Solutions to Homework Eight

CSE 101

1. *Textbook problem 6.1.*

   Subproblem: Let $S(j)$ be the sum of the maximum-sum contiguous subsequence which ends exactly at a_j (but is possibly of length zero). We want $\max_j S(j)$.

   Recursive formulation: The subsequence defining $S(j)$ either (i) has length zero, or (ii) consists of the best subsequence ending at a_{j-1}, followed by element a_j. Therefore,

   $$S(j) = \max\{0, a_j + S(j-1)\}.$$

   For consistency $S(0) = 0$.

   Algorithm:

   $S[0] = 0$
   `for` $j = 1$ `to` n:
       $S[j] = \max(0, a_j + S[j-1])$
   `return` $\max_j S[j]$

   Running time: Single loop: $O(n)$.

2. *Textbook problem 6.2.*

   Subproblem: Let $T(j)$ be the minimum penalty incurred upto location a_j, assuming you stop there. We want $T(n)$.

   Recursive formulation: Suppose we stop at a_j. The previous stop is some $a_i, i < j$ (or maybe a_j is the very first stop). Let's try all possibilities for a_i:

   $$T(j) = \min_{0 \le i < j} T(i) + (200 - (a_j - a_i))^2,$$

   where for convenience we set $T(0) = 0$ and $a_0 = 0$.

   Algorithm:

   `for` $j = 1$ `to` n:
       $T(j) = (200 - a_j)^2$
       `for` $i = 1$ `to` $j - 1$:
           $T(j) = \min\{T(j), T(i) + (200 - (a_j - a_i))^2\}$
   `return` $T(n)$

   Running time: Two loops, $O(n^2)$.

3. *Textbook problem 6.7.*

   Subproblem: Define $T(i, j)$ to be the length of the longest palindromic subsequence of $x[i \ldots j]$. We want $T(1, n)$.

   Recursive formulation: In computing $T(i, j)$, the first question is whether $x[i] = x[j]$. If so, we can match them up and then recurse inwards, to $T(i+1, j-1)$. If not, then at least one of them is not in the palindrome.

   $$T(i,j) = \begin{cases} 1 & \text{if } i = j \\ 2 + T(i+1, j-1) & \text{if } i < j \text{ and } x[i] = x[j] \\ \max\{T(i+1, j), T(i, j-1)\} & \text{otherwise} \end{cases}$$

   For consistency set $T(i, i-1) = 0$ for all i.

   Algorithm: Compute the $T(i, j)$ in order of increasing interval length $|j - i|$.

```
for i = 2 to n+1:
   T[i, i-1] = 0
for i = 1 to n:
   T[i, i] = 1
for d = 1 to n-1:      (interval length)
   for i = 1 to n-d:
      j = i + d
      if x[i] = x[j]:
         T[i, j] = 2 + T[i+1, j-1]
      else:
         T[i, j] = max{T[i+1, j], T[i, j-1]}
return T[1, n]
```

Running time: There are $O(n^2)$ subproblems and each takes $O(1)$ time to compute, so the total running time is $O(n^2)$.

4. *Textbook problem 6.17.*

   Subproblem: For any integer $0 \leq u \leq v$, define $T(u)$ to be `true` if it is possible to make change for u using the given coins $x_1, x_2, \ldots, x_n$. The answer we want is $T(v)$.

   Recursive formulation: Notice that

   $T(u)$ is `true` if and only if $T(u - x_i)$ is true for some i.

   For consistency, set $T(0)$ to `true`.

   Algorithm:

   ```
   T[0] = true
   for u = 1 to v:
      T[u] = false
      for i = 1 to n:
         if u ≥ x_i and T[u - x_i]:   T[u] = true
   ```

   Running time: The table has size v and each entry takes $O(n)$ time to fill; therefore the total running time is $O(nv)$.

5. *Number of paths in a DAG.*

   Subproblem: Suppose G is a directed acyclic graph. For any node v in the graph, define $\texttt{numpaths}[v]$ to be the number of paths from s to v. The quantity we want is $\texttt{numpaths}[t]$.

   Recursive formulation: Pick any node $v \neq s$ in the graph. Any path from s to v ends in an edge $(u, v) \in E$. Thus:

   $$\texttt{numpaths}[v] = \sum_{u:(u,v)\in E} \texttt{numpaths}[u].$$

   And of course, $\texttt{numpaths}[s] = 1$.

   Algorithm: We can fill out the array by considering the nodes in topological order:

   ```
   Find a topological ordering of G
   for all v ∈ V:
      numpaths[v] = 0
   numpaths[s] = 1
   for all u ∈ V, in topological order:
      for all (u, v) ∈ E:
        numpaths[v] = numpaths[v] + numpaths[u]
   return numpaths[t]
   ```

Running time: The total running time is $O(V+E)$, linear.

6. *Textbook problem 6.21.*

   Subproblem: Root the tree at any node r. For each $u \in V$, define

   $$T(u) = \text{size of smallest vertex cover of the subtree rooted at } u.$$

   We want $T(r)$.

   Recursive formulation: In figuring out $T(u)$, the most immediate question is whether u is in the vertex cover. If not, then its children must be in the vertex cover. Let $C(u)$ be the set of u's children, and let $G(u)$ be its grandchildren. Then

   $$T(u) = \min \begin{cases} 1 + \sum_{w \in C(u)} T(w) \\ |C(u)| + \sum_{z \in G(u)} T(z) \end{cases}$$

   where $|C(u)|$ is the number of children of node u. The first case includes u in the vertex cover; the second case does not.

   Algorithm:

   ```
   Pick any root node r
   dist[·] = BFS(tree, r)

   for all nodes u, in order of decreasing dist:
      S_1 = 1 (option 1: include u in the vertex cover)
      for all (u,w) ∈ E such that dist[w] = dist[u] + 1:  (ie. w = child of u):
         S_1 = S_1 + T[w]
      S_2 = 0 (option 2: don't include u in the vertex cover)
      for all (u,w) ∈ E such that dist[w] = dist[u] + 1:  (ie. w = child of u):
         S_2 = S_2 + 1
         for all (w,z) ∈ E such that dist[z] = dist[w] + 1:  (ie. z = grandchild of u):
            S_2 = S_2 + T[z]
      T[u] = min{S_1, S_2}
   return T[r]
   ```

   Running time: The work done at each node is proportional to its number of grandchildren, $|G(u)|$. Since $\sum_u |G(u)| \leq |V|$ (each node has at most one grandparent), the overall work done is linear.

7. *Textbook problem 6.19.*

   Subproblem: For any integers $0 \leq u \leq v$ and $0 \leq j \leq k$, define $T(u,j)$ to be `true` if it is possible to make change for u using at most j coins with denominations chosen from $x_1, x_2, \ldots, x_n$. The answer we want is $T(v,k)$.

   Recursive formulation: Notice that

   $T(u,j)$ is `true` if and only if (either $u = 0$ or ($T(u - x_i, j-1)$ is true for some i)).

   For consistency, set $T(0,j)$ to `true` for all j and $T(u,0)$ to `false` for $u > 0$.

   Algorithm:

   ```
   for j = 0 to k:
      T[0, j] = true
   for u = 1 to v:
      T[u, 0] = false
   ```

```
for j = 1 to k:
   for u = 1 to v:
      T[u, j] = false
      for i = 1 to n:
         if u ≥ x_i and T[u − x_i, j − 1]:   T[u, j] = true
return T[v, k]
```

Running time: The table has size $k \times v$ and each entry takes $O(n)$ time to fill; therefore the total running time is $O(nkv)$.