Laboratório de Programação Avançada 2018/19
Week 5 – Dynamic Programming

Universidade de Coimbra

Outline

1. Introduction
2. Coin Changing
3. Subset sum
4. Knapsack

Problem decomposition

- A problem may be decomposed in a sequence of nested sub-problems
- The original problem is solved by combining the solutions to the various sub-problems
- The choices made at the inner levels influence the choices to be made at the outer levels (in general)

- Coin changing: What is the minimum number of coins to make a change for C with n coin denominations? (assume infinite coins for each denomination)

- Subset sum problem: Is there a subset of coins that sums to C? (assume a finite set of n coins)

- Knapsack problem: I have n objects. Each object has a given weight and value. My knapsack can only carry W Kgs. Which objects should I pick that maximize the value and fit into the knapsack?

Coin Changing

What is minimum number of coins for a given change C?

Change 36 Euros with coin denominations 1, 5, 10, 20.

1. 36 - 20 = 16
2. 16 - 10 = 6
3. 6 - 5 = 1
4. 1 - 1 = 0

This is a greedy algorithm but it does not work with an arbitrary coin denominations.

Coin Changing

A counter-example for the greedy algorithm:

Change 30 Euros with coin denominations 1, 10, 25.

1. $30 - 25 = 5$
2. $5 - 1 = 4$
3. $4 - 1 = 3$
4. $3 - 1 = 2$
5. $2 - 1 = 1$
6. $1 - 1 = 0$

This totals 6 coins, but we could have used 3 coins of 10!

Coin Changing

Sub-problem

- Find the change for $C' \leq C$ with minimum number of coins using the first $i \leq n$ coin denominations.

Optimal substructure

1. If the optimal solution for the problem above contains a coin with denomination i, then by removing it, we have an optimal solution for the change without that coin. (We prove this in the following.)
2. If the optimal solution for the problem above does not contain a coin with denomination i, then we have an optimal solution for the same change for the first $i - 1$ denominations.

1. Let S be the set with the minimum number of coins to change C', taken from the first i denominations, and using a coin with denomination d_i.
2. Then, S without that coin is optimal for $C' - d_i$.

Sketch of the proof (by contradiction)

- (negate 2.) Assume that you can find a change for $C' - d_i$ with less coins using the first i denominations.
- (contradict 1.) Then, it is also possible to change C' with less coins by adding a coin with denomination d_i.

Recursive approach

For denomination d_i:

1. Use denomination d_i and make the change for $C' - d_i$ with the denominations available (including d_i) or
2. Do not use denomination d_i, and make the change for C' with the remaining denominations.
3. Choose the minimum of the two.

A first recursive solution:

Function *change*(i, C)
  if $C > 0$ and $i = 0$ **then** {1st base case - change > 0 and }
    return ∞ {no more denominations}
  if $C = 0$ **then** {2nd base case - change is 0}
    return 0
  if $d_i > C$ **then**
    return $\underbrace{change(i-1, C)}_{\text{don't take denom. } d_i}$
  else
    return $\min(\underbrace{change(i-1, C)}_{\text{don't take denom. } d_i}, \underbrace{1 + change(i, C - d_i)}_{\text{take denom. } d_i})$

The number of recursive calls is exponential.

Can we do memoizing?

A top-down dynamic programming solution:

Function *change*(i, C)
  if $C > 0$ and $i = 0$ **then** {1st base case - change > 0 and }
    return ∞ {no more denominations}
  if $C = 0$ **then** {2nd base case - change is 0}
    return 0
  if $T[i, C] > 0$ **then**
    return $T[i, C]$
  if $d_i > C$ **then**
    $T[i, C] = change(i - 1, C)$
  else
    $T[i, C] = \min(change(i - 1, C), 1 + change(i, C - d_i))$
  return $T[i, C]$

Table T stores the minimum number of coins for the first i denominations and each change C

Example:

Change 12 Euros with coin denominations 1, 6, 10.

C	0	1	2	3	4	5	6	7	8	9	10	11	12
$T[0, C]$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$T[1, C]$	0	1	2	3	4	5	6	7	8	9	10	11	12
$T[2, C]$	0	1	2	3	4	5	1	2	3	4	5	6	2
$T[3, C]$	0	1	2	3	4	5	1	2	3	4	1	2	2

Can we do bottom-up DP? What are the base cases?

Can we order the computations?

Bottom-up dynamic programming:

Function *change*(n, C)
    for $i = 0$ **to** n **do**
        $T[i, 0] = 0$
    for $j = 0$ **to** C **do**
        $T[0, j] = \infty$
    for $i = 1$ **to** n **do**
        for $j = 1$ **to** C **do**
            if $d_i > j$ **then**
                $T[i, j] = T[i - 1, j]$
            else
                $T[i, j] = \min(T[i - 1, j], 1 + T[i, j - d_i])$
    return $T[n, C]$

The time complexity is $O(nC)$, which is *pseudo-polynomial*.

Subset Sum

- Suppose you want to know if there exists a subset S of a set of n coins that makes the change for C (decision problem).

- This is know as the Subset Sum problem and it sounds similar to Coin Changing. Does it also have optimal substructure?

Sub-problem

- Find whether it is possible to have a change for $C' \leq C$ using the first $i \leq n$ coins.
- Let S be a subset of coins, taken from the first i coins, that make change for C'.

Optimal substructure

- If S contains the i-th coin, then by removing it from S, we obtain a subset of coins for the change without that coin for the first $i - 1$ coins.
- If S does not contain the i-th coin, then it also makes the same change for the first $i - 1$ coins.

Why? (very trivial!)

Subset Sum

Recursion

- Choose the i-th coin:
    1. Either use it and solve sub-problem for $C - d_i$ with the remaining $i - 1$ coins, or
    2. Do not use it and solve sub-problem for C with the remaining $i - 1$ coins

A simple recursive solution:

Function *subset*(i, C)
  if $i = 0$ and $C \neq 0$ **then** {1st base case - no more coins and}
    return false {change is not 0}
  if $C = 0$ **then** {2nd base case - change is 0}
    return true
  if $d_i > C$ **then**
    return $\underbrace{subset(i-1, C)}_{\text{don't take the } i\text{-th coin}}$
  else
    return $\underbrace{subset(i-1, C)}_{\text{don't take the } i\text{-th coin}} \vee \underbrace{subset(i-1, C-d_i)}_{\text{take the } i\text{-th coin}}$

It is an exponential approach. Can we do memoizing?

A top-down dynamic programming:

```
Function subset(i, C)
  if i = 0 and C ≠ 0 then                {1st base case - no more coins and}
    return false                                         {change is not 0}
  if C = 0 then                          {2nd base case - change is 0}
    return true
  if T[i, C] is not empty then
    return T[i, C]
  if d_i > C then
    T[i, C] = subset(i − 1, C)
  else
    T[i, C] = subset(i − 1, C) ∨ subset(i − 1, C − d_i)
  return T[i, C]
```

Table T stores whether there is change or not with the first i coins.

Example:

Coins $= \{2, 6, 10\}$ and $C = 12$.

C	0	1	2	3	4	5	6	7	8	9	10	11	12
$T[0, C]$	T	F	F	F	F	F	F	F	F	F	F	F	F
$T[1, C]$	T	F	T	F	F	F	F	F	F	F	F	F	F
$T[2, C]$	T	F	T	F	F	F	T	F	T	F	F	F	F
$T[3, C]$	T	F	T	F	F	F	T	F	T	F	T	F	T

Can we do bottom-up DP? What are the base cases?

Can we order the computation?

Bottom-up dynamic programming:

Function *subset*(n, C)
  for $i = 0$ **to** n **do** {1st base case}
    $T[i, 0] = \texttt{true}$
  for $j = 1$ **to** C **do** {2nd base case}
    $T[0, j] = \texttt{false}$
  for $i = 1$ **to** n **do**
    for $j = 1$ **to** C **do**
      if $d_i > j$ **then**
        $T[i, j] = T[i-1, j]$
      else
        $T[i, j] = T[i-1, j] \vee T[i-1, j-d_i]$
  return $T[n, C]$

Also pseudo-polynomial since its time complexity is $O(nC)$.