV] Problem 6.19 – Making Change K

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
T(j) = minimum number of coins needed to make the exact value j using
a multiset of coins x[1], x[2], ..., x[n] if possible, or infinity otherwise
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(0) = 0

T(j) = min {1 + T(j-x[i]) : 1 <= i <= n if x[i] <= j}
     = inf    otherwise
where 1 <= j <= v
```

c. Implementation Analysis

(1) State the number of subproblems: $O(v)$
(2) State the runtime for table fill: $O(v * n)$
(3) State how the return is extracted: `return TRUE if T(v) <= k, else FALSE`
(4) State the runtime of that return extraction: $O(1)$

[DPV] Problem 6.20 – Optimal Binary Search Tree

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
T(i,j) = minimum cost to look up words i, i+1, .., j
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(i, i)   = p[i],  where 1 <= i <= n
T(i, i-1) = 0,     where 1 <= i <= n+1

T(i, j) = sum{p[i..j]} + min{T(i, k-1) + T(k+1, j) : i <= k <= j}
where 1 <= i < j <= n
```

c. Implementation Analysis

(1) State the number of subproblems: $O(n^2)$
(2) State the runtime for table fill: $O(n^3)$
(3) State how the return is extracted: `return T(1, n)`
(4) State the runtime of that return extraction: $O(1)$

[DPV] Problem 6.26 – Alignment

a. Define the entries of your table in words. E.g., T(i) or T(i, j) is ...

```
T(i,j) = maximum score of x[1], x[2], ..., x[i] and y[1], y[2], ..., y[j]
```

b. State recurrence for entries of your table in terms of smaller subproblems.

```
T(0, 0) = 0
T(i, 0) = T(i-1, 0) + δ(x[i], -), where 1 <= i <= n
T(0, j) = T(0, j-1) + δ(-, y[j]), where 1 <= j <= m

T(i, j) = max { T(i-1, j-1) + δ(x[i], y[j]),
                T(i-1, j) + δ(x[i], -),
                T(i, j-1) + δ(-, y[j])}
where 1 <= i <= n
where 1 <= j <= m
```

c. Implementation Analysis

(1) State the number of subproblems: $O(n * m)$
(2) State the runtime for table fill: $O(n * m)$
(3) State how the return is extracted: `return T(n, m)`
(4) State the runtime of that return extraction: $O(1)$