COMP3711: Design and Analysis of Algorithms

Tutorial 9

Question 1

Suppose you want to make change for n (HK) dollars using the fewest number of coins. Assume that each coin's value is an integer.

Give an $O(nk)$-time dynamic programming algorithm that makes change for any set of k different coin denominations, assuming there is always the 1-dollar coin.

Solution 1

Let us define $c[j]$ to be the minimum number of coins we need to make change for j dollars. Let the coin denominations be $d_1, d_2, ..., d_k$. Since one of the coins is a one-dollar, there is a way to make change for any amount $j \geq 1$.

Solution 1

Because of the optimal substructure, if we knew that an optimal solution for the problem of making change for j dollars used a coin of denomination d_i, we would have $c[j] = 1 + c[j - d_i]$. As base cases, we have that $c[j] = 0$ for all $j \leq 0$.
To develop a recursive formulation, we have to check all denominations, giving

$$c[j] = \begin{cases} 0 & \text{if } j \leq 0, \\ 1 + \min\limits_{1 \leq i \leq k} \{c[j - d_i]\} & \text{if } j > 1. \end{cases}$$

Solution 1

We can compute the $c[j]$ values in order of increasing j by using a table. The following procedure does so, producing a table $c[1..n]$. It avoids even examining $c[j]$ for $j \leq 0$ by ensuring that $j \geq d_i$ before looking up $c[j - d_i]$. The procedure also produces a table $denom[1..n]$, where $denom[j]$ is the denomination of a coin used in an optimal solution to the problem of making change for j dollars.

COMPUTE-CHANGE(n, d, k)

for $j \leftarrow 1$ **to** n
    $c[j] \leftarrow \infty$
    for $i \leftarrow 1$ **to** k
        if $j \geq d_i$ and $1 + c[j - d_i] < c[j]$
            $c[j] \leftarrow 1 + c[j - d_i]$
            $denom[j] \leftarrow d_i$
return c and $denom$

Solution 1

This procedure obviously runs in $O(nk)$ time.
We use the following procedure to output the coins used in the optimal solution computed by COMPUTE-CHANGE:

GIVE-CHANGE($j, denom$)
if $j > 0$
    give one coin of denomination $denom[j]$
    GIVE-CHANGE($j - denom[j], denom$)

The initial call is GIVE-CHANGE($n, denom$). Since the value of the first parameter decreases in each recursive call, this procedure runs in $O(n)$ time.

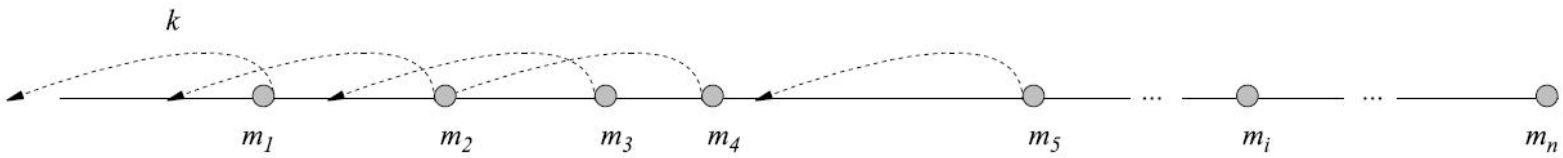
Question 2

KFCC is considering opening a series of restaurants along the Highway. The n possible locations are along a straight line, and the distances of these locations from the start of the Highway are, in miles and in increasing order: $m_1, m_2, \ldots, m_n$. The constraints are as follows:

1. At each location, KFCC may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
2. Any two restaurants should be at least k miles apart, where k is a positive integer.

Give a dynamic programming algorithm that determines the locations to open restaurants which maximizes the total expected profit.

Solution 2: Step 1: Space of Subproblems



We define $T[i]$ to be the total profits from the best valid configuration using locations $1, 2, \ldots, i$ only.

We also store $R[i]$ which is 1 if there is a restaurant at location i and 0 otherwise.

Case 1: Base case. If $i = 0$, then there is no location available to choose from to open a restaurant. So $T[0] = 0$.

Solution 2: Step 2: Recursive Formulation

Case 2: General case. If $i > 0$, then we have two options.

1. Do not open a restaurant at location i
   If we choose to not open a restaurant at location i, then the optimal value will come about by considering how to obtain total profits from the best valid configuration using the remaining location $1, 2, \ldots, i-1$. This is just $T[i-1]$.
2. Open a restaurant at location i
   If we open a restaurant at location i, then we gain the expected profit p_i. As we want to build a restaurant at location i, then the closest location to build another restaurant should be at c_i, where c_i denote the maximum j which $m_j \leq m_i - k$. To obtain a maximum profit, we need to obtain the maximum profits from the remaining locations $1, 2, \ldots, c_i$. This is just $p_i + T[c_i]$.

Solution 2: Step 2: Recursive Formulation

Since these are the only two possibilities, we can see that we have the following rule for constructing table T:

$$T[i] = \begin{cases} 0, \text{ if } i = 0 \\ \max\{T[i-1], p_i + T[c_i]\}, \text{ if } i > 0 \end{cases}$$

If $T[i] = T[i-1]$, then $R[i] = 0$; and $R[i] = 1$ otherwise.

Note: We can compute $c_i = \max\{j : m_j \leq m_i - k\}$ for every i. Note that for some values of i and c_i may not exist in which case, we assume that $c_i = 0$.

Solution 2: Step 3: Bottom-up Computation

Recurrence:

$$T[i] = \max\{T[i-1], p_i + T[c_i]\}$$

We compute and save $T[i]$ in such an order that: When it is time to compute $T[i]$, the values of $T[i-1]$ and $T[c_i]$ are available. So we will fill the table in an order of increasing i.

Solution 2

Algorithm to compute c_i for every i

Compute-ci$(m_1, \ldots, m_n, k)$

1: **for** $j = 1$ to n **do**
2: $\quad m'_j = m_j - k$
3: **end for**
4: $i = 1, j = 1$
5: **while** $i \leq n$ **do**
6: $\quad$ **if** $m'_i < m_j$ **then**
7: $\quad\quad c_i = j - 1$
8: $\quad\quad i++$
9: $\quad$ **else**
10: $\quad\quad j++$
11: $\quad$ **end if**
12: **end while**
13: **return** $c_1, \ldots, c_n$

Solution 2

Algorithm to find optimal profit and locations to open restaurants
Find-Optimal-Profit-And-Pos$(m_1, \ldots, m_n, p_1, \ldots, p_n, c_1, \ldots, c_n,)$

```
1:  T[0] = 0
2:  for i = 0 to n do
3:      Not-Open-At-i = T[i − 1]
4:      Open-At-i = p_i + T[c_i]
5:      if Not-Open-At-i > Open-At-i then
6:          T[i] =Not-Open-At-i
7:          R[i] =0
8:      else
9:          T[i] =Open-At-i
10:         R[i] =1
11:     end if
12: end for
13: return  T[n] and R;
```

Solution 2

Algorithm to report optimal locations to open restaurants

Report-Optimal-Locations$(R, c_1, \ldots, c_n,)$

```
1: j = n
2: S = ∅
3: while j ≥ 1 do
4:     if R[j] = 1 then
5:         Insert m_j into S;
6:         j = c_j
7:     else
8:         j − −
9:     end if
10: end while
11: return S;
```

Solution 2: Running Time Analysis:

- The **Compute-ci** takes $O(n)$ time to compute c_i for every i.
- The Find **Optimal-Profit-And-Pos** takes $O(n)$ time to compute T and R.
- The **Report-Optimal-Locations** takes $O(n)$ time to report the optimal locations for opening restaurants along the Highway.

Therefore, the overall running time is $O(n)$.