Workshop #4
CSCI 432

Making Change

In general, suppose a country has denominations:

$$1 = d_1 < d_2 < \cdots < d_k$$

(US coins: $d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25$)

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

So, we'll check all possibilities.

But we don't know what actual coin is in the optimal solution.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

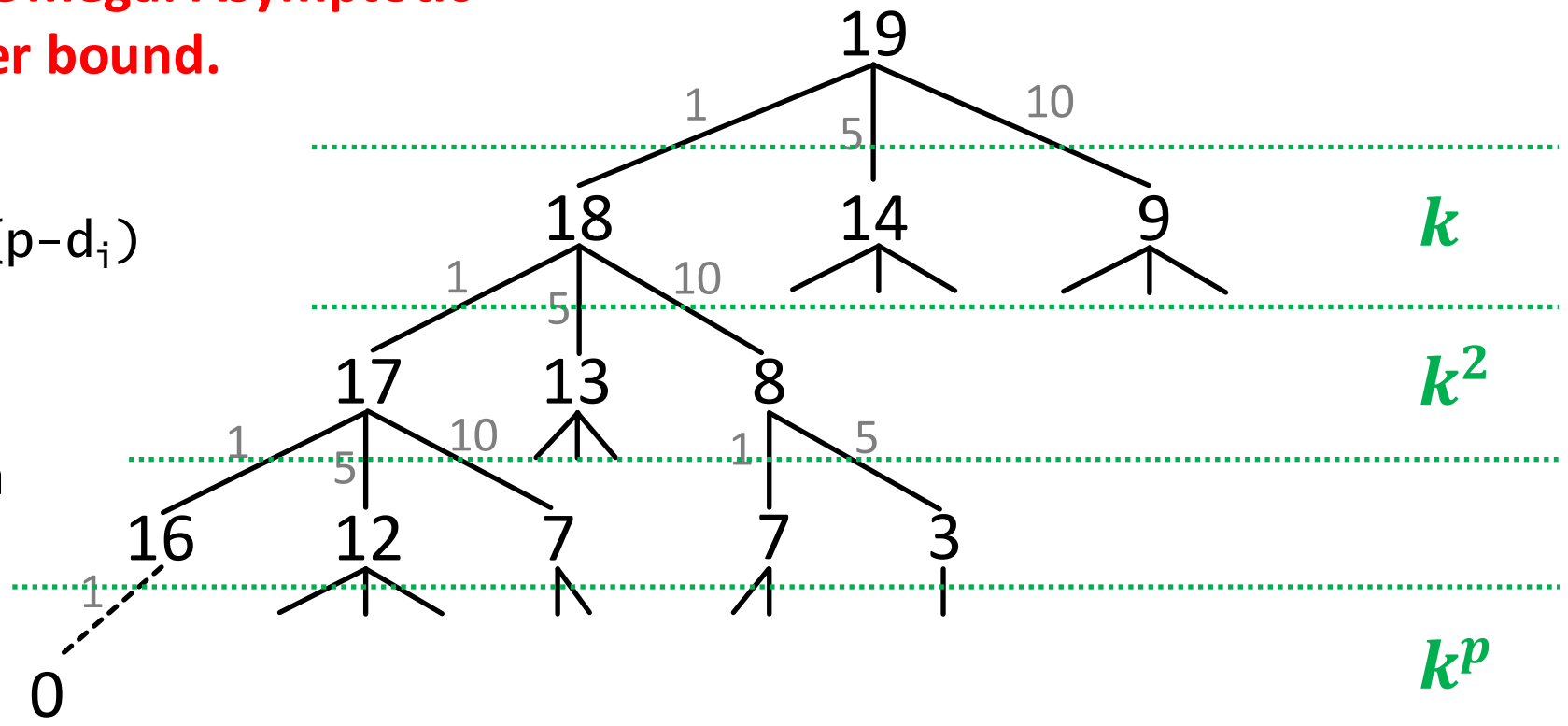
$O(k^p) \in \Omega(\text{Bad})$

Big-Omega: Asymptotic lower bound.

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

p = value to make change for



Making Change – Dynamic Programing

Plan (for value p):

1. Calculate optimal change for each value less than p.
2. Save those values.
3. Apply formula to find optimal change for p.

If I know the best way to make change for \$0.01 - \$0.32, how can I figure out the best way to make change for \$0.33?

$$\text{change}(\$0.33) = \min \begin{cases} \text{change}(\$0.08) + 1 \\ \text{change}(\$0.23) + 1 \\ \text{change}(\$0.28) + 1 \\ \text{change}(\$0.32) + 1 \end{cases}$$

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

array holding optimal values for all values $\leq p$.

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

Making Change – Dynamic Programing

```
changeDP(p)
    Chng[0,...,p] = [0,...,0]
    for m = 1 to p
        min = ∞
        for d_i ≤ m
            if Chng[m − d_i] + 1 < min
                min = Chng[m − d_i] + 1
        Chng[m] = min
    return Chng[p]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

for each denomination value.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

for each denomination value.

check optimal value of $m - d_i$

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Suppose we are filling this out for $m = 13$.

$d_1 = 10$:

$d_2 = 5$:

$d_3 = 1$:

Chng[]=?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for di ≤ m
         if Chng[m − di] + 1 < min
            min = Chng[m − di] + 1
      Chng[m] = min
   return Chng[p]
```

Suppose we are filling this out for $m = 13$.

$d_i = 10$:
Chng[13−10]+1 = 4

$d_i = 5$:
Chng[13−5]+1 = 5

$d_i = 1$:
Chng[13−1]+1 = 4

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Suppose we are filling this out for $m = 13$.

$d_i = 10$:
Chng[13−10]+1 = 4

$d_i = 5$:
Chng[13−5]+1 = 5

$d_i = 1$:
Chng[13−1]+1 = 4

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	**4**	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
changeDP(p)
    Chng[0,...,p] = [0,...,0]
    for m = 1 to p
        min = ∞
        for d_i ≤ m
            if Chng[m − d_i] + 1 < min
                min = Chng[m − d_i] + 1
        Chng[m] = min
    return Chng[p]
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	3	4	5	6	2

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Running time?

k = # denominations

p = value to make change for

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Running time?

Outer for loop $\in O(p)$

Inner for loop $\in O(k)$

Total $\in O(pk)$

k = # denominations

p = value to make change for

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Running time?

- Size of array $\in O(p)$
- Checks per spot $\in O(k)$
- Time per check $\in O(1)$
- Total $\in O(pk)$

k = # denominations

p = value to make change for

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	3	4	5	6	2

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

What if we want the actual coin values for the optimal solution?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	3	4	5	6	2

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
      Chng[m] = min
   return Chng[p]
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞, coin = 0
      for d_i ≤ m
         if Chng[m - d_i] + 1 < min
            min = Chng[m - d_i] + 1
      Chng[m] = min
   return Chng[p]
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞, coin = 0
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
            coin = d_i
      Chng[m] = min
   return Chng[p]
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞, coin = 0
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
            coin = d_i
      Chng[m] = min
      lastCoin[m] = coin
   return lastCoin
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞, coin = 0
      for d_i ≤ m
         if Chng[m – d_i] + 1 < min
            min = Chng[m – d_i] + 1
            coin = d_i
      Chng[m] = min
      lastCoin[m] = coin
   return lastCoin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

Making Change – Dynamic Programing

```
remain = 0
Set<Integer> coins
while remain > 0
   coin = lastCoin[remain]
   coins.add(coin)
   remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 13

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 13

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 3

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 3

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 2

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 2

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 1

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 1

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer>
while remain
    coin = last
    coins.add(coin)
    remain = remain - coin
```

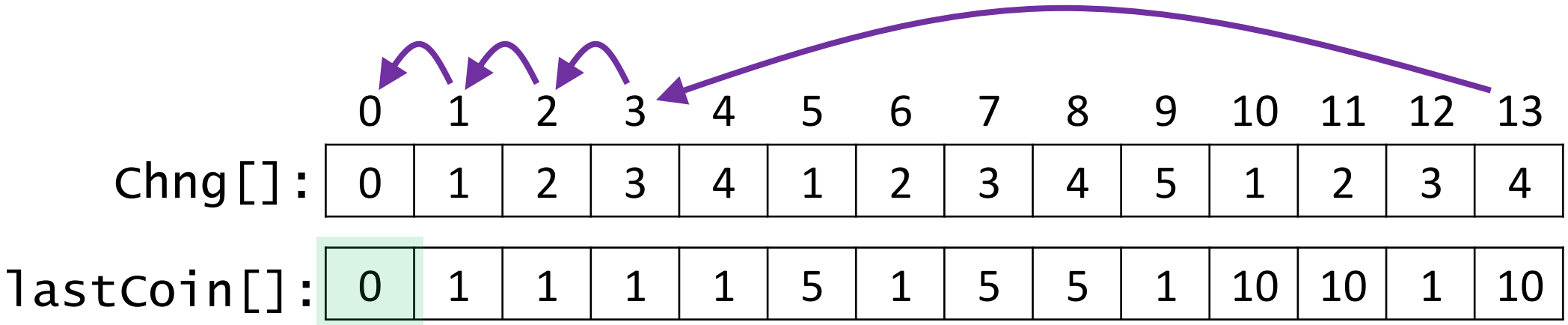
remain = 0

lastCoin is an array of the final coins added to al solutions.

ve get all of the he solution?

This is called backtracking, where we reconstruct the optimal solution from the value of the optimal solution.

13 = 10 + 1 + 1 + 1



```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

$O(k^p)$

VS.

```
changeDP(p)
    Chng[0,...,p] = [0,...,0]
    for m = 1 to p
        min = ∞
        for d_i ≤ m
            if Chng[m - d_i] + 1 < min
                min = Chng[m - d_i] + 1
        Chng[m] = min
    return Chng[p]
```

$O(pk)$

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Independent sub-problems

VS.

```
changeDP(p)
    Chng[0,...,p] = [0,...,0]
    for m = 1 to p
        min = ∞
        for d_i ≤ m
            if Chng[m - d_i] + 1 < min
                min = Chng[m - d_i] + 1
        Chng[m] = min
    return Chng[p]
```

Memoizing intermediate results.

```
Chng[0,...,p] = [e,...,e]
change(p)
    if p == 0
        Chng[p] = 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            if Chng[p-d_i] = e
                a = change(p-d_i)
            else
                a = Chng[p-d_i]
            if a < min
                min = a
        Chng[p] = min
    return 1 + min
```

VS.

```
changeDP(p)
    Chng[0,...,p] = [0,...,0]
    for m = 1 to p
        min = ∞
        for d_i ≤ m
            if Chng[m - d_i] + 1 < min
                min = Chng[m - d_i] + 1
        Chng[m] = min
    return Chng[p]
```