アルゴリズム講座

第5回 〜動的計画法〜

2020/08/22 11:00~

@utsubo_21 高澤 栄一

今回の内容

- 下記のアルゴリズム・データ構造を10回程度に分けて１つずつ解説

Lesson1 ‣ 全探索
Lesson2 ‣ 二分探索
Lesson3 ‣ 深さ優先探索
Lesson4 ‣ 幅優先探索
Lesson5 ‣ 動的計画法
‣ 累積和
‣ グラフ・木
‣ ダイクストラ法
‣ ワーシャルフロイド法
‣ クラスカル法
‣ Union-Find

参考：レッドコーダーが教える、競プロ・AtCoder上達のガイドライン【中級編：目指せ水色コーダー！】
https://qiita.com/e869120/items/eb50fdaece12be418faa

Lesson 5

動的計画法とは？

動的計画法とは？

- 対象となる問題を複数の部分問題に分割し、部分問題の計算結果を記録しながら解いていく手法を総称してこう呼ぶ
- 英語だとDynamic Programming,

  競プロ界隈では略して「DP」と呼ばれている

今回のゴール

DPのイメージを掴む

Question

問題

- N日間の夏休みの計画を立てる (N ≦ 10^5)
- i 日目には，次の活動のうちどれか1つをする
  - A. 海で泳ぐ　　幸福度 a[i] を得る
  - B. 山で虫取り　　幸福度 b[i] を得る
  - C. 家で宿題　　幸福度 c[i] を得る
- 飽き性なので同じ活動を2日続けて出来ない
- 得られる幸福度の総和の最大値を求めよ

問題　入出力例

	1日目	2日目	3日目
 a[i]:	10	20	30
 b[i]:	40	50	60
 c[i]:	70	80	90

1日目に海，2日目に虫取り，3日目に宿題

➠ **幸福度の総和 150** (=10 + 50 + 90)

問題　入出力例

	1日目	2日目	3日目
 a[i]:	10	20	30
 b[i]:	40	50	60
 c[i]:	70	80	90

1日目に宿題，2日目に宿題，3日目に虫取り

➠ 2日連続で同じ活動は出来ない

問題　入出力例

	1日目	2日目	3日目
 a[i]:	10	20	30
 b[i]:	40	50	60
 c[i]:	70	80	90

1日目に宿題，2日目に虫取り，3日目に宿題

➠ **幸福度の総和 210** (=70 + 50 + 90)

まずは全探索

- 全ての組み合わせの幸福度の総和を列挙
  - 再帰等で実装できる

まずは全探索

- 全探索の実装例

```
// 引数は順に，何日目，前日の活動，合計幸福度
void DFS(day, prev_action, sum_happy) {
  // 幸福度の総和が今の最大値を超えていたら更新
  if (day == N) {
    max_happy = max(max_happy, sum_happy);
    return;
  }
  if (prev_action != ‘A’) // 前日と同じ活動は出来ない
    DFS(day + 1, ‘A’, sum_happy + a[day]); // 海で泳ぐ
  if (prev_action != ‘B’)
    DFS(day + 1, ‘B’, sum_happy + b[day]); // 山で虫取り
  if (prev_action != ‘C’)
    DFS(day + 1, ‘C’, sum_happy + c[day]); // 宿題
}
```

* A, B, C をそれぞれ 海で泳ぐ，虫取り，宿題とする

まずは全探索

- 全探索の実装例

```
// 引数は順に，何日目，前日の活動，合計幸福度
void DFS(day, prev_action, sum_happy) {
  // 幸福度の総和が今の最大値を超えていたら更新
  if (day == N) {
    max_happy = max(max_happy, sum_happy);
    return;
  }
  if (prev_action != ‘A’) // 前日と同じ活動は出来ない
    DFS(day + 1, ‘A’, sum_happy + a[day]); // 海で泳ぐ
  if (prev_action != ‘B’)
    DFS(day + 1, ‘B’,
  if (prev_action != ‘
    DFS(day + 1, ‘C’, sum_happy + c[day]); // 宿題
}
```

day日目にAを選択

sum_happy に a[day] を足して翌日へ

まずは全探索

- しかし，活動の組み合わせ数は O(2^N)
  - 1日目は3通り
  - 2日目以降は2通り (同じ活動を連続で出来ないので)
  - よって，全組み合わせは 3 * 2^(N-1) 通り

- N ≦ 10^5 なのでこれでは間に合わない🙅‍♀️

動的計画法的に解く

- 動的計画法では
  - 対象となる問題を複数の部分問題に分割し、
    部分問題の計算結果を記録しながら解いていく
- 今回の問題における方針
  - N日間のうち1日目までの幸福度を最大化する
  - 1日目までの結果を使い，2日目の幸福度を最大化する
  - 2日目までの結果を使い，3日目の幸福度を最大化する

動的計画法的に解く

- 2日目にAを選んだ時の幸福度の最大値を考える
- 答えは次の2つの中で大きい方となる
  - 1日目にBを選ぶ幸福度(b[1]) + 2日目にAを選ぶ幸福度(a[2])
  - 1日目にCを選ぶ幸福度(c[1]) + 2日目にAを選ぶ幸福度(a[2])

*** A, B, C をそれぞれ 海で泳ぐ，虫取り，宿題とする**

動的計画法的に解く

- 2日目にAを選んだ時の幸福度の最大値を考える
- 答えは次の2つの中で大きい方となる
  - 1日目にBを選ぶ幸福度(b[1]) + 2日目にAを選ぶ幸福度(a[2])
  - 1日目にCを選ぶ幸福度(c[1]) + 2日目にAを選ぶ幸福度(a[2])

B, Cを選択した場合も同様に考えれば,

2日目時点での幸福度の最大値が出せる

*** A, B, C をそれぞれ 海で泳ぐ，虫取り，宿題とする**

動的計画法的に解く

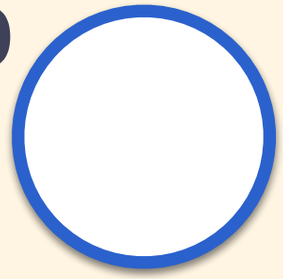
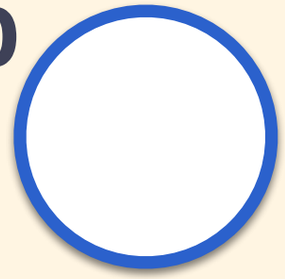
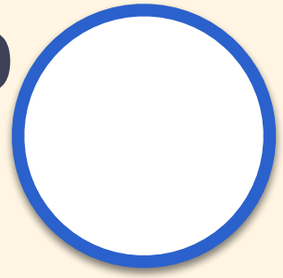
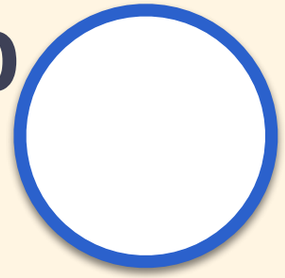
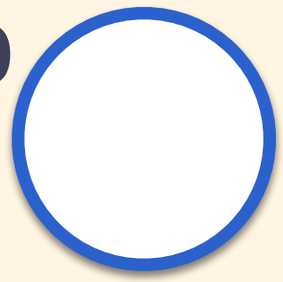
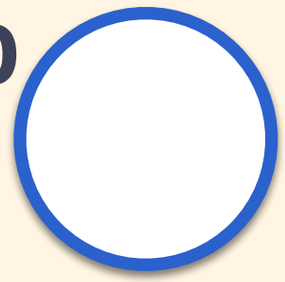
- では，3日目にAを選ぶ時の幸福度の最大値は？
- 答えは次の2つの中で大きい方となる
  - 2日目にBを選ぶ幸福度の最大値 + 3日目にAを選ぶ幸福度
  - 2日目にCを選ぶ幸福度の最大値 + 3日目にAを選ぶ幸福度

動的計画法的に解く

- では，3日目にAを選ぶ時の幸福度の最大値は？

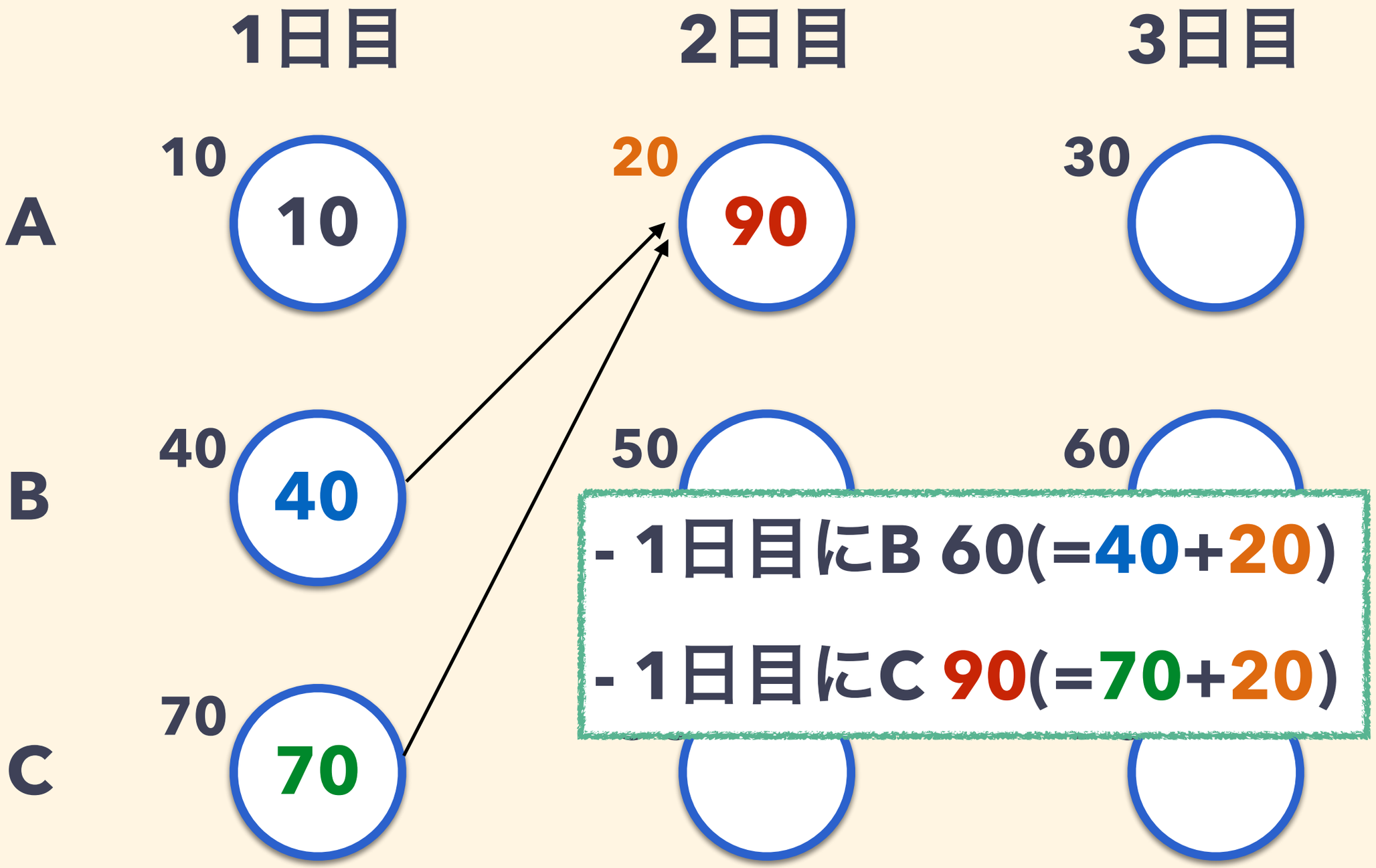
- 「2日目にBを選んだ場合の最大値」と
  「2日目にCを選んだ場合の最大値」のみ
  分かっていればこれは解ける

図で見ると

	1日目	2日目	3日目
A	10 	20 	30 
B	40 	50 	60 
C	70 	80 	90 

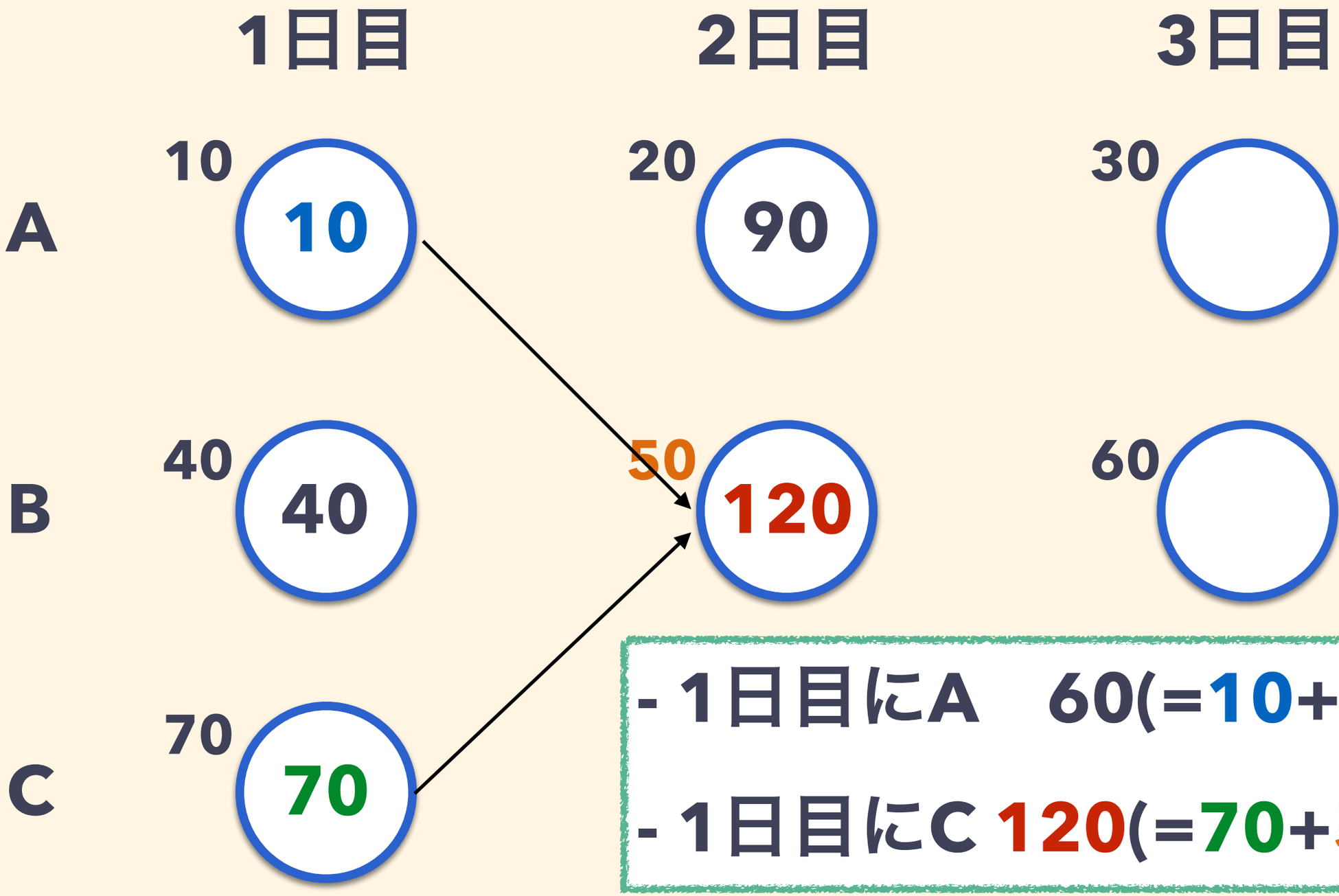
1日目の各活動を選んだ時の最大値

図で見ると



2日目にAを選んだ時の最大値

図で見ると

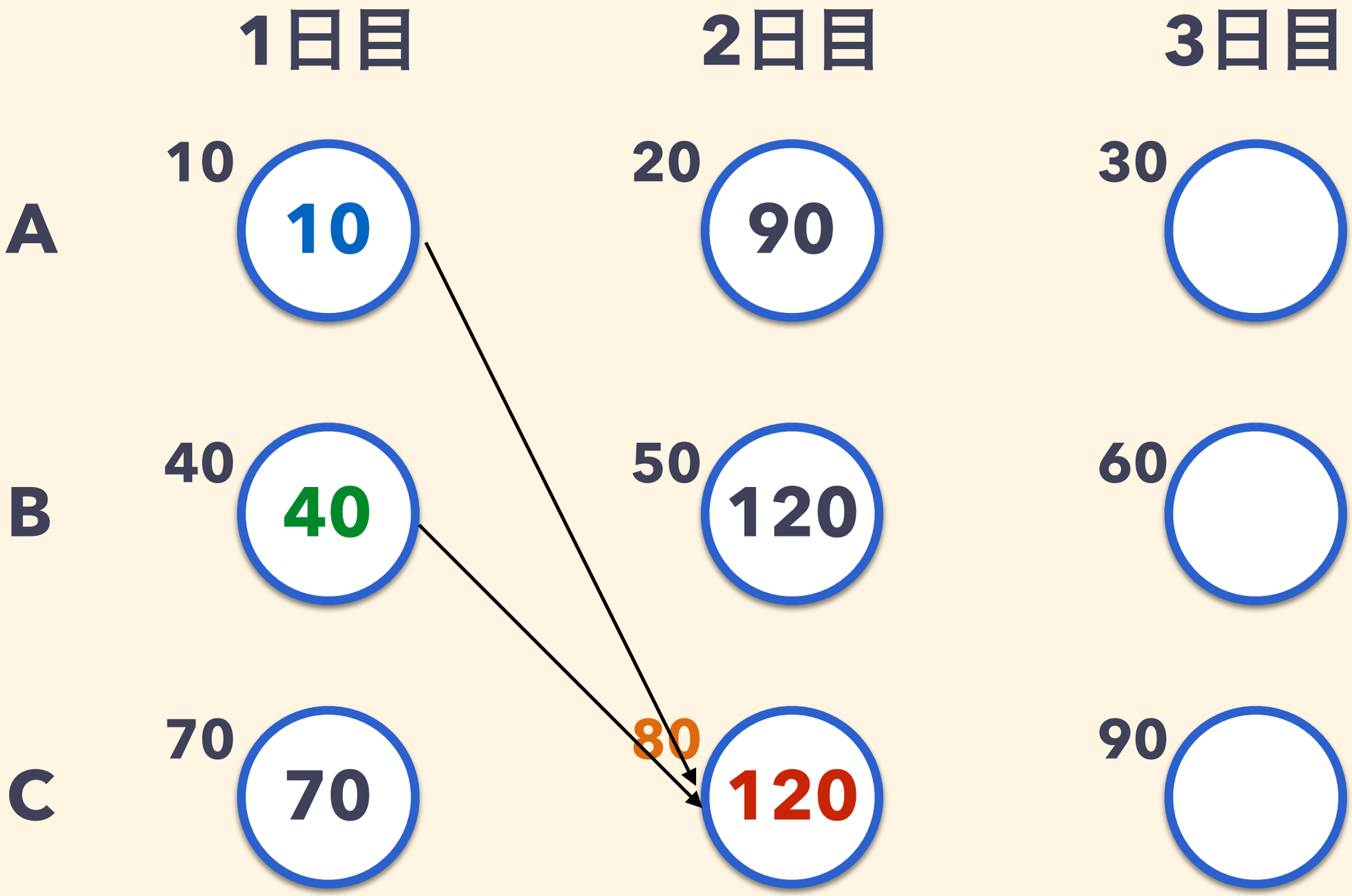


- 1日目にA　60(=10+50)

- 1日目にC 120(=70+50)

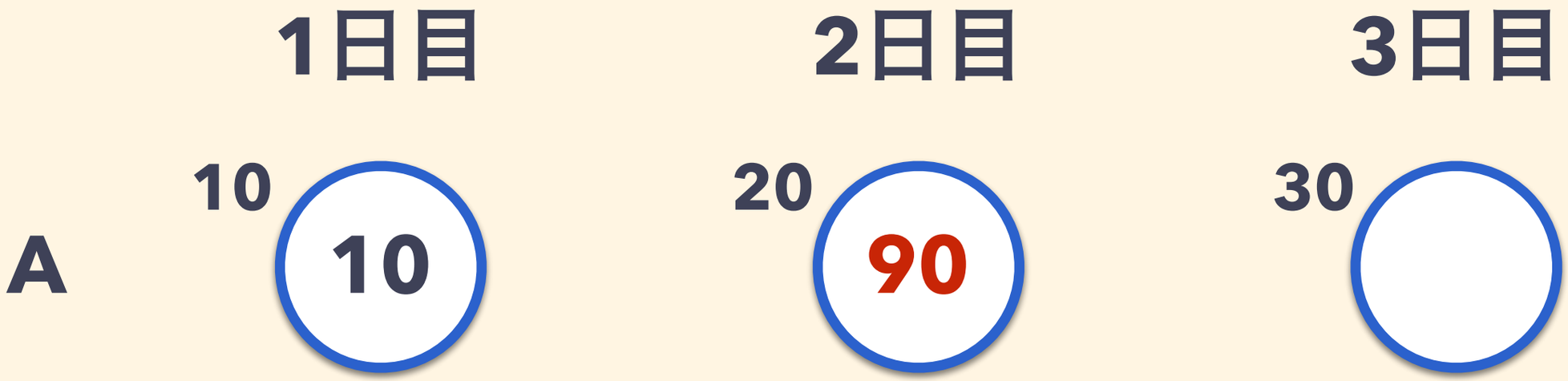
2日目にBを選んだ時の最大値

図で見ると



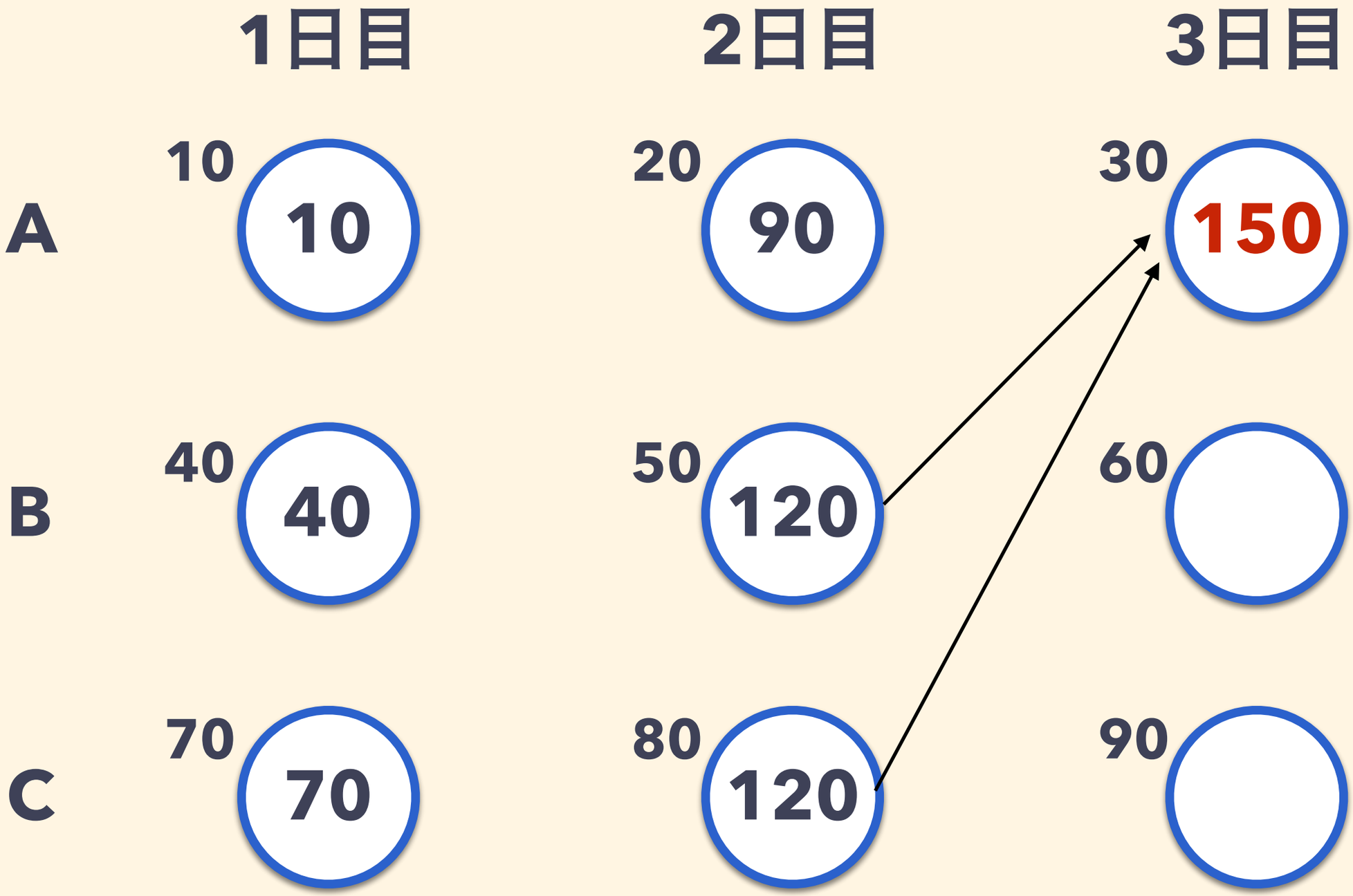
2日目にCを選んだ時の最大値

図で見ると



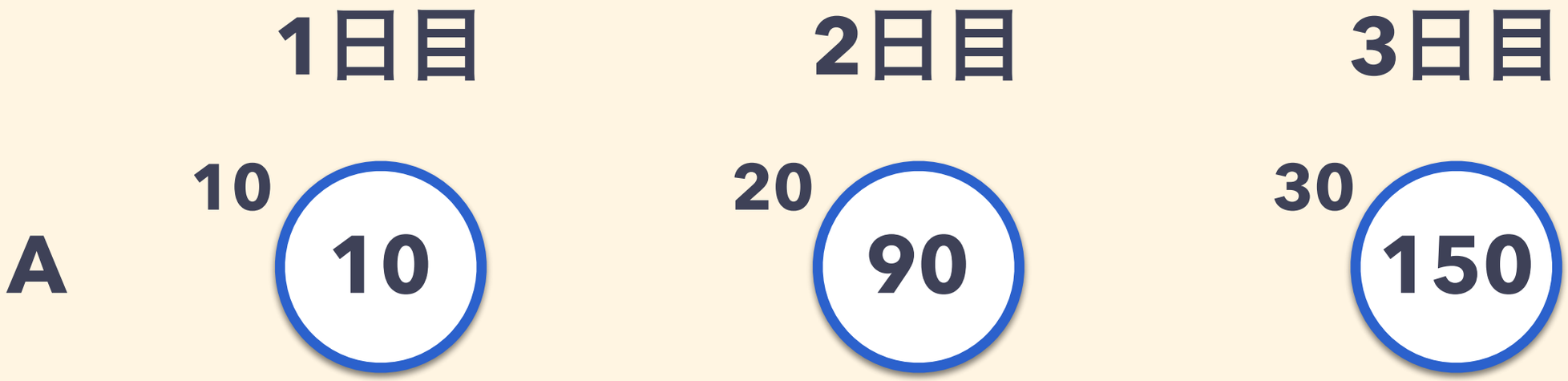
2日目に各活動を選んだ時の最大値

図で見ると



3日目にAを選んだ時の最大値

図で見ると

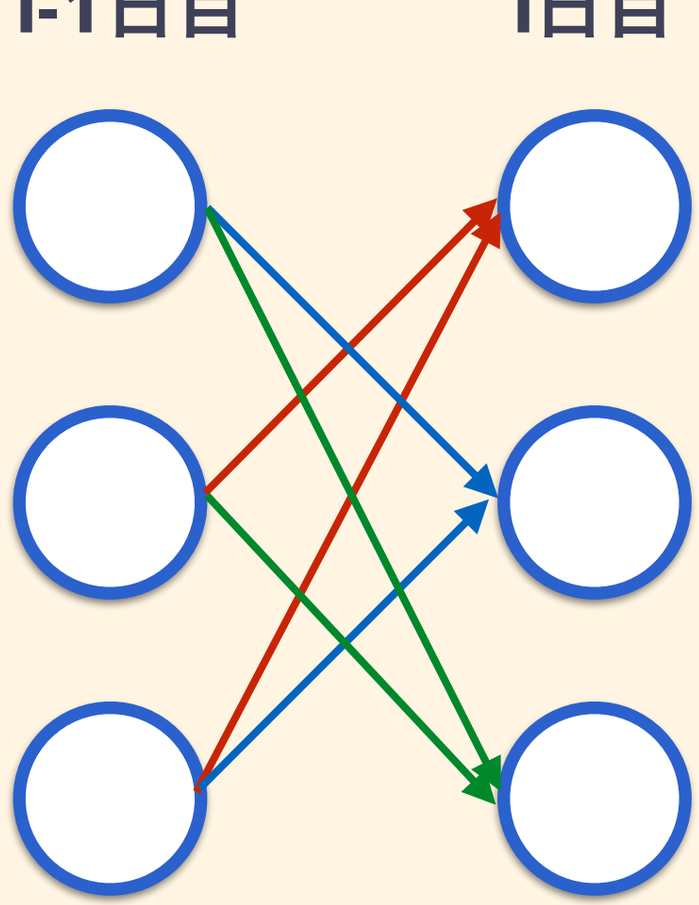


3日目に各活動を選んだ時の最大値

計算量

- i - 1日目の情報からi日目の情報を求める時,

  下図の様に6個遷移がある



計算量

- i - 1日目の情報からi日目の情報を求める時,
  下図の様に6個遷移がある

- 1日目からN日目まで繰り返せば良いので6N

- よって計算量はO(N)

実装

- 部分問題を解いた結果を配列に入れる
  - dp[i][j] := i日目に活動 j をした時の幸福度最大値

- 部分問題を解いた結果を使って，
  少し大きい部分問題を解く
  - i 日目までの幸福度最大値は，i-1 日目までの結果で求まる

$$dp[i][0] = a[i] + \max \begin{cases} dp[i-1][1] \\ dp[i-1][2] \end{cases}$$

実装（ボトムアップ）

```
int dp[100001][3];

// 1日目の計算
dp[1][0] = a[1]; dp[1][1] = b[1]; dp[1][2] = c[1];
// 2日目以降の計算
for (int i = 2; i <= N; i++) {
  // i日目にAを選んだ時の幸福度最大値
  //  = i-1日目のB,Cを選んだ時の幸福度最大値のmax + a[i]
  dp[i][0] = max(dp[i-1][1], dp[i-1][2]) + a[i];

  // B, Cも同様に
  dp[i][1] = max(dp[i-1][0], dp[i-1][2]) + b[i];
  dp[i][2] = max(dp[i-1][0], dp[i-1][1]) + c[i];
}
// N日目にA,B,Cを選んだ時の最大値のmaxが答え
cout << max({dp[N][0], dp[N][1], dp[N][2]}) << endl;
```

実装（トップダウン）

```
// n日目に活動actionをした時の幸福度の最大値
// 2ページ前の漸化式をそのまま書けば良し
int solve(int n, int action) {
  if (n == 0) return 0; // 0日目は便宜上0に
  if (action == 0) {
    return max(solve(n-1, 1), solve(n-1, 2)) + a[n];
  }
  if (action == 1) {
    return max(solve(n-1, 0), solve(n-1, 2)) + b[n];
  }
  if (action == 2) {
    return max(solve(n-1, 0), solve(n-1, 1)) + c[n];
  }
}
```

実装（トップダウン）メモ化

```
int memo[100001][3]; // -1に初期化
// 一度計算した結果は使い回して高速化（メモ化）
int solve(int n, int action) {
  int& cache = memo[n][action];
  if (cache != -1) return cache;
  if (n == 0) return 0;
  if (action == 0) {
    return cache = max(solve(n-1, 1), solve(n-1, 2)) + a[n];
  }
  if (action == 1) {
    return cache = max(solve(n-1, 0), solve(n-1, 2)) + b[n];
  }
  if (action == 2) {
    return cache = max(solve(n-1, 0), solve(n-1, 1)) + c[n];
  }
}
```

まとめ

- 対象となる問題を複数の部分問題に分割し、
  部分問題の計算結果を記録しながら解いていく
- 部分問題の定義と更新（遷移）の仕方が重要
- 実装はn次元配列を定義して,
  良い感じに更新していく

1問解きます

付録

- 題材にした問題
  - Educational DP Contest C - Vacation
    https://atcoder.jp/contests/dp/tasks/dp_c
- 提出コード
  - https://atcoder.jp/contests/dp/submissions/16084931

アンケート