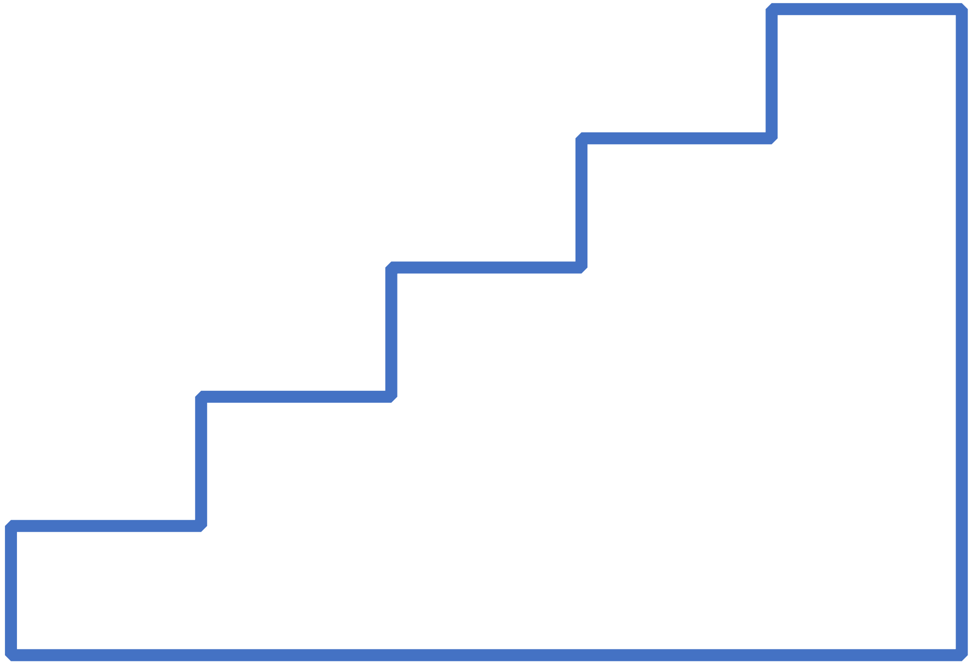
動的計画法(DP)を使って
無駄な計算を省こう

DPが使用できる条件

1. 部分構造最適性 : 問題の最適解がその内部に, その部分問題に対する最適解を含む.

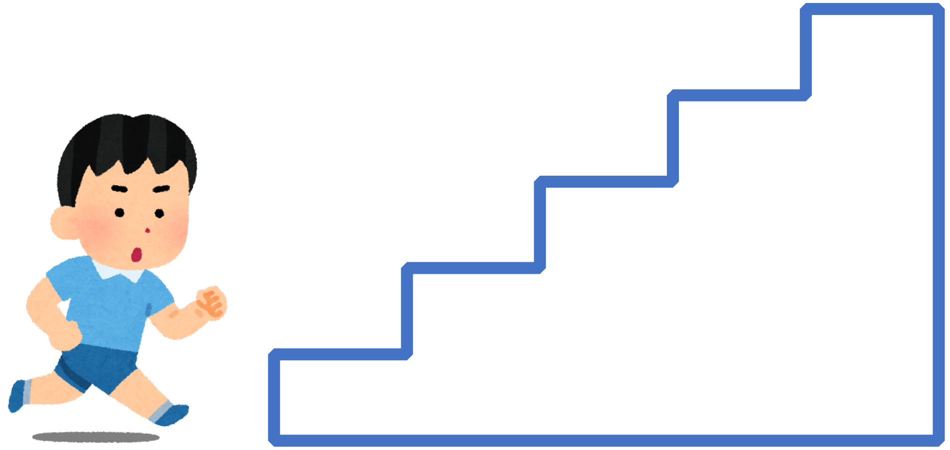
2. 部分問題の重複性 : 同じ部分問題が繰り返し出現する．異なる部分問題の数が少ない.

階段の上り方問題



- 5段の階段を上ることを考える
- 1段上る or 1段飛ばして2段上る
- 登り方は何通り？

階段の上り方問題

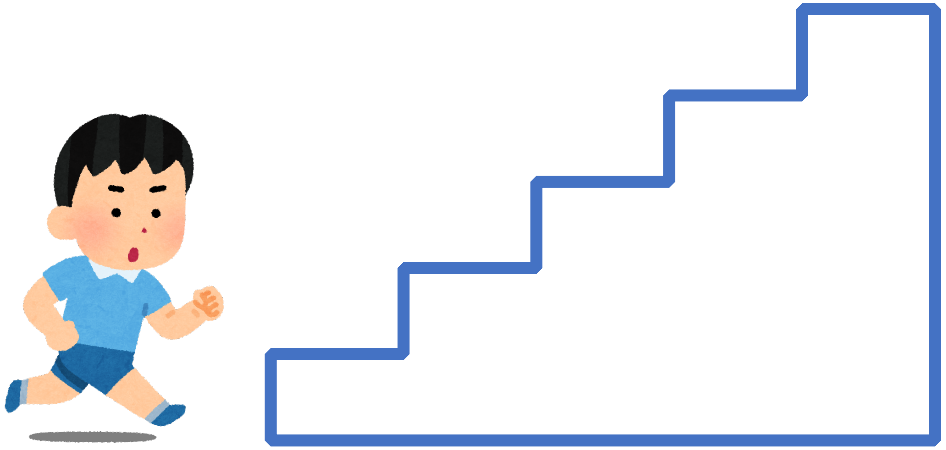


5段目の答えを直接求めようとすると……

[1, 1, 1, 1, 1],
[1, 1, 1, 2], [1, 1, 2, 1], [1, 2, 1, 1], [2, 1, 1, 1],
[1, 2, 2], [2, 1, 2], [2, 2, 1]

8通り！

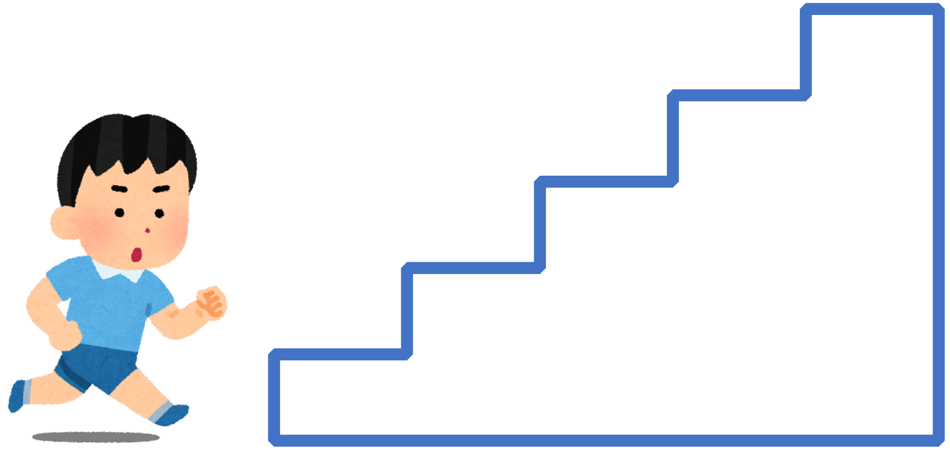
階段の上り方問題



N段目へ上る方法は,
1. (N - 1)段目から1段上ってくる
2. (N - 2)段目から2段上ってくる
の2パターンしかない

⇨足し合わせるだけで場合の数は計算できる！

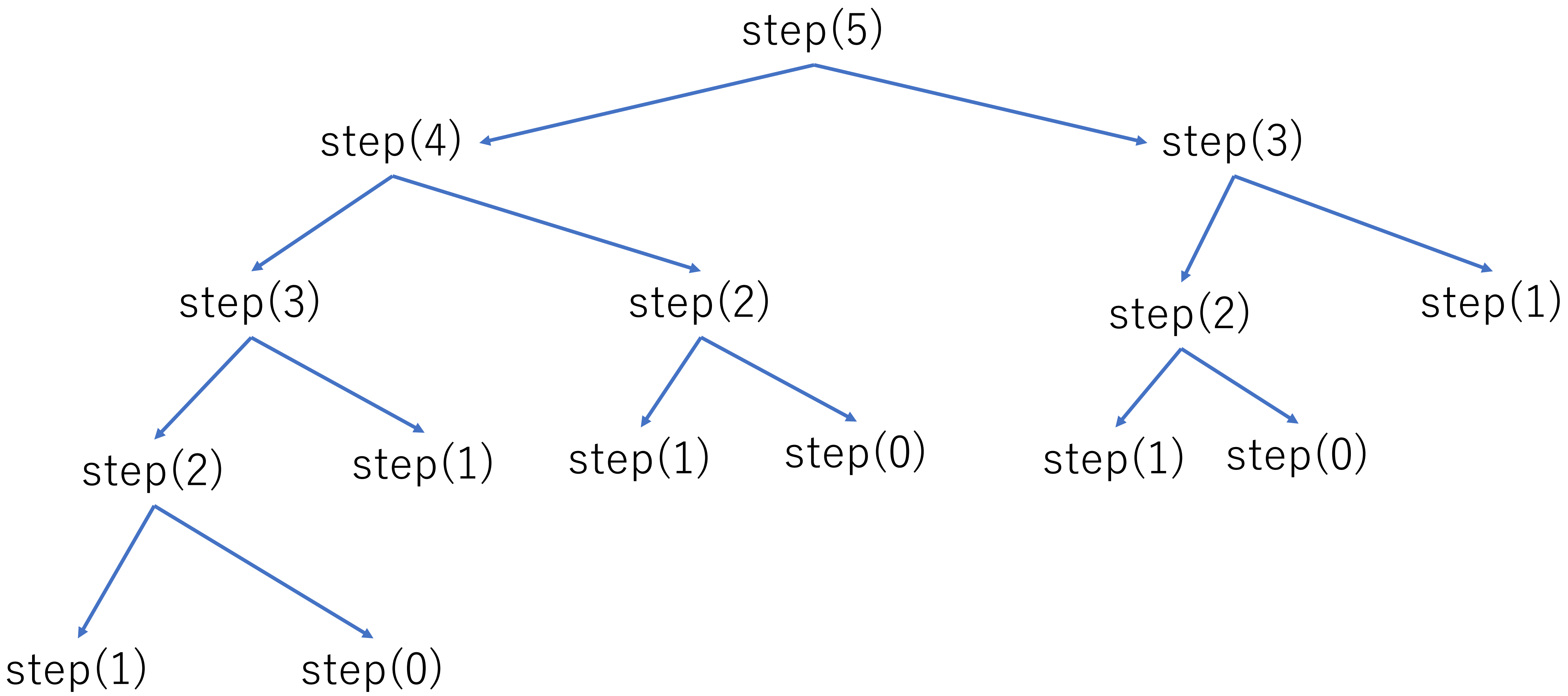
階段の上り方問題



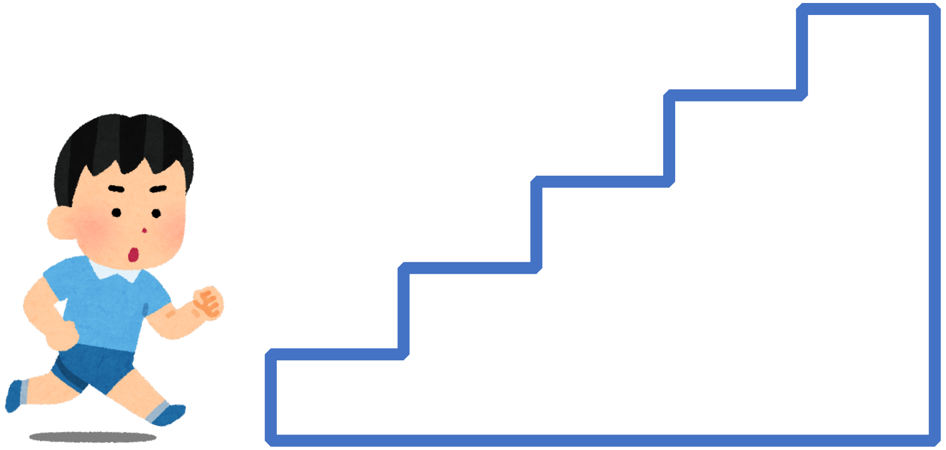
step(0) = 1
step(1) = 1
n > 1のとき,
step(n) = step(n – 1) + step(n - 2)

みたいな簡単な再帰関数を使って解いてみると……

階段の上り方問題

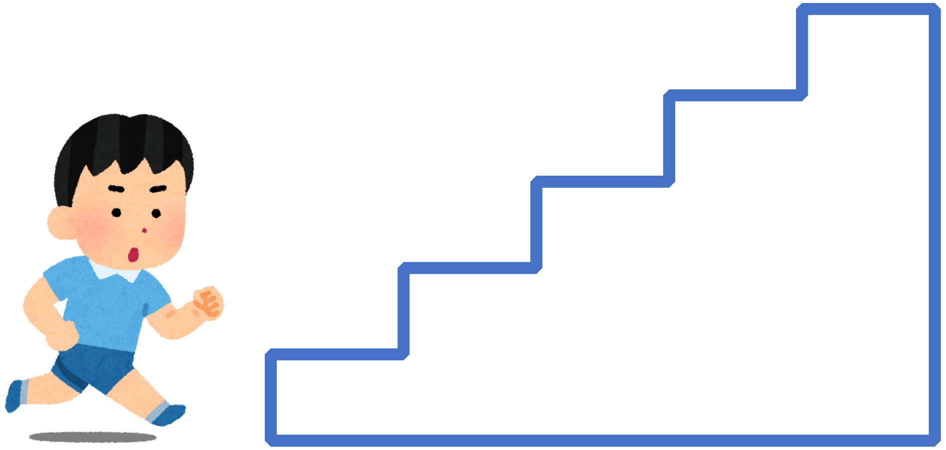

step(5)
step(4)
step(3)
step(3)
step(2)
step(2)
step(1)
step(2)
step(1)
step(1)
step(0)
step(1)
step(0)
step(1)
step(0)

階段の上り方問題



じゃあ100段目の答えは？
⇨ 厳しい！！

階段の上り方問題



各段の場合の数を記録するメモを用意する！
dp[i] : i段目への上り方の総数

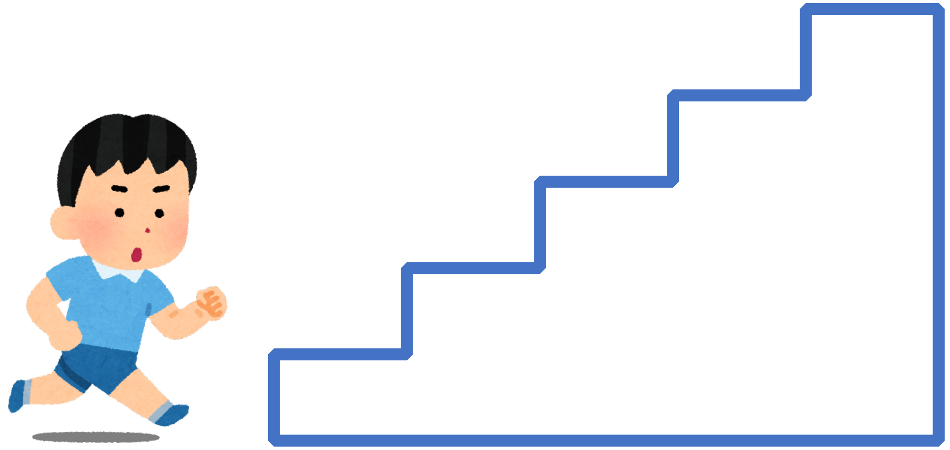
dp[i] = dp[i -1] + dp[i -2]

1段目は当然1通り
2段目は[1, 1], [2]の2通り

それ以降は計算で表を埋めていく

	1段	2段	3段	4段	5段
場合の数	1	2			

階段の上り方問題



3段目：1 + 2 = 3 通り
4段目：2 + 3 = 5 通り
5段目：3 + 5 = 8 通り

以前の段の場合の数はメモを参照しているだけ
⇨Nに関わらない，O(1)で取り出せる

これをN段まで繰り返せば良いので，
計算量はO(N)で済む！

	1段	2段	3段	4段	5段
場合の数	1	2	3	5	8

階段の上り方問題

DPが使用できる条件

1. 部分構造最適性 : 問題の最適解がその内部に, その部分問題に対する最適解を含む.

2. 部分問題の重複性 : 同じ部分問題が繰り返し出現する．異なる部分問題の数が少ない.

演習

- Typical Stairsを解いてみよう!

https://atcoder.jp/contests/abc129/tasks/abc129_c

- 余裕があればFrog 1を解いてみよう!

https://atcoder.jp/contests/dp/tasks/dp_a

Vacation

https://atcoder.jp/contests/dp/tasks/dp_c

明日から太郎君の夏休みが始まります．太郎君は夏休みの計画を立てることにしました．
夏休みは N日からなります．
各 i ($1 \le i \le N$) について，i 日目には太郎君は次の活動のうちひとつを選んで行います．

A : 海で泳ぐ．幸福度 a_i を得る．
B : 山で虫取りをする．幸福度 b_iを得る．
C : 家で宿題をする．幸福度 c_iを得る．

太郎君は飽き性なので，2日以上連続で同じ活動を行うことはできません．
太郎君が得る幸福度の総和の最大値を求めてください．

Vacation

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

表のような7日の例を考える.
幸福度が最大となるような組み合わせは？

⇨C, A, B, A, C, B, Aの時，46が最大.

どのようにして求めれば良い？

- 全探索だと$O(2^N)$
- i日目までの
  累計の幸福度の最大値 = dp[i]
  のDP?

Vacation

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

メモを二次元に！

j : i日目の行動

job[i][j] :
i日目，jの仕事をした時の幸福度

dp[i][j] :
i 日目に j の活動をする場合の，
i 日目までの累計の幸福度の最大値

Vacation

漸化式は？

$$dp[i][A] = job[i][A] + \max(dp[i-1][B], dp[i-1][C])$$

$$dp[i][B] = job[i][B] + \max(dp[i-1][A], dp[i-1][C])$$

$$dp[i][C] = job[i][C] + \max(dp[i-1][A], dp[i-1][B])$$

⇨ 部分構造最適性，部分問題の重複性を満たす！！

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目			
3日目			
4日目			
5日目			
6日目			
7日目			

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16		
3日目			
4日目			
5日目			
6日目			
7日目			

2日目にAを行うときの
幸福度の累計の最大値
C→Aの16

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	
3日目			
4日目			
5日目			
6日目			
7日目			

2日目にBを行うときの
幸福度の累計の最大値
C→Bの16

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目			
4日目			
5日目			
6日目			
7日目			

2日目にCを行うときの
幸福度の累計の最大値
B→Cの10

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目			
4日目			
5日目			
6日目			
7日目			

同様に埋めていく

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目	18	21	18
4日目			
5日目			
6日目			
7日目			

同様に埋めていく

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目	18	21	18
4日目	28	26	27
5日目			
6日目			
7日目			

同様に埋めていく

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目	18	21	18
4日目	28	26	27
5日目	31	34	36
6日目	38	39	38
7日目	46	43	40

Vacation

job

	A	B	C
1日目	6	7	8
2日目	8	8	3
3日目	2	5	2
4日目	7	8	6
5日目	4	6	8
6日目	2	3	4
7日目	7	5	1

dp[i][j]

	A	B	C
1日目	6	7	8
2日目	16	16	10
3日目	18	21	18
4日目	28	26	27
5日目	31	34	36
6日目	38	39	38
7日目	46	43	40

7日目の中での最大値,
46が答え

Vacation

演習

- Grid1を解いてみよう!

https://atcoder.jp/contests/dp/tasks/dp_h

- 余裕があればCoinsを解いてみよう!

https://atcoder.jp/contests/dp/tasks/dp_i

ナップサック問題

https://atcoder.jp/contests/dp/tasks/dp_d

N 個の品物があります。 品物には 1,2,……,N と番号が振られています。
各 i ($1 \leq i \leq N$) について、品物 i の重さは w_i で、価値は v_i です。

太郎君は、N 個の品物のうちいくつかを選び、
ナップサックに入れて持ち帰ることにしました。
ナップサックの容量は W であり、
持ち帰る品物の重さの総和は W 以下でなければなりません。

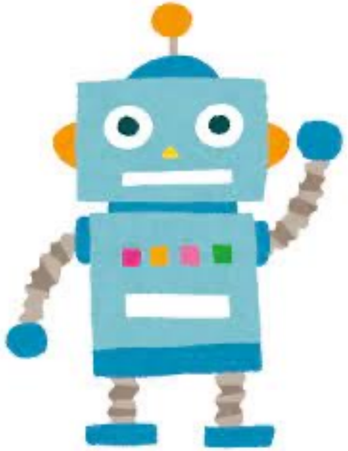
太郎君が持ち帰る品物の価値の総和の最大値を求めてください。

ナップサック問題

重さ：3
価値：6

重さ：4
価値：3

重さ：13
価値：20



重さ：10
価値：12

重さ：2
価値：3

ナップサックに重さ 15 まで入れられるとすると，
どの組み合わせで持って帰れば価値が最大になる？

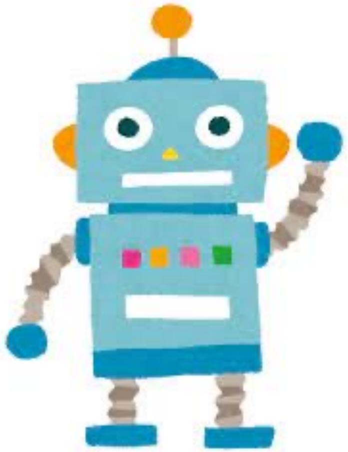
ナップサック問題

答え：

重さ：3
価値：6

重さ：4
価値：3

重さ：13
価値：20



重さ：10
価値：12

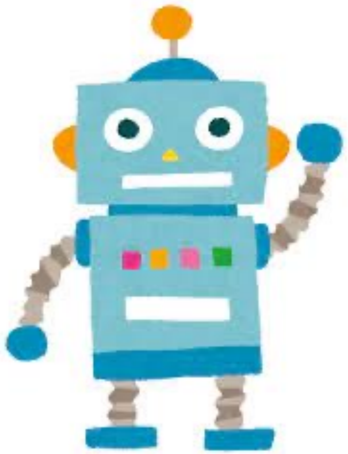
重さ：2
価値：3

ナップサック問題

重さ：3
価値：6
価／重：1位

重さ：4
価値：3
価／重：5位

重さ：13
価値：20
価／重：2位

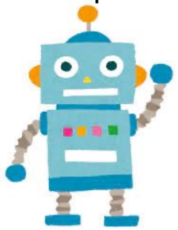


重さ：10
価値：12
価／重：4位

重さ：2
価値：3
価／重：3位

重さあたりの価値の高いものから選んでいっても(貪欲法)，
最善の結果が得られるとは限らない！

ナップサック問題

1	2	3	4	5
				
重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3

dp[i][W] :
i番目までの品物の中から,
W以下の重さになるように選んだ時の価値の最大値

今回求めたい答えは dp[5][15] となる

ナップサック問題

1

重さ：3
価値：6

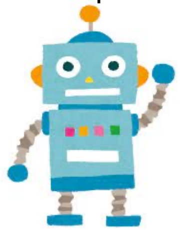
2

重さ：4
価値：3

3

重さ：13
価値：20

4



重さ：10
価値：12

5

重さ：2
価値：3

dp[i][W] について，
i番目の品物を選んだ場合の最大値と，選ばなかった場合の最大値に分け比較する．

- i番目の品物を選ぶ場合

i番目の品物を入れた後の余裕は $W - w_i$
i番目の品物を入れた後の余裕で得ることのできる価値の最大値は $dp[i - 1][W - w_i]$
価値の最大値は $v_i + dp[i - 1][W - w_i]$

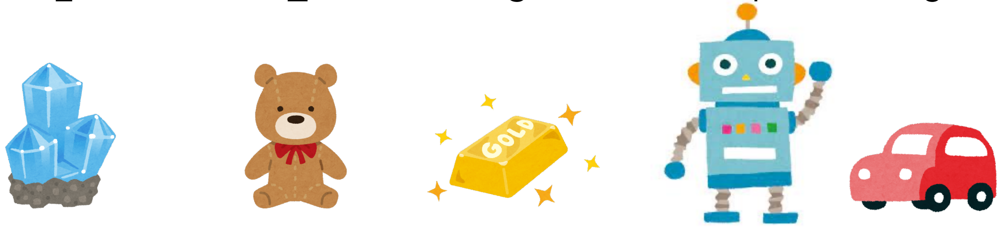
- i番目の品物を選ばない場合

価値の最大値は，$dp[i - 1][W]$

大きい方を採用！

ナップサック問題

1	2	3	4	5



1	2	3	4	5
重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3

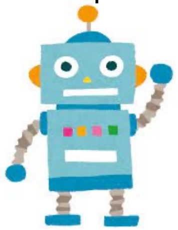
漸化式はこんな感じ．

$$dp[i][W] = \max(v_i + dp[i-1][W - w_i],\ dp[i-1][W])$$

$W > w_i$ のとき，配列外を参照してしまうので場合分けの必要はあるが……
d[i][W]は漸化式のような形で表すことが可能であることは確認できた．

⇨ 部分構造最適性，部分問題の重複性を満たす！！

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

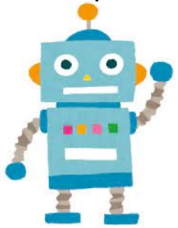
dp[i][W]

W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
4																
5																

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

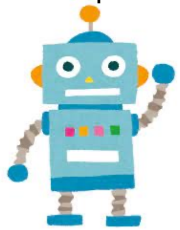
dp[i][W]

W：重さの上限

i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1																
	2																
	3																
	4																
	5																

i = 0 の時，使える品物はない！
何も選べないので，価値は常に 0

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

W：重さの上限

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2																
3																
4																
5																

i = 1 の時，使える品物は水晶のみ
W < 3 の時，選べないので 0
W ≧ 3 の時，水晶の価値 6

ナップサック問題

1	2	3	4	5

重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3

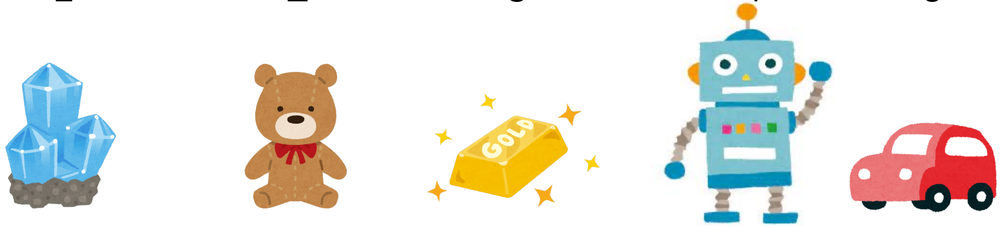
dp[i][W]

W：重さの上限

i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
i	**2**	0	0	0	6												
	3																
	4																
	5																

i = 2 の時，使えるようになった商品はクマ
とりあえず，W < 4(クマの重さ)の時，
dp[2][W] = dp[1][W]

ナップサック問題



1	2	3	4	5
重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3

ここで dp[2][W] について，（W ≧ 4）
クマを選んだ場合の最大値と，クマを選ばなかった場合の最大値に分け比較する．

- クマを選ぶ場合

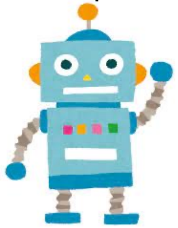
クマの重さ ＝ 4 より，クマを入れた後の余裕は W - 4
クマを入れた後の余裕で得ることのできる価値の最大値は dp[1][W - 4]
よって価値の最大値は 3(クマの価値) + dp[1][W – 4]

- クマを選ばない場合

価値の最大値は，dp[1][W]

大きい方を採用！

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

W：重さの上限

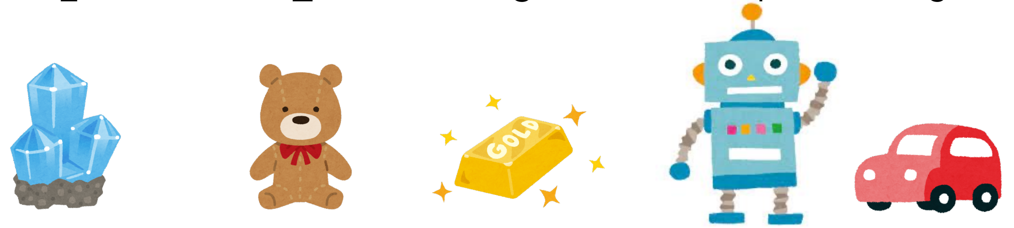
i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	3											
3																
4																
5																

dp[2][4]について
クマを選んだ場合，3 + dp[1][0] = 3

ナップサック問題

1	2	3	4	5



重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6											
3																
4																
5																

dp[2][4]について
選ばない場合，dp[1][4] = 6（こっち！）

ナップサック問題

	1	2	3	4	5
重さ	3	4	13	10	2
価値	6	3	20	12	3

dp[i][W]

W：重さの上限

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6										
3																
4																
5																

ナップサック問題

1 2 3 4 5

重さ：3 価値：6
重さ：4 価値：3
重さ：13 価値：20
重さ：10 価値：12
重さ：2 価値：3

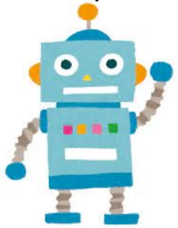
dp[i][W]

W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6	6									
3																
4																
5																

ナップサック問題

1	2	3	4	5
				
重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3

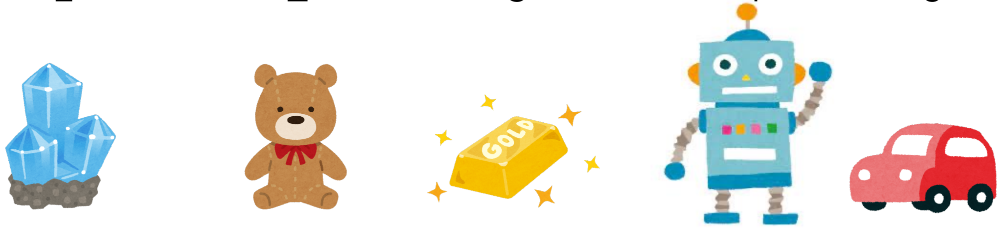
dp[i][W]

W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6	6	9								
3																
4																
5																

ナップサック問題



	1	2	3	4	5
	重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
	価値：6	価値：3	価値：20	価値：12	価値：3

dp[i][W]

W：重さの上限

i		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
	2	0	0	0	6	6	6	6	9	9							
	3																
	4																
	5																

ナップサック問題

1	2	3	4	5

重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

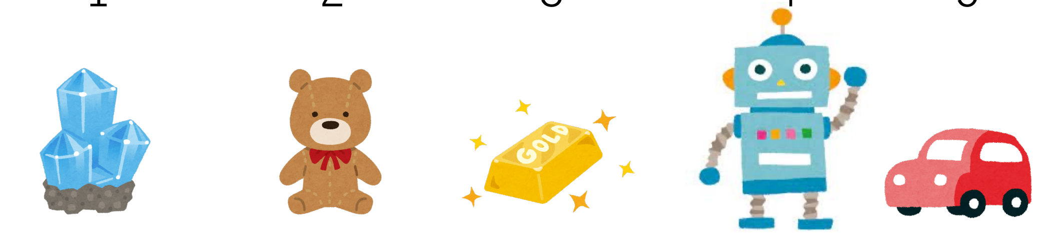
W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6	6	9	9	9						
3																
4																
5																

ナップサック問題

1 2 3 4 5



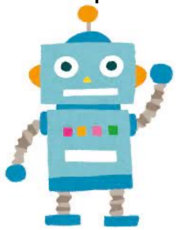
	1	2	3	4	5
重さ	3	4	13	10	2
価値	6	3	20	12	3

dp[i][W]

W：重さの上限

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6	6	9	9	9	9					
3																
4																
5																

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

W：重さの上限

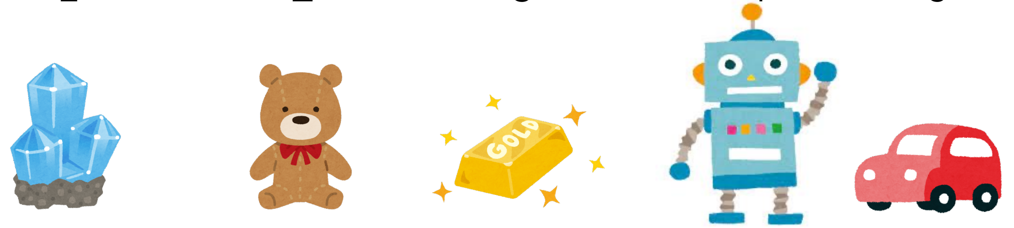
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
i	**2**	0	0	0	6	6	6	6	9	9	9	9	9	9	9	9	9
	3																
	4																
	5																

ナップサック問題

dp[i][W] について，

$W < w_i$ のとき，　dp[i - 1][W]

1	2	3	4	5
重さ：3	重さ：4	重さ：13	重さ：10	重さ：2
価値：6	価値：3	価値：20	価値：12	価値：3



$W \geqq w_i$ のとき，
i番目の品物を選んだ場合の最大値と，選ばなかった場合の最大値に分け比較する．

- i番目の品物を選ぶ場合

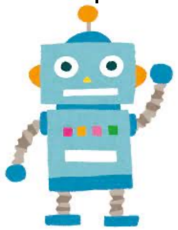
i番目の品物を入れた後の余裕は $W - w_i$
i番目の品物を入れた後の余裕で得ることのできる価値の最大値は dp[i - 1][$W - w_i$]
価値の最大値は v_i + dp[i - 1][$W - w_i$]

- i番目の品物を選ばない場合

価値の最大値は，dp[i - 1][W]

大きい方を採用！

ナップサック問題

1	2	3	4	5
				
重さ：3 価値：6	重さ：4 価値：3	重さ：13 価値：20	重さ：10 価値：12	重さ：2 価値：3

dp[i][W]

W：重さの上限

i

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6
2	0	0	0	6	6	6	6	9	9	9	9	9	9	9	9	9
3	0	0	0	6	6	6	6	9	9	9	9	9	9	20	20	20
4	0	0	0	6	6	6	6	9	9	9	12	12	12	20	20	20
5	0	0	3	6	6	9	9	9	9	12	12	12	15	20	20	23

ナップサック問題

演習

- Knapsack 1を解いてみよう！

https://atcoder.jp/contests/dp/tasks/dp_d

- 余裕があればKnapsack 2を解いてみよう！

https://atcoder.jp/contests/dp/tasks/dp_e

動的計画法(DP)とは？
Dynamic Programming

1. 分割統治法 : 部分問題に分割．その結果を利用して問題全体を解く

2. メモ化 : 繰り返し出現する部分問題の計算結果を再利用する

DPが使用できる条件

1. 部分構造最適性 : 問題の最適解がその内部に,
   その部分問題に対する最適解を含む.

2. 部分問題の重複性 : 同じ部分問題が繰り返し
   出現する．異なる部分問題の数が少ない.