

競プロ勉強会04

elzup

DP (Dynamic Programming)

動的計画法

DP 動的計画法

- ・ 分割統治
 - ・ 問題を分割しその結果から全体を解決する
- ・ メモ化
 - ・ 計算結果を保存しておき使い回す

DP: フィボナッチ

- ・ フィボナッチ数列の n 番目を求める
- ・ 1 1 2 3 5 8 13

問題の分割とは

- ・ フィボナッチ数列の定義

- ・ $f(0) = 0$

- $f(1) = 1$

- $f(n) = f(n - 1) + f(n - 2) \quad (n \geq 2)$

問題の分割とは

- ・ n 番目のフィボナッチを求める
- ・ $\rightarrow (n - 1)$ 番目のフィボナッチを求める +
- ・ $(n - 2)$ 番目のフィボナッチを求める
- ・ 漸化式は典型的な例
- ・ \rightarrow 漸化式に落とし込んでから解く

実装してみよう

- Aoj - Fibonacci Number
- http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_A

fibonacci 関数(再帰)

- ```
int fibonacci(int n) {
 if (n == 0) { return 0; }
 if (n == 1) { return 1; }
 return fibonacci(n - 1) + fibonacci(n - 2);
}
```



# メモ化する (メモ化再帰)

```
• int fibonacci(int n) {
 if (memo[n] != -1) {
 return memo[n];
 }
 if (n == 0) { return 1; }
 if (n == 1) { return 1; }
 memo[n] = fibonacci(n - 1) + fibonacci(n - 2);
 return memo[n];
}
```

# ボトムアップとトップダウン

- ・ トップダウン: 必要となった部分問題を解く  
再帰
- ・ ボトムアップ: 先に部分問題を解いておく  
配列とループ
- ・ さっきfibonacciの書き方はトップダウン

# フィボナッチを ボトムアップで書いてみる

```
int main() {
 int n;
 cin >> n;
 fill(dp, dp + 100, -1);
 dp[0] = dp[1] = 1;
 for (int i = 2; i <= n; ++i) {
 dp[i] = dp[i - 1] + dp[i - 2];
 }
 cout << dp[n] << endl;
}
```

# フィボナッチを ボトムアップで書いてみる

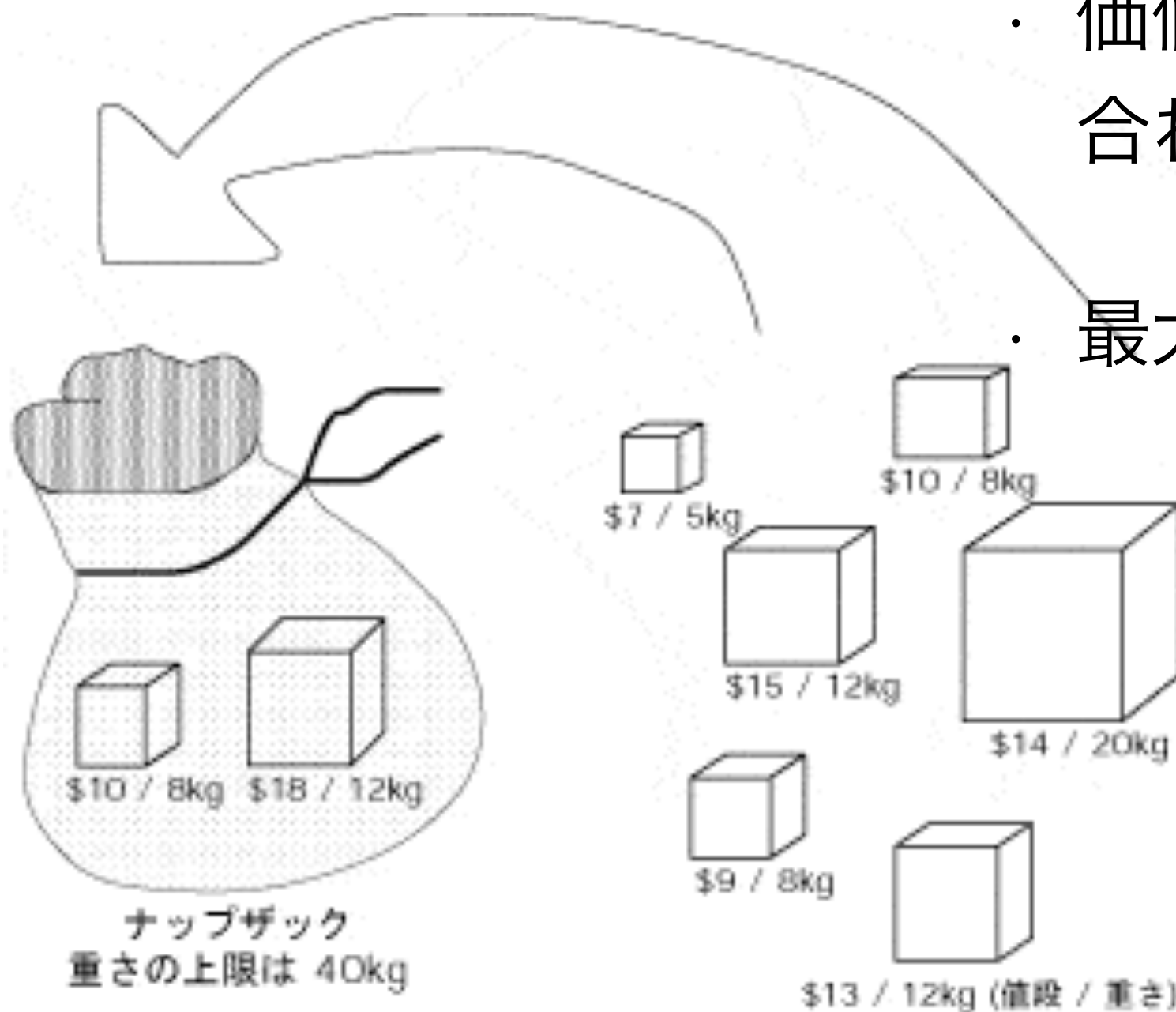
- ・ 漸化式を順番に計算していく感じ
- ・ => 再帰がなくなる!

# DP: もう一問

- AOJ - 0-1 Knapsack Problem
- [http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL\\_1\\_B&lang=jp](http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_B&lang=jp)

# ナップサック問題

- ・ 価値が最大になる組み合わせはどれ？
- ・ 最大となる時の価値は？



# Input

• 4 5

4 2

5 2

2 1

8 3

• N W

$v_1$   $w_1$

$v_2$   $w_2$

...

$v(N)$   $w(N)$

# DP: 考え方

- ・ 全てのパターン(それぞれの有無)を試す？
- ・  $\rightarrow$   $n$ 個なら  $O(2^n)$
- ・  $\rightarrow (1 \leq n \leq 100)$  なので 不可能



# DP: 考え方(ト ッ プダウ ン)

- ・ 1 つずつ順番に入れるか入れないかを考えていったとする
- ・ k番目の品物を **入れた場合** と **入れなかった場合** で価値が高い方
- ・ [k 番目までと容量  $w$  で最大価値]  $\rightarrow$
- ・  $\text{MAX}(\begin{array}{l} (k - 1)\text{番目までと容量 } w \text{ で最大価値,} \\ (k - 1)\text{番目までと容量 } (w - \text{k番目の重さ}) \text{ で最大価値} + \text{k番目の価値} \end{array})$

# 再帰実装

```
• int dp(int k, int w) {
 if (w < 0) { return NEGA_INF; }
 if (k == -1 || w < 0) { return 0 ; }
 return max(
 dp(k - 1, w - weight[k]) + value[k],
 dp(k - 1, w));
}
```

# メモ化

- 関数について (k, w) の組み合わせが同じ時は前回の結果を返す -  
> memo[k][w]
- ```
int dp(int k, int w) {  
    if (w < 0) { return NEGA_INF; }  
    if (k == -1 || w < 0) { return 0; }  
    int &m = memo[k][w];  
    if (m != -1) { return m; }  
    return m = max(  
        dp(k - 1, w - weight[k]) + value[k],  
        dp(k - 1, w));  
}
```

DP: 考え方(ボトムアップ)

- ・ http://judge.u-aizu.ac.jp/onlinejudge/commentary.jsp?id=DPL_1_B
- ・ ループする時の順番を意識 k, w はそれぞれ昇順?降順?
- ・ k 番目の品物を入れた場合と入れなかった場合で価値が高い方

実装

```
• for (int i = 1; i <= n; i++) {  
    for (int j = 0; j <= w; j++) {  
        dp[i][j] = max(  
            dp[i - 1][j],  
            dp[i - 1][j - weight[i - 1]] + value[i - 1];  
    }  
}
```

実装

- $dp[i][j] = dp[i - 1][j];$
if ($j - \text{weight}[i - 1] \geq 0$) {
 $dp[i][j] = \max(dp[i][j], dp[i - 1][j - \text{weight}[i - 1]] + \text{value}[i - 1]);$
}

参考資料

- AOJ - DP-Introduction
http://judge.u-aizu.ac.jp/onlinejudge/topic.jsp?cid=ALDS1#problems/ALDS1_10
http://judge.u-aizu.ac.jp/onlinejudge/topic.jsp?cid=DPL#problems/DPL_1
- AOJのDP問題集 <http://d.hatena.ne.jp/kyuridenamida/20111009/1318091499>
- DP - フィボナッチで理解 <http://qnighy.hatenablog.com/entry/20091220/1261284700>