Problem 1: Max Sum Contiguous Subsequence
A contiguous subsequence of a list S is a subsequence made up of consecutive elements of S. For instance, if S is $(5, 15, -30, 10, -5, 40, 10)$ then $(15, -30, 10)$ is a contiguous subsequence, but $(5, 15, 40)$ is not. Give a dynamic programming linear time algorithm for the following task:
Input: A list of numbers, $(a_1, a_2, \dots, a_n)$.
Output: A contiguous subsequence of maximum sum (a subsequence of length zero has sum zero).
For the preceding example, the answer would be $(10, -5, 40, 10)$, with a sum of 55.
Hint: For each $j \in \{1, 2, \dots, n\}$ consider a contiguous subsequences ending exactly at position j.
Write explicitly (a)the recursive form of the solution (ie your thinking in words) (b)the pseudocode implementation and (c)the complexity analysis.

Answer:

(a) For each $j \in \{1, 2, \cdots, n\}$, let $L(j)$ be the maximum sum of contiguous path ending exactly at position j and $s(j)$ be the start position of that path. The answer we seek is $max_{j=1\cdots n}\{L(j)\}$ and its corresponding path. Then the maximum path can be obtained using its start position.(i.e. if $k = argmax_{j=1\cdots n}\{L(j)\}$, the maximum path is $(a_{s(k)}, a_{s(k)+1}, \cdots, a_k)$.) For each j, the contiguous path ending with j can be classified in two cases: 1) started earlier 2) just start at j. So we can express as two smaller subproblems and deduce that

$$L(j) = max\{L(j-1) + a_j, a_j\}$$

(b)

```
initialize $L(1) = a(1), s(1) = 1$ for all $j \in 1, 2, \cdots, n$;
for $j = 2$ to $n$ do
    if $L(j-1) > 0$ then
        $L(j) = L(j-1) + a(j)$
        $s(j) = s(j-1)$
    else
        $L(j) = a(j)$
        $s(j) = j$
    end
end
$pe = argmax_j L(j)$
$ps = s(pe)$
$maxsum = L(pe)$
```

(c) This algorithm fills in a one-dimensional table of length n, in left-to-right order. Each entry take constant time, so the overall running time is $O(n)$.

Problem 2, Hotel Trip
You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \ldots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination. You would ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the penalty for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty, that is, the sum, over all travel days, of the daily penalties. Give and analyze a polynomial time dynamic programming algorithm that determines the optimal sequence of hotels at which to stop.
Write explicitly (a)the recursive form of the solution (i.e. your thinking in words) (b)the pseudo code implementation and (d)the complexity analysis.

Answer:

(a) For each $j \in \{1, 2, \cdots, n\}$, let $P(j)$ be the minimum penalty when you arrive at hotel j and stay there. And let $pre(j)$ be the index of hotel that you stayed the previous day. We assume that the starting point has index 0. The answer we seek is $P(n)$ and its corresponding path. We can get the optimal path through bookkeeping by $pred(\cdot)$. For each j, $P(j)$ is $P(k) + (200 - a_j + a_k)^2$ for some $k < j$, i.e. we stayed at some hotel before(including starting point - just think the starting point as hotel 0) and traveled to current hotel without staying elsewhere. So we can deduce that

$$P(j) = min\{P(k) + (200 - a_j + a_k)^2 : k < j\}$$

(b)

```
initialize $P(0) = 0, pre(0) = -1$
**for** $j$ = *1 to n* **do**
    $P(j) = min_{k=0,1,\cdots,j-1}\{P(k) + (200 - a_j + a_k)^2\}$
    $pre(j) = argmin_{k=0,1,\cdots,j-1}\{P(k) + (200 - a_j + a_k)^2\}$
**end**
print($P(n)$)
$j = n$
**while** $pred(j)! = -1$ **do**
    print($j$)
    $j = pred(j)$;
**end**
```

! The above code outputs the path in reverse order.
(c) This algorithm fills in a one-dimensional table of length n, in left-to-right order. Each entry take $O(n)$ time, so the overall running time is $O(n^2)$.

Problem 3, Making Change
Given an unlimited supply of coins of denominations $x_1 < x_2 < \ldots < x_n$, we wish to make change for a given value v; that is, we want to find a set of coins whose total value is v. This might not be possible: for example, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an $O(nv)$ dynamic programming algorithm for the following problem:
Input: $x_1, x_2, \ldots, x_n$
Output: Yes or No, is it possible to make change for v using denominations $x_1, x_2, \ldots, x_n$?
Write explicitly (a)the recursive form of the solution (ie your thinking in words) (b)the pseudocode implementation and (d)the complexity analysis.

Answer:

(a) For each $j \in \{1, 2, \cdots, v\}$, let $A(j)$ be the boolean variable which takes true if it's possible to make change for j, false otherwise. The answer we seek is $A(v)$. For each j, if it's possible to make change for $j - x_k$ for some $k \in \{1, 2, ..., n\}(x_k < j)$, then it would be also possible to make change for j. So we can deduce that

$$A(j) = \bigvee_{k \in = \{1,2,\cdots,n\}, x_k \leq j} A(j - x_k)$$

For the initialization, we can set $A(0) = true$ which means that we don't need any change for no payment.
(b)

```
initialize $A(0) = true, A(j) = false$ for $j = 1 \cdots v$
for $j = 1$ to $v$ do
    for $k = 1$ to $n$ do
        if $x_k \leq j$ then
            $A(j) = A(j) \vee A(j - x_k)$
        end
    end
end
if $A(v) = true$ then
    print('Yes')
else
    print('No')
end
```

(c) This algorithm fills in a one-dimensional table of length v, in left-to-right order. Each entry take $O(n)$ time, so the overall running time is $O(vn)$.

Problem 4, Dynamic Programming, Making Change Optimally
Here is yet another variation on the change-making problem. Given an unlimited supply of coins of denominations $1 = x_1 < x_2 < \ldots < x_n$, we wish to make change for a value v using as few coins as possible (note that, since $x_1 = 1$, any value v is realizable.)
(a) Give an $O(nv)$ algorithm that finds the minimum number of coins, say $k(v)$, whose total value is v.
(b) Give an $O(nv)$ algorithm that finds a solution $s_1, \ldots, s_n$, where s_i denotes the number of coins of denomination x_i, $1 \leq i \leq n$, such that (a)$\sum_{i=1}^{n} s_i x_i = v$ and (b)$\sum_{i=1}^{n} s_i = k(v)$.
Write explicitly (a)the recursive form of the solution (ie your thinking in words) (b)the pseudocode implementation and (d)the complexity analysis.

Answer:

(a) For each $j \in \{1, 2, \cdots, v\}$, let $k(j)$ be the minimum number of coins necessary to make change for j The first answer we seek is $k(v)$. And the second answer can be obtained by keeping track of which coin is used last. For this we introduce $p(j)$, which tracks the type of coin that's used 'last' in the optimal plan.
As in the previous problem, we can express it as subproblems finding optimal plan.
We can deduce that

$$k(j) = min_{l=1,2,\cdots,n}\{k(j - x_l) + 1 : x_l \leq j\}.$$

To keep the trace of coin usage, we set

$$p(j) = argmin_{l=1,2,\cdots,n}\{k(j - x_l) + 1 : x_l \leq j\}.$$

What we need to get the optimal plan is just counting $p(j)$.
(b)

```
initialize $k(0) = 0, p(0) = -1, s(l) = 0$ for $l = 1, \cdots, n$
for $j = 1$ to $v$ do
    $k(j) = min_{l=1,2,\cdots,n}\{k(j - x_l) + 1 : x_l \leq j\}$
    $p(j) = argmin_{l=1,2,\cdots,n}\{k(j - x_l) + 1 : x_l \leq j\}$
end
print($k(v)$)
$j = v$
while $j > 0$ do
    $s(p(j)) = s(p(j)) + 1$
    $j = j - x_{p(j)}$
end
print($s_l : l = 1, \cdots, n$)
```

(c) This algorithm fills in a one-dimensional table of length v, in left-to-right order. Each entry take $O(n)$ time, so the overall running time is $O(vn)$.

Problem 5, Dynamic Programming, Longest Path out of Specific Vertex
Let $G(V, E)$ be a directed acyclic graph presented in topologically sorted order, that is, if $V = \{v_1, \ldots, v_n\}$, then $(v_i \to v_j \in E) \Rightarrow (i < j)$ - ie, all edges are directed from lower order vertices to higher order vertices. Give an $O(|V| + |E|)$ complexity alogithm that finds the length of the longest path that starts in v_1.
Write explicitly (a)the recursive form of the solution (ie your thinking in words) (b)the pseudocode implementation and (d)the complexity analysis.

Answer:

(a) For each $j \in \{1, 2, \cdots, n\}$, let $L(j)$ be the length of the longest path that starts from v_j. The answer we seek is $L(1)$. We have a finite number of vertices that can be the next of v_1, we can express the problem as subproblems. So we state that if v_k is the next of v_1 in the longest path, the subpath starting from v_k should be the longest path starting from v_k.
We can deduce that

$$L(j) = max\{L(k) + d(j,k) : (v_j, v_k) \in E\}.$$

where $d(j,k)$ is the length of the edge from v_j to v_k.
We set $L(n) = 0$ for initialization.
(b)

```
initialize $L(n) = 0$
**for** $j = n - 1$ *to* 1 **do**
 |  $L(j) = max\{L(k) + d(j,k) : (v_j, v_k) \in E\}$.
**end**
print($L(1)$)
```

(c) The for loop traverses every vertex and every edge only once. So the overall running time is $O(|E|+|V|)$.