基礎的な動的計画法トピック

ICPC 国内予選突破に向けて

tsutaj (@_TTJR_)

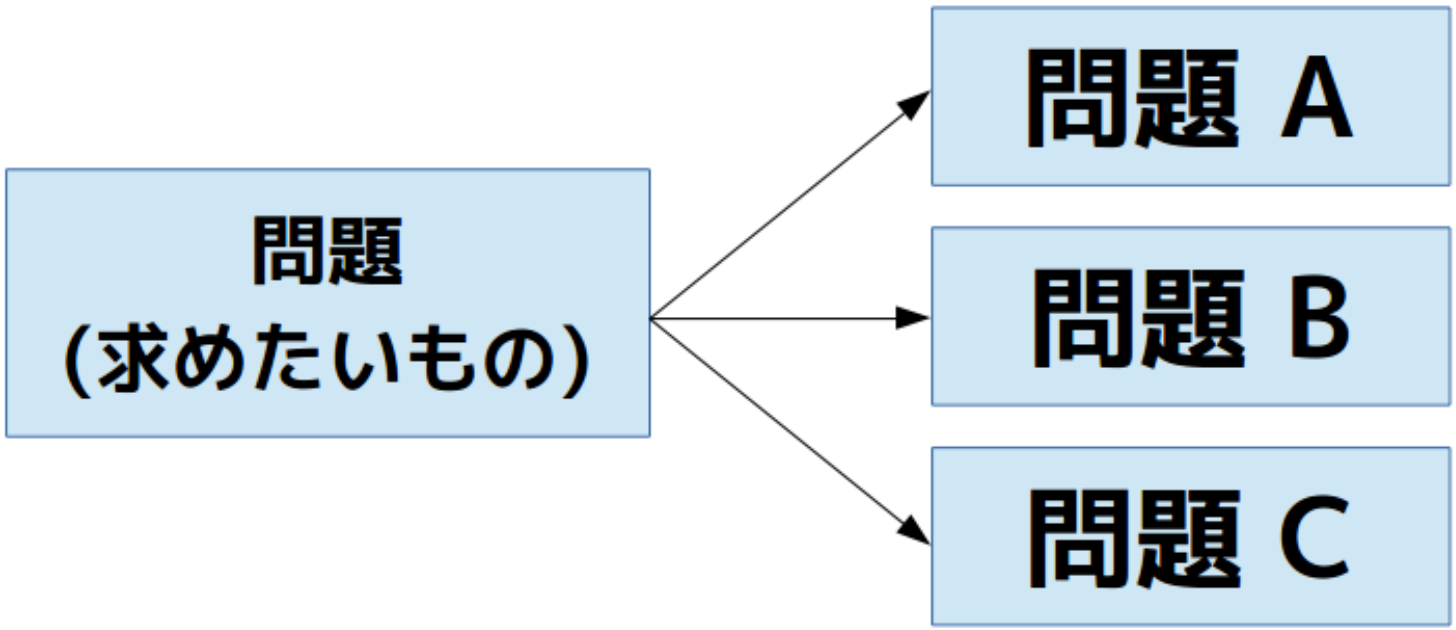
Hokkaido University M1

2018 年 6 月 7 日

動的計画法とは

動的計画法 (Dynamic Programming)

- 問題を複数の部分問題に分けて，それを統合して解く手法
- 同じとみなせる状態を見極めて，まとめる
- 汎用性がめちゃめちゃある分野

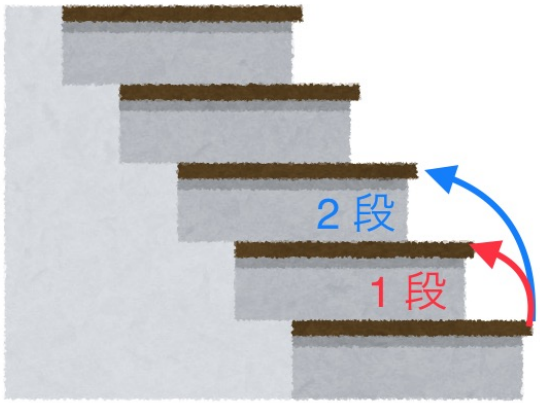


状態をまとめる、とは？

- 状態をまとめるとはどういうことなのか？
- 例として以下の問題を考える

階段のぼり

- N 段からなる階段がある
- 1 段のぼるか、2 段一気にのぼるかできる
- ちょうど N 段のぼる方法は何通りあるか？
  - 例えば 4 段のぼる方法は、
    $\{1,1,1,1\}, \{1,1,2\}, \{1,2,1\}, \{2,1,1\}, \{2,2\}$ の 5 通り



状態をまとめる、とは？

- i 段までのぼった時の通り数を知るためには、どんな情報がわかっていればよいか？

状態をまとめる、とは？

- i 段までのぼった時の通り数を知るためには、どんな情報がわかっていればよいか？
  - 一度にのぼれるのは 1 段または 2 段なのだから・・・
  - $i-1$ 段目までのぼった通り数と、$i-2$ 段目までのぼった通り数がわかっていれば、求められそう！
- 通り数の情報だけあればよく、具体的にどんなのぼりかたをしてきたかは覚える必要がない！

階段のぼりの解法

$a_i := i$ 段目までのぼる場合の数 としたとき、

$$a_i = \begin{cases} 0 & i < 0 \\ 1 & i = 0 \\ a_{i-1} + a_{i-2} & \text{それ以外} \end{cases} \tag{1}$$

答えとなるのは a_N の値

状態をまとめる、とは？

- このように、既に計算済みの情報を組み合わせて「新たな情報」を得ていく方法が、動的計画法！
  - 前の状態を組み合わせるため、漸化式の形になる
  - (ちなみに先ほどの問題で登場した数列はフィボナッチ数列と同じです)
- 数え上げであろうが最大・最小問題であろうが yes/no 問題であろうが、考え方は全く同じ！
  - さっきの問題の拡張を考えてみよう (算数やれば $O(1)$ なのは勘弁)
  - ちょうど N 段のぼるようなのぼりかたのうち、ステップ数の最小値を DP で解くには？
  - ちょうど K ステップでちょうど N 段のぼれるかどうかを DP で解くには？

部分和問題

部分和問題

- 自然数集合 $S = \{a_1, a_2, \cdots, a_N\}$ がある
- S 内の数字はそれぞれ 1 度だけ使える
- S 内の数字を組み合わせて W を作ることは可能か？
  - 例: $S = \{1, 2, 8\}$ から $W = 10$ は作れるが、$W = 7$ は作れない

- 漸化式を作ってみよう！　(ヒント: 二次元の漸化式)

部分和問題

部分和問題

- 自然数集合 $S = \{a_1, a_2, \cdots, a_N\}$ がある
- S 内の数字はそれぞれ 1 度だけ使える
- S 内の数字を組み合わせて W を作ることは可能か？
  - 例: $S = \{1, 2, 8\}$ から $W = 10$ は作れるが、$W = 7$ は作れない

- 漸化式を作ってみよう！　(ヒント: 二次元の漸化式)
- $\mathrm{dp}[i$ 番目の要素まで使って$][$合計が j である状態$]$:= True / False
- 遷移はどのようになる？

部分和問題

部分和問題

- 自然数集合 $S = \{a_1, a_2, \cdots, a_N\}$ がある
- S 内の数字はそれぞれ 1 度だけ使える
- S 内の数字を組み合わせて W を作ることは可能か？
  - 例: $S = \{1, 2, 8\}$ から $W = 10$ は作れるが、$W = 7$ は作れない

- 漸化式を作ってみよう！　(ヒント: 二次元の漸化式)
- $\mathrm{dp}[i$ 番目の要素まで使って$][$合計が j である状態$]$:= True / False
- 遷移はどのようになる？
- i 番目の要素を「足し合わせる」場合と「足しあわせない」場合がある
  - i 番目の要素の値を a_i とおく
  - 足しあわせると、合計は a_i 増える → 合計が $j - a_i$ である状態を参照！
  - 足しあわせないと、合計は増えない → 合計が j である状態を参照！
- どう実装すればいいの？

部分和問題

このような実装になる

```cpp
#include <cstdio>
using namespace std;

int value[110];
bool dp[110][1010];

int main() {
    int N, W; scanf("%d%d", &N, &W);
    for(int i=1; i<=N; i++) {
        scanf("%d", &value[i]);
    }

    // 何も足しあわせないと合計は 0
    dp[0][0] = true;

    for(int i=1; i<=N; i++) {
        for(int j=0; j<=W; j++) {
            // i 番目までの要素を組み合わせて合計を j にできるか？

            // 足しあわせない (前で合計が j の状態を参照)
            dp[i][j] = dp[i][j] or dp[i-1][j];

            // 足し合わせる (前で合計が j - value[i] の状態を参照)
            // 合計が負の状態はありえない！
            if(j - value[i] >= 0) {
                dp[i][j] = dp[i][j] or dp[i-1][j - value[i]];
            }
        }
    }
    printf("%s\n", dp[N][W] == true ? "Yes" : "No");
    return 0;
}
```

部分和問題

階段のぼりのときと同様に、yes / no 問題以外にも様々な形に発展できる！

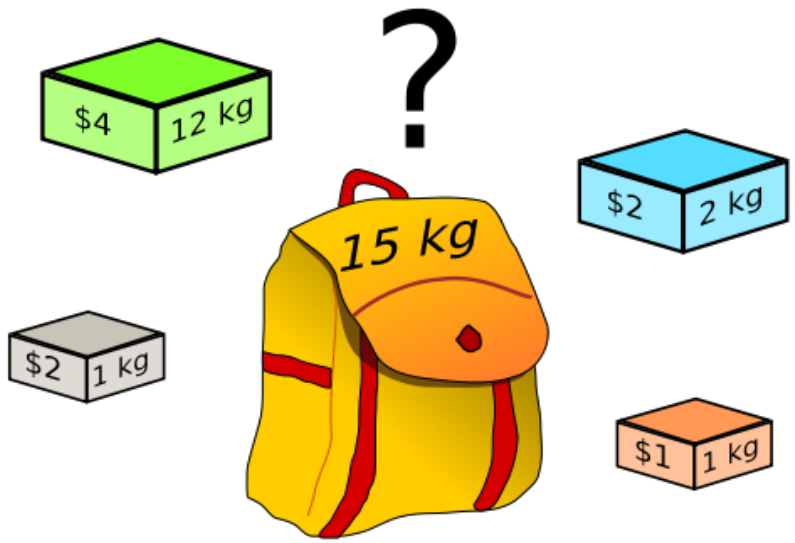
- 合計がちょうど W になるような要素の選び方は何通りある？
- 合計がちょうど W になるような要素の選び方の中で、選ぶ要素数の最小 (最大) はいくらか？
- 集合 S の要素を組み合わせてできる整数は全部で何通りある？ (何も選ばない場合の 0 もカウント)

先ほどの漸化式を少し変えると対応できるので、やってみよう！

ナップザック問題

01 ナップザック問題

- 容量 C のナップザックと、N 個の品物がある
- i 番目の品物は容量 c_i で、その価値は v_i
- 選んだ品物の容量の合計が C を超えない範囲で、品物を自由に選択できる
- 選んだ品物の価値の合計の最大値はいくらか？
  - 下の画像だと価値の合計 8 ドルが最大 (容量の合計 15 kg)



ナップザック問題

- **漸化式を作ってみよう！　(ヒント: 二次元の漸化式)**

ナップザック問題

- 漸化式を作ってみよう！　(ヒント: 二次元の漸化式)
- $\mathrm{dp}[i$ 番目の品物まで使って$][$容量合計が $j] :=$ 価値合計の最大値
- 遷移はどのようになる？

ナップザック問題

- 漸化式を作ってみよう！　(ヒント: 二次元の漸化式)
- $\mathrm{dp}[i$ 番目の品物まで使って$][$容量合計が $j] :=$ 価値合計の最大値
- 遷移はどのようになる？
- i 番目の品物を「足し合わせる」場合と「足しあわせない」場合がある
  - 足しあわせると、容量合計は c_i 増え、価値合計は v_i 増える
    - 容量合計が $j - c_i$ である状態を参照
    - その状態における価値合計の最大に v_i を足したものを使おう！
  - 足しあわせないと、容量合計も価値合計も増えない
    - j である状態を参照！
- つまり、さっきの部分和問題と考え方はほぼ同じ！

ナップザック問題

このような実装になる

```
6  int cap[110], value[110];
7  int dp[110][10010];
8
9  int main() {
10     int N, C; scanf("%d%d", &N, &C);
11     for(int i=1; i<=N; i++) {
12         scanf("%d%d", &value[i], &cap[i]);
13     }
14
15     // dp 配列を -1 埋めする (-1 はありえない状態を表す)
16     memset(dp, -1, sizeof(dp));
17
18     // 何も足しあわせないと価値合計は 0
19     dp[0][0] = 0;
20
21     for(int i=1; i<=N; i++) {
22         for(int j=0; j<=C; j++) {
23             // i 番目までの要素を組み合わせて容量合計が j である状態
24
25             // 足しあわせない (前で合計が j の状態を参照)
26             dp[i][j] = max(dp[i][j], dp[i-1][j]);
27
28             // 足し合わせる (前で合計が j - value[i] の状態を参照)
29             // 合計が負の状態はありえない！
30             if(j - cap[i] >= 0 && dp[i-1][j - cap[i]] != -1) {
31                 dp[i][j] = max(dp[i][j], dp[i-1][j - cap[i]] + value[i]);
32             }
33         }
34     }
35
36     // 各容量合計における最大値をすべてみて、その最大値を答えとする
37     int ans = 0;
38     for(int i=0; i<=C; i++) {
39         ans = max(ans, dp[N][i]);
40     }
41     printf("%d\n", ans);
42     return 0;
43 }
```

その他の DP

- 今回は時間の都合上入門編しか扱えませんが、動的計画法にはさまざまなジャンルがあります
  - bit DP (2017 年 D 問題)
  - 区間 DP (2016 年 D 問題)
  - 桁 DP
  - 木 DP
  - 挿入 DP
  - などなど・・・
- 様々な種類の問題を解いてなれておくと、対応しやすくなると思います

実践問題

- **実際に問題を解いてみよう！**
- DP **の問題を集めたバーチャルコンテストを用意していますので、取り組んでみてください。**
  - Div1 ▸ Link
  - Div2 ▸ Link
- **コンテスト時間は** 19:25 **〜** 20:40 **です。**
- **終了後に** Div2 **の各問題について簡単に解説を行います。**