個数制限ありナップサック問題を考える

1. 全探索を考える
2. メモ化を考える
3. 動的計画法（forループ版）を考える
4. 動的計画法バージョン２

「問題」

重さと価値がそれぞれのWi ViであるようなN個の品物がある。これらの品物から、重さの総和がWを超えないよう選んだときの価値の最大値

1 <= N <= 100

1 <= wi, vi <= 100

1 <= W <= 10000

入力の例

N = 4

(w, v) = ((2, 3), (1, 2), (3, 4), (2, 2))

W = 5

この場合の出力は７

-------------------------------------------------------------------------------------------------------

「全探索」

とりあえず総当たりのように全ての組み合わせを考える。

再帰で入れる場合入れない場合を順番に試して一番価値が大きい値が返ってくるようにする。この場合の計算量はとても多くなるのでメモ化を考える。

-------------------------------------------------------------------------------------------------------

「メモ化」

i=現在見ている品物 j=残り入る重さ

現在の再帰では全く同じ計算をしている箇所がある。それは、全く同じiとjが与えられた時その後の計算は全く一緒となる。※

→ということはiとjが与えられた時の計算をメモしていくとその後の計算がカットされる。

※これに関しては関数がどういう順番で呼ばれるか二分木を描いてみるといい。

-------------------------------------------------------------------------------------------------------

「動的計画法版（forループ）」

再帰はややこしいので単純なforループで実現できないかを考える。

→Memoの値の決まり方を観察してみる。

↓へ

memo

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i/j | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 2 | 3 | 5 | 6 | 7 |
| 1 | 0 | 2 | 2 | 4 | 6 | 6 |
| 2 | 0 | 0 | 2 | 4 | 4 | 6 |
| 3 | 0 | 0 | 2 | 2 | 2 | 2 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |

これはmemoの二次元配列の最終的な中身。

Memo[i][j]の値はi番目以降の品物を使用して重さjまでの制限でどれだけ入るかである。

[0][5]が初期値で決まり下の方から順番に決まっていく。

→ならばiは4からjは0からの2重ループで出来ないだろうか？

/\*

dp[i + 1][j - w[i]] + v[i] これが何を表すか？

j - w[i]で今見ている品物分の重さを引いたjに飛ぶ

i + 1で今見ている商品が入ってない場所を表す

→今見ている商品が入ってないかつ今見ている商品が入るだけの空きがある場所

→それはそこまでの最大値がすでに入っているはず

→表で言うなら左下

dp[i][j] = dp[i + 1][j]

こっちの式は今のjではもう入らない状態の価値を取ってくる

これらを比べる

\*/

このプログラムで実現していることはバッグの小さい状態かつ選べる品物が小さい状態から最大最大になる状態を順に選んでいき条件に合うものを見つける。

-------------------------------------------------------------------------------------------------------

「動的計画法バージョン２」

iがマイナスループしているのをどうにかならないか？

Memo[i][j]の値はi番目以降の品物を使用して重さjまでの制限でどれだけ入るかである。

↓

Memo[i][j]の値はi番目までの品物を使用して重さjまでの制限でどれだけ入るかである。

こうすると左上から順に下方向へ出来る様になる。