Dynamic Programming - 1

Debarka Sengupta

Origin

"Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities"

- Richard E. Bellman

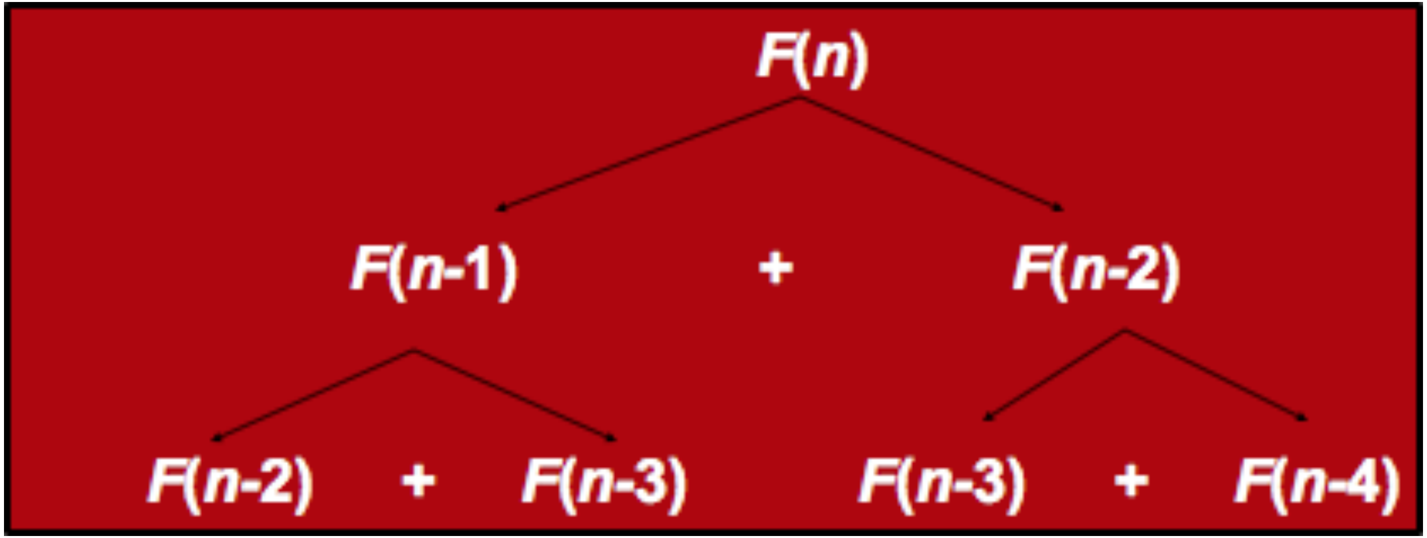
Origin

- A method for solving complex problems by breaking them into smaller, easier, sub problems
- Term *Dynamic Programming* coined by mathematician Richard Bellman in early 1950s
  - employed by Rand Corporation
  - Rand had many, large military contracts
  - Secretary of Defense, Charles Wilson "against research, especially mathematical research"
  - how could anyone oppose "dynamic"?

Fibonacci

- Computing the n^{th} Fibonacci number recursively:
  - $F(n) = F(n-1) + F(n-2)$
  - $F(0) = 0$
  - $F(1) = 1$
  - Top-down approach

Recursive calls



Naive recursive solution

naive_fibo(n):

if $n = 0$: return 0

else if $n = 1$: return 1

else: return naive_fibo($n - 1$) + naive_fibo($n - 2$).

Failing spectacularly

```
1th fibonnaci number: 1 -  Time: 4.467E-6
2th fibonnaci number: 1 -  Time: 4.47E-7
3th fibonnaci number: 2 -  Time: 4.46E-7
4th fibonnaci number: 3 -  Time: 4.46E-7
5th fibonnaci number: 5 -  Time: 4.47E-7
6th fibonnaci number: 8 -  Time: 4.47E-7
7th fibonnaci number: 13 -  Time: 1.34E-6
8th fibonnaci number: 21 -  Time: 1.787E-6
9th fibonnaci number: 34 -  Time: 2.233E-6
10th fibonnaci number: 55 -  Time: 3.573E-6
11th fibonnaci number: 89 -  Time: 1.2953E-5
12th fibonnaci number: 144 -  Time: 8.934E-6
13th fibonnaci number: 233 -  Time: 2.9033E-5
14th fibonnaci number: 377 -  Time: 3.7966E-5
15th fibonnaci number: 610 -  Time: 5.0919E-5
16th fibonnaci number: 987 -  Time: 7.1464E-5
17th fibonnaci number: 1597 -  Time: 1.08984E-4
```

Failing spectacularly

```
36th fibonnaci number: 14930352 -  Time: 0.045372057
37th fibonnaci number: 24157817 -  Time: 0.071195386
38th fibonnaci number: 39088169 -  Time: 0.116922086
39th fibonnaci number: 63245986 -  Time: 0.186926245
40th fibonnaci number: 102334155 -  Time: 0.308602967
41th fibonnaci number: 165580141 -  Time: 0.498588795
42th fibonnaci number: 267914296 -  Time: 0.793824734
43th fibonnaci number: 433494437 -  Time: 1.323325593
44th fibonnaci number: 701408733 -  Time: 2.098209943
45th fibonnaci number: 1134903170 -  Time: 3.392917489
46th fibonnaci number: 1836311903 -  Time: 5.506675921
47th fibonnaci number: -1323752223 -  Time: 8.803592621
48th fibonnaci number: 512559680 -  Time: 14.295023778
49th fibonnaci number: -811192543 -  Time: 23.030062974
50th fibonnaci number: -298632863 -  Time: 37.217244704
51th fibonnaci number: -1109825406 -  Time: 60.224418869
```

Analysis

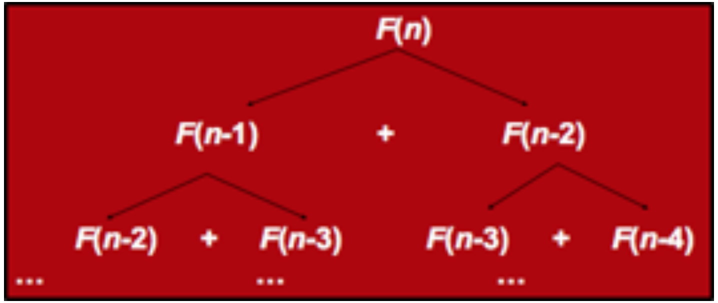
- What is the Recurrence relationship?
    - $T(n) = T(n-1) + T(n-2) + 1$
- What is the solution to this?
    - Clearly it is $O(2^n)$, but this is not tight.
    - A lower bound is $\Omega(2^{n/2})$.
    - You should notice that $T(n)$ grows very similarly to $F(n)$, so in fact $T(n) = \Theta(F(n))$.
- Obviously not very good, but we know that there is a better way to solve it!

Computing Fibonacci using bottom-up

- Computing the n^{th} Fibonacci number using a bottom-up approach:
  - F(0) = 0
  - F(1) = 1
  - F(2) = 1+0 = 1
  - …
  - F(n-2) =
  - F(n-1) =
  - F(n) = F(n-1) + F(n-2)
- Efficiency:
  - Time – O(n)
  - Space – O(n)

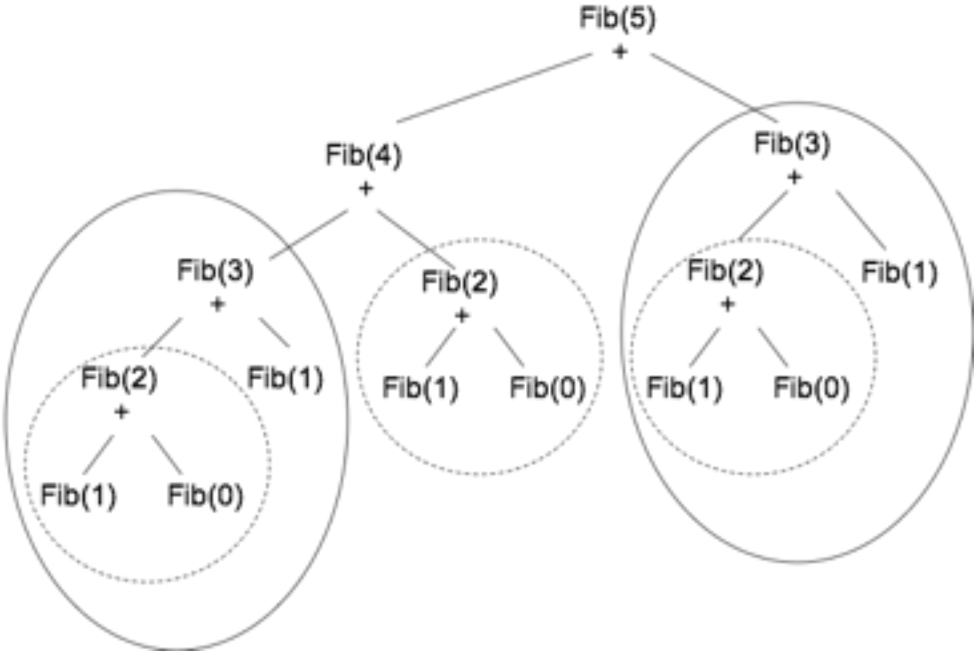
Inefficiency of the recursive solution

- The bottom-up approach is only Θ(n).
- Why is the top-down so inefficient?
  - Recomputes many sub-problems.
    - Check out F(n-2) and F(n-3)
- When calculating the 40th Fibonacci number the algorithm calculates the 4th Fibonacci number 24,157,817 times!!!



Overlapping subproblems - the magnitude

Fibonacci Numbers



Memoization

memo = { }

fib(n):

    if n in memo: return memo[n]

    else if $n = 0$: return 0

    else if $n = 1$: return 1

    else: $f = \text{fib}(n-1) + \underbrace{\text{fib}(n-2)}_{\text{free of charge!}}$

    memo[n] = f

    return f

Fast

```
1th fibonnaci number: 1 -  Time: 4.467E-6
2th fibonnaci number: 1 -  Time: 4.47E-7
3th fibonnaci number: 2 -  Time: 7.146E-6
4th fibonnaci number: 3 -  Time: 2.68E-6
5th fibonnaci number: 5 -  Time: 2.68E-6
6th fibonnaci number: 8 -  Time: 2.679E-6
7th fibonnaci number: 13 -  Time: 3.573E-6
8th fibonnaci number: 21 -  Time: 4.02E-6
9th fibonnaci number: 34 -  Time: 4.466E-6
10th fibonnaci number: 55 -  Time: 4.467E-6
11th fibonnaci number: 89 -  Time: 4.913E-6
12th fibonnaci number: 144 -  Time: 6.253E-6
13th fibonnaci number: 233 -  Time: 6.253E-6
14th fibonnaci number: 377 -  Time: 5.806E-6
15th fibonnaci number: 610 -  Time: 6.7E-6
16th fibonnaci number: 987 -  Time: 7.146E-6
17th fibonnaci number: 1597 -  Time: 7.146E-6
```

Fast

```
45th fibonnaci number: 1134903170 -  Time: 1.7419E-5
46th fibonnaci number: 1836311903 -  Time: 1.6972E-5
47th fibonnaci number: 2971215073 -  Time: 1.6973E-5
48th fibonnaci number: 4807526976 -  Time: 2.3673E-5
49th fibonnaci number: 7778742049 -  Time: 1.9653E-5
50th fibonnaci number: 12586269025 -  Time: 2.01E-5
51th fibonnaci number: 20365011074 -  Time: 1.9207E-5
52th fibonnaci number: 32951280099 -  Time: 2.0546E-5
```

```
67th fibonnaci number: 44945570212853 -  Time: 2.3673E-5
68th fibonnaci number: 72723460248141 -  Time: 2.3673E-5
69th fibonnaci number: 117669030460994 -  Time: 2.412E-5
70th fibonnaci number: 190392490709135 -  Time: 2.4566E-5
71th fibonnaci number: 308061521170129 -  Time: 2.4566E-5
72th fibonnaci number: 498454011879264 -  Time: 2.5906E-5
73th fibonnaci number: 806515533049393 -  Time: 2.5459E-5
74th fibonnaci number: 1304969544928657 -  Time: 2.546E-5
```

Runtime

- Memoization does not invoke redundant calls
- Therefore does exactly same operations as bottom up
- Complexity is same as bottom-up i.e., O(n)

When to use DP

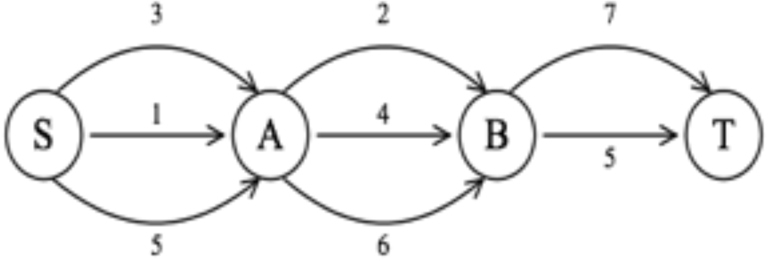
Dynamic Programming is an algorithm design method that can be used when the solution to a problem may be viewed as the result of a sequence of decisions.

Shortest path in a multi-stage graph

- To find a shortest path in a multi-stage graph

- Apply the greedy method: the shortest path from S to T:

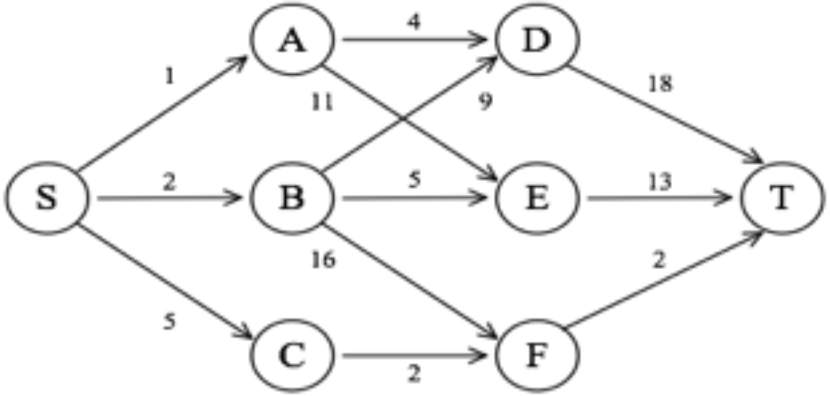
  1 + 2 + 5 = 8.



Example

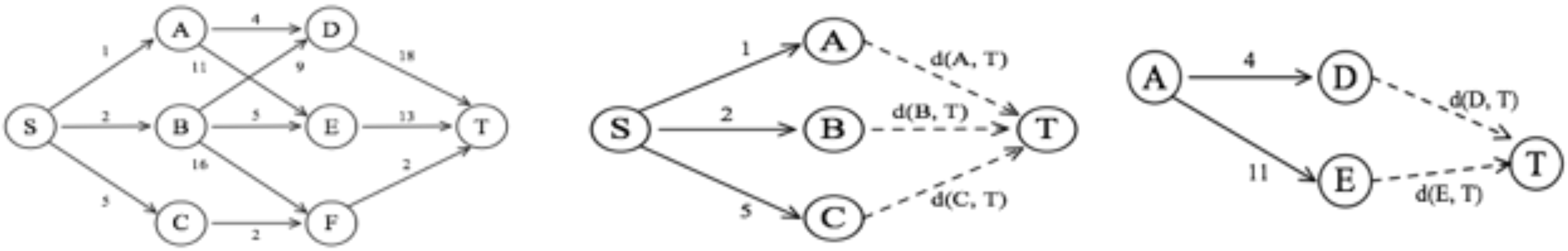
- e.g. The greedy method cannot be applied to this case: (S, A, D, T) 1 + 4 + 18 = 23.
- The real shortest path is:

  (S, C, F, T) 5 + 2 + 2 = 9.



Dynamic programming approach

- Top down
- d(S, T) = min{1+d(A, T), 2+d(B, T), 5+d(C, T)}
- d(A,T) = min{4+ d(D,T), 11+d(E,T)} = min{4+18, 11+13} = 22.



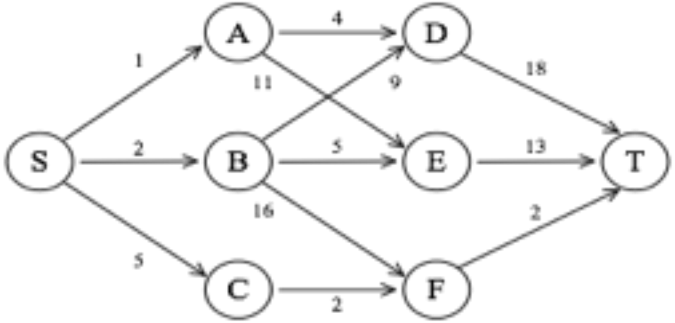
Continued ...

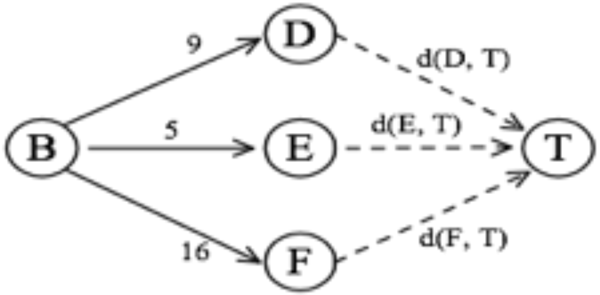
- $d(B, T) = \min\{9+d(D, T), 5+d(E, T), 16+d(F, T)\}$

  $= \min\{9+18, 5+13, 16+2\} = 18.$

- $d(C, T) = \min\{ 2+d(F, T) \} = 2+2 = 4$
- $d(S, T) = \min\{1+d(A, T), 2+d(B, T), 5+d(C, T)\}$

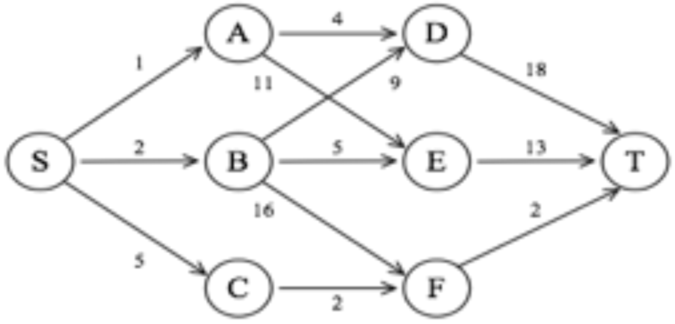
  $= \min\{1+22, 2+18, 5+4\} = 9.$





Bottom up

- d(S, A) = 1; d(S, B) = 2; d(S, C) = 5
- d(S,D)=min{d(S, A)+d(A, D),d(S, B)+d(B, D)} = min{ 1+4, 2+9 } = 5
- d(S,E)=min{d(S, A)+d(A, E),d(S, B)+d(B, E) = min{ 1+11, 2+5 } = 7
- d(S,F)=min{d(S, A)+d(A, F),d(S, B)+d(B, F)} = min{ 2+16, 5+2 } = 7

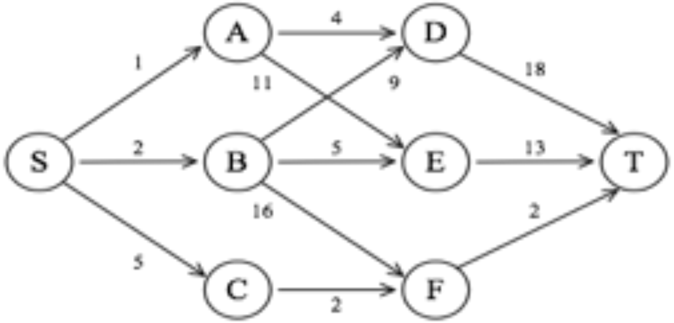


Continued ...

- d(S,T) = min{d(S, D)+d(D, T),d(S,E) + d(E,T), d(S, F)+d(F, T)}

  = min{ 5+18, 7+13, 7+2 }

  = 9



Optimal substructure

- Principle of optimality: Suppose that in solving a problem, we have to make a sequence of decisions $D_1, D_2, \ldots, D_n$. If this sequence is optimal, then the last k decisions, $1 < k < n$ must be optimal.

- e.g. in the shortest path problem, if $i, i_1, i_2, \ldots, j$ is a shortest path from i to j, then $i_1, i_2, \ldots, j$ must be a shortest path from i_1 to j

- In summary, if a problem can be described by a multi-stage graph, then it can be solved by dynamic programming.

DP when

- Optimal substructure
- Overlapping subproblems

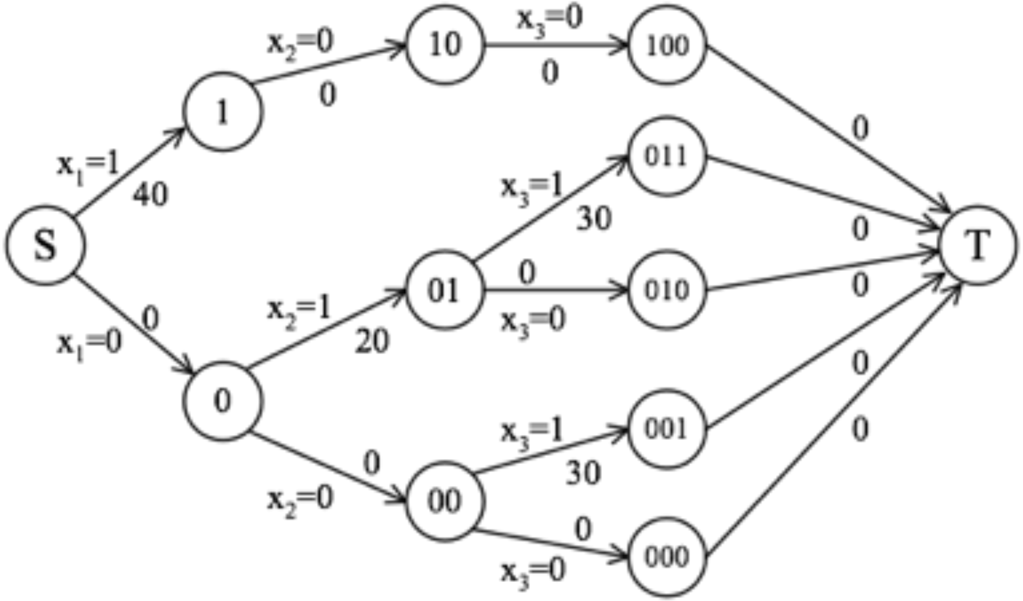
Binary knapsack

Take as many item as you can with weight limit.

i	W_i	P_i
1	10	40
2	3	20
3	5	30

$M=10$

Same as MSG


S
T
1
0
10
01
00
100
011
010
001
000
x1=1
40
x1=0
0
x2=0
0
x2=1
20
x2=0
0
x3=0
0
x3=1
30
0
x3=0
x3=1
30
0
x3=0
0
0
0
0
0

0/1 Knapsack - example

1	2	3	4
$V_1 = 5$ $W_1 = 5$	$V_2 = 4$ $W_2 = 6$	$V_3 = 7$ $W_3 = 8$	$V_4 = 7$ $W_4 = 4$

Greedy approach picks items 1 and 4

3 and 4 is better

Doesn't work!

Max = 13

Optimization

How to optimize?

Try all possibilities?

$O(2^n)$

Optimal substructure

- To show this for the 0-1 problem, consider the most valuable load weighing at most W pounds
  - *If we remove item j from the load, what do we know about the remaining load?*
  - A: remainder must be the most valuable load weighing at most $W - w_j$ that thief could take from museum, excluding item j

Recurrence

c[i][M] = value of solution for items 1...i with max weight M

$$
c[i][M] = \begin{cases} 0 & \text{If } i=0, M=0 \\ c[i-1][M] & \text{If } w_i > M \\ \max(v_i + c[i-1][M-w_i], c[i-1][M]) & \text{If } i > 0 \text{ and } w_i < M \end{cases}
$$

An instance

1	2	3	4
$V_1=5$	$V_2=4$	$V_3=7$	$V_4=7$
$W_1=5$	$W_2=6$	$W_3=8$	$W_4=4$

Max = 13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	5	5	5	5	9	9	9
3	0	0	0	0	0	5	5	5	7	7	7	9	9	12
4	0	0	0	0	7	7	7	7	7	12	12	12	14	14

Tracking the solution

1	2	3	4
$V_1 = 5$ $W_1 = 5$	$V_2 = 4$ $W_2 = 6$	$V_3 = 7$ $W_3 = 8$	$V_4 = 7$ $W_4 = 4$

Max = 13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	5	5	5	5	9	9	9
3	0	0	0	0	0	5	5	5	7	7	7	9	9	12
4	0	0	0	0	7	7	7	7	7	12	12	12	14	14

Solution: {4, 3}

Complexity

Done!

O(n*M)