COMP3711: Design and Analysis of Algorithms

Tutorial 6

HKUST

Question 1

Give an $O(nk)$-time dynamic programming algorithm that makes change using the minimal possible number of coins.

The solution obviously depends upon the country you are in.

Let the local given coin denominations be

$$d_1 < d_2 < \cdots < d_k,$$

where $d_1 = 1$ (which guarantees that some solution always exists).

Solution 1

This problem has the optimal substructure property.

If an optimal solution for j dollars used **a** coin of denomination d_i,
=> change for the remaining $j - d_i$ dollars must be an optimal (minimum) set of coins for that subproblem, i.e.,

$$c[j] = 1 + c[j - d_i].$$

As base cases, we have that

$c[j] = \infty$ for all $j < 0$ and $c[0] = 0.$

$c[j] = \infty$ for $j < 0$, indicates that $j < 0$ is "impossible". It's not actually stored in a table.

The recurrence finds the best way to give change *assuming that first coin has value* d_j and then takes the minimum value over all possible j. This yields

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

Solution 1

The code uses the recurrence to evaluate the $c[j]$ by filling in a table $c[1 \cdots n]$, in increasing order of index j.

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

Note: code avoids explicitly calling $c[j]$ *for* $j < 0$,
by checking $j \geq d_i$ *before looking up* $c[j - d_i]$.

The procedure also produces a table *denom*$[1 \cdots n]$, where *denom*$[j]$ is the denomination of the "first" coin used in an optimal solution to the problem of making change for j dollars.

```
COMPUTE-CHANGE$(n, d, k)$
$c[0] = 0$
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

```
COMPUTE-CHANGE$(n, d, k)$
$c[0] = 0$
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 $denom$
```

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

This procedure obviously runs in $O(nk)$ time.

Solution 1

```
COMPUTE-CHANGE($n, d, k$)
$c[0] = 0$
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 $denom$
```

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

The procedure below outputs the coins used in the optimal solution computed by COMPUTE-CHANGE:

```
GIVE-CHANGE ($j, denom$)
if $j > 0$
    output 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.

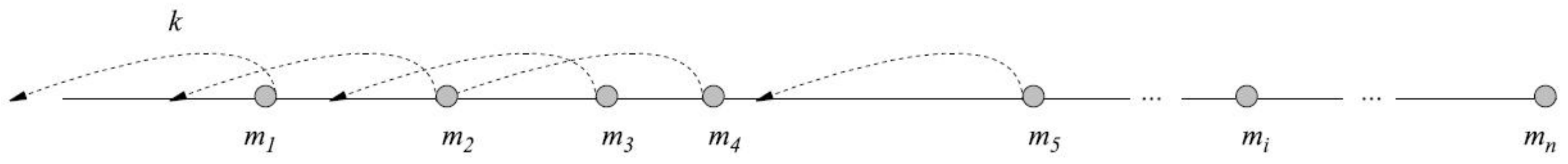
The n available locations are along a straight line; the distances of these locations from the start of the Highway are given in miles and in increasing order: $m_1, m_2, \cdots, m_n$. The constraints are as follows:

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 $i = 1,2,\cdots,n$.

② Any two restaurants should be at least k miles apart, where k is a given positive integer.

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

Solution 2: Step 1: Space of Subproblems



Define $T[i]$ to be the total profit from the best valid configuration using only locations $1,2,\cdots,i$.

Define
$R[i]=1$ if there is a restaurant at location i in the (chosen) best valid configuration using only locations $1,2,\cdots,i$
$R[i]=0$, otherwise.

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

Solution 2: Step 2: Recursive Formulation

Case 2: General case. If $i > 0$, there are two options.

1. Do not open a restaurant at location i

If no restaurant opened at location i , then the optimal value will be optimal profit from valid configuration using only location $1,2,\cdots,i-1$.
=> $\mathrm{T[i]} = T[i-1]$.

2. Do open a restaurant at location i

Opening a restaurant at location i, gives expected profit p_i .
After building at location i, the nearest location to the left at which a restaurant could be built is c_i , where

$$c_i = \max\{j \le i : m_j \le m_i - k\}.$$

Obtaining a maximum profit requires obtaining maximum profits from the remaining locations $1,2,\cdots,c_i$.
=> $\mathrm{T[i]} = p_i + T[c_i]$.

Solution 2: Step 2: Recursive Formulation

Since these are the only two possibilities, the recurrence is:

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

If $T[i] = T[i-1]$, => $R[i] = 0$;
If $T[i] \neq T[i-1]$, => $R[i] = 1$.

Note: Similar to the weighted interval problem in class, we could evaluate

$$c_i = \max\{j : m_j \leq m_i - k\}$$

in $O(\log n)$ time using binary search.

This would give a $O(n \log n)$ algorithm. We will see soon that it's possible to calculate all of the c_i in $O(n)$ time, permitting an $O(n)$ algorithm.

Note that for some values of i, c_i may not exist:
in which case we set $c_i = 0$.

Solution 2: Algorithm

Algorithm to find optimal profit and locations to open restaurants.

Assumes c_i are known.

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

Find-Optimal-Profit-And-Pos$(m_1, \cdots, m_n, p_1, \cdots, p_n, c_1, \cdots, c_n)$

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

Solution 2: Algorithm for finding c_i

Algorithm to compute $c_i = \max\{j : m_j \le m_i - k\}$ for every i

Uses fact that $0 = c_1 \le c_2 \le \cdots \le c_n < n$.

Starts looking for c_i at $j = c_{i-1}$;
increments j until finds correct value of c_i.

Note that algorithm runs in $O(n)$ time because line 5 can only be implemented $O(n)$ times!

```
Compute-$c_i$($m_1, \cdots, m_n, k$)
1.  $c_1 = 0$
2.  for $i = 2$ to $n$ do
3.      $j = c_{i-1}$
4.      while $(m_{j+1} \le m_i - k)$ do
5.          $j = j + 1$
6.      end while
7.      $c_i = j$
8.  end for
9.  return $c_1, \cdots, c_n$
```

Solution 2: Location Reporting

Algorithm to report optimal locations to open restaurants

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

```
1.  $j = n; S = \emptyset$
2.  while $j \geq 1$ do
3.      if $R[j] = 1$ then
4.          Insert $m_j$ into $S$; $j = c_j$
5.      else
6.          $j = j - 1$
7.      end if
8.  end while
9.  return $S$;
```

Solution 2: Time Analysis:

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

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