応用アルゴリズム演習

動的計画法　/　Dynamic Programming

（２回目ー１１月３０日）

動的計画法とは？

動的計画法（dynamic programming）は，目的とする問題を**より小さな部分問題の解を利用して解いていく**手法である．その際，**部分問題の解を逐次記録ておき**，途中で**同じ問題が現れた時に**記録してある解を利用することで**計算の重複を防ぎ，全体の計算量を削減**しようというアイデアである．

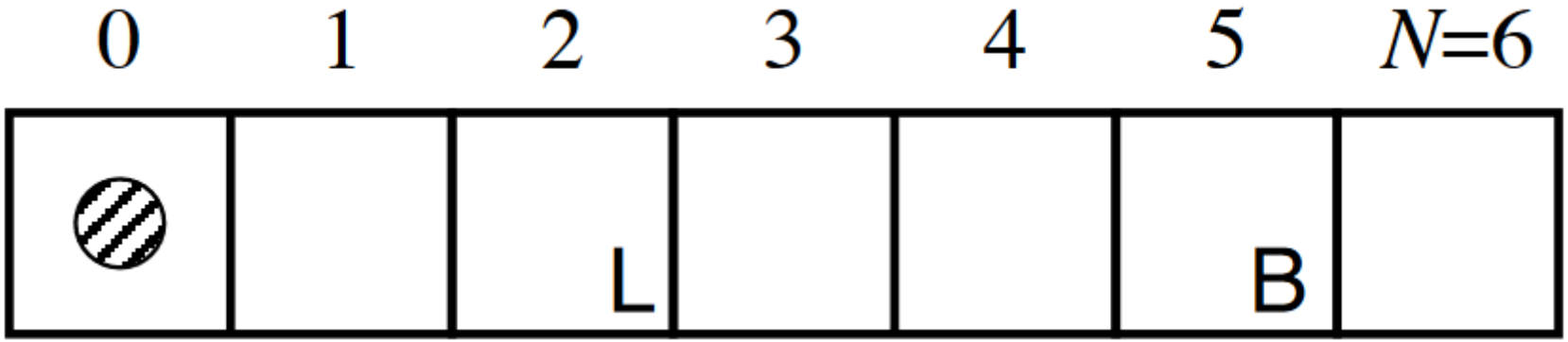
[アルゴリズムとデータ構造]から引用

実は，素手に経験したことがあります！

例から学びましょう　**Backgammon問題**

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：

- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る



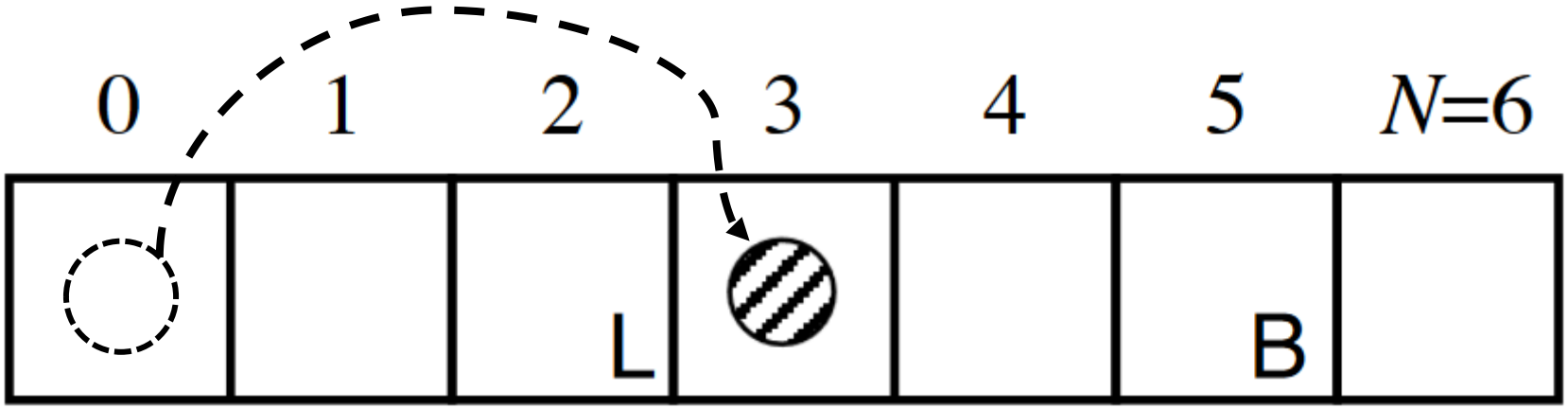
例から学びましょう　**Backgammon問題**

サイコロ：

Turn 1: 3

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：

- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る



例から学びましょう　**Backgammon問題**

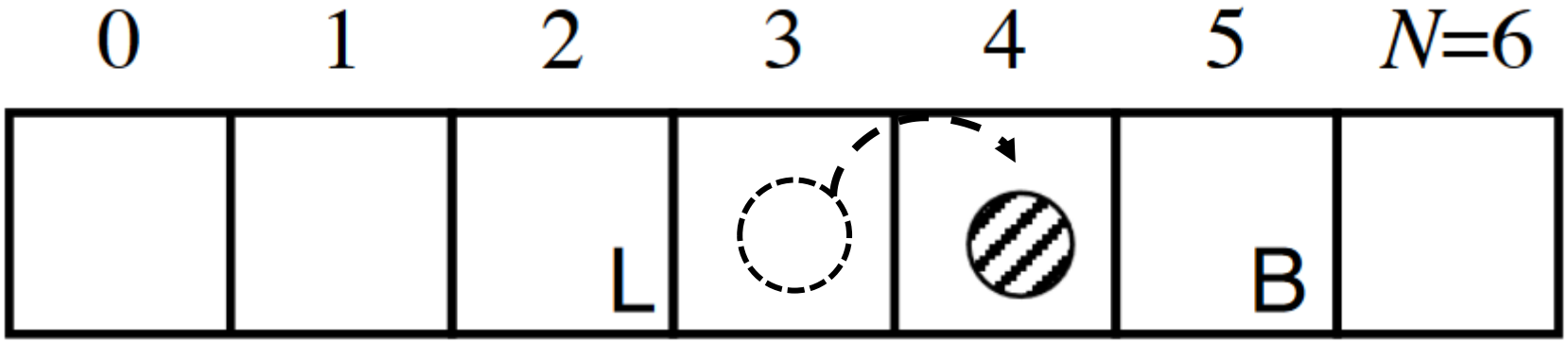
サイコロ：

Turn 1: 3

Turn 2: 5

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：

- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る



例から学びましょう　**Backgammon問題**

サイコロ：

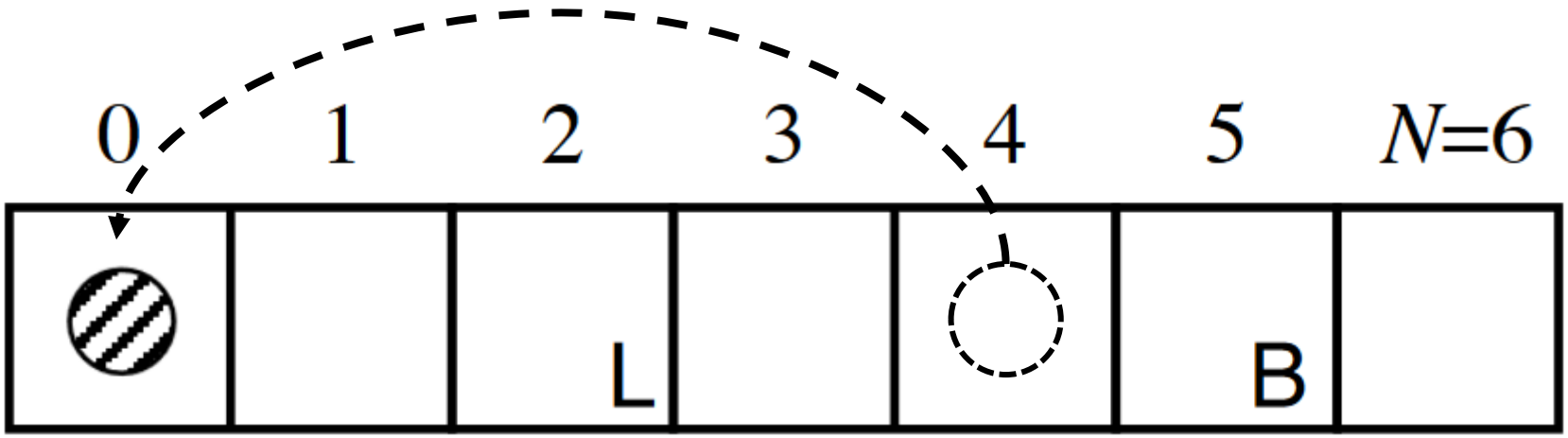
Turn 1: 3

Turn 2: 5

Turn 3: 1

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：

- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る



例から学びましょう　**Backgammon問題**

サイコロ：

Turn 1: 3

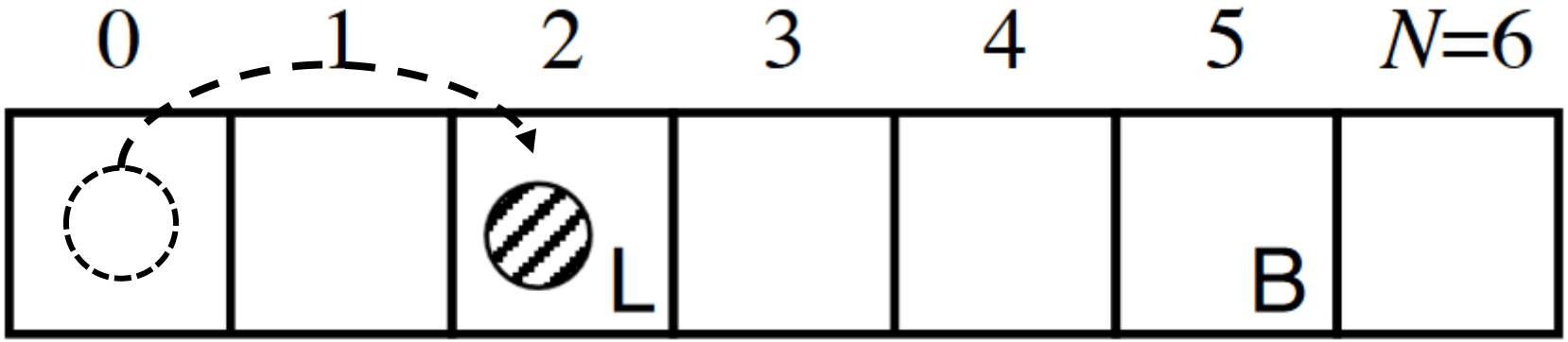
Turn 2: 5

Turn 3: 1

Turn 4: 2

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：

- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る



例から学びましょう　**Backgammon問題**

サイコロ：

Turn 1: 3

Turn 2: 5

Turn 3: 1

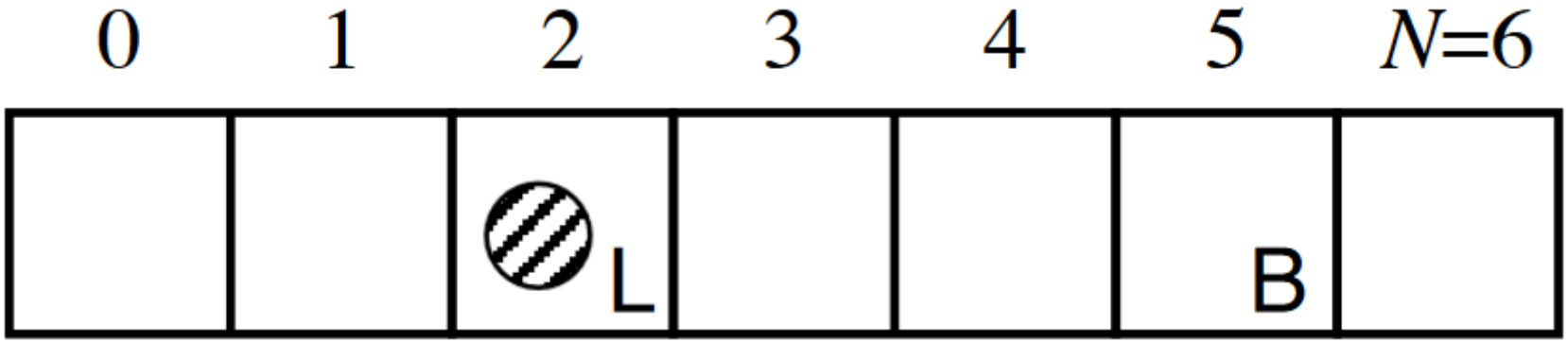
Turn 4: 2

Turn 5: /

動かない！

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB：

- **L：Lose one turn** 一回休む
- **B：Back to the start** スタートに戻る



例から学びましょう　**Backgammon問題**

サイコロ：

Turn 1: 3

Turn 2: 5

Turn 3: 1

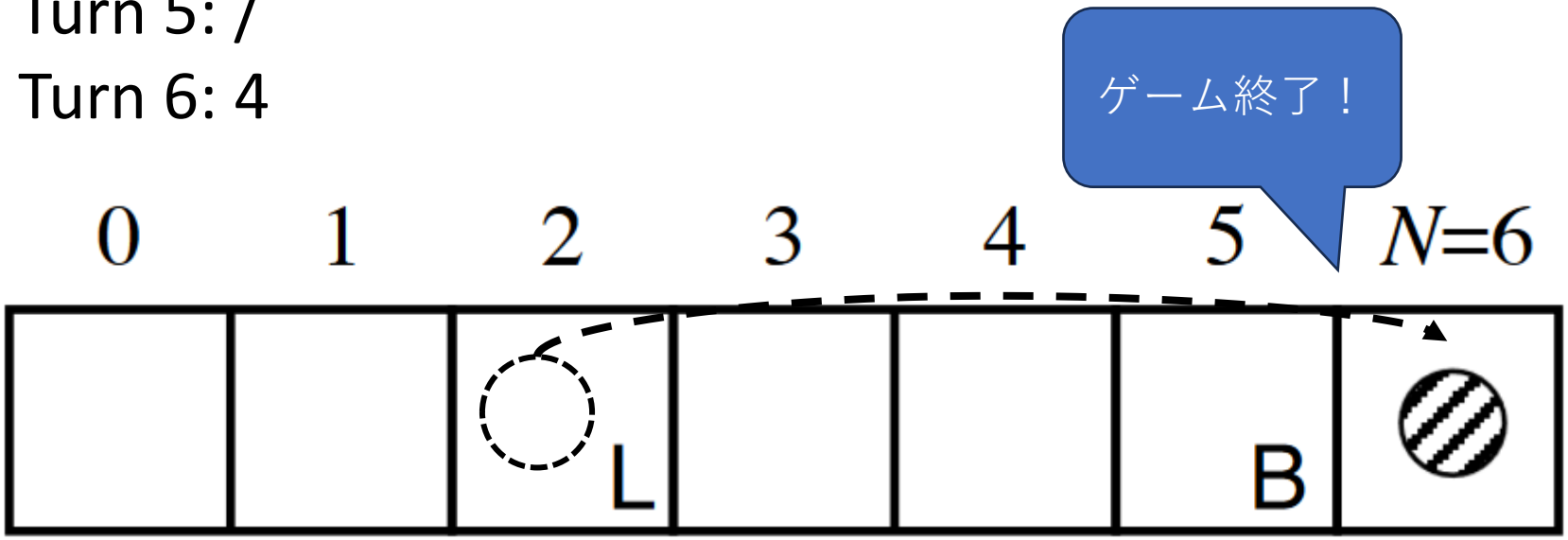
Turn 4: 2

Turn 5: /

Turn 6: 4

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB：

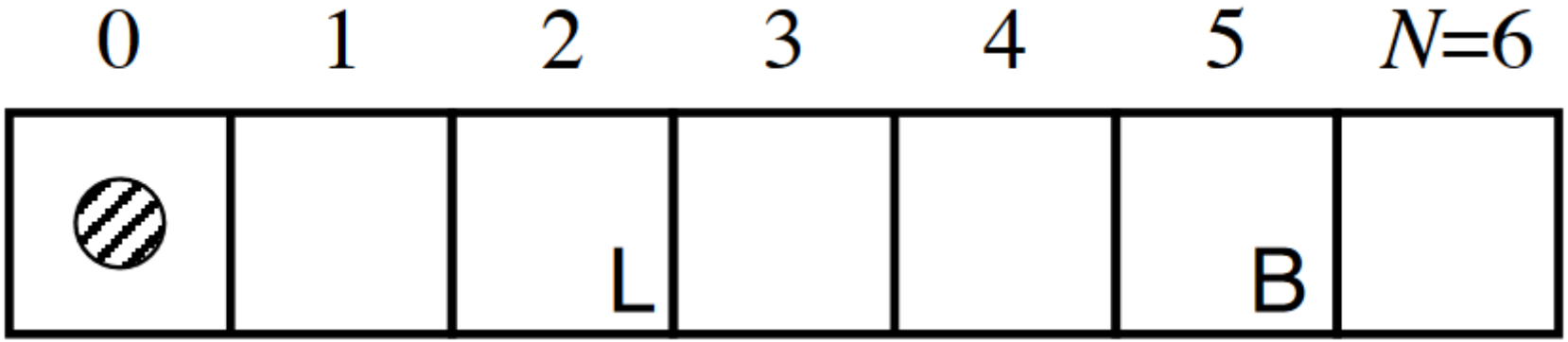
- **L：Lose one turn** 一回休む
- **B：Back to the start** スタートに戻る



例から学びましょう　**Backgammon問題**

サイコロを振って、目の数だけ進む．ゴールを超えた場合はその分戻る．特別なルールが２つある、LとB ：
- **L：L**ose one turn 一回休む
- **B：B**ack to the start スタートに戻る

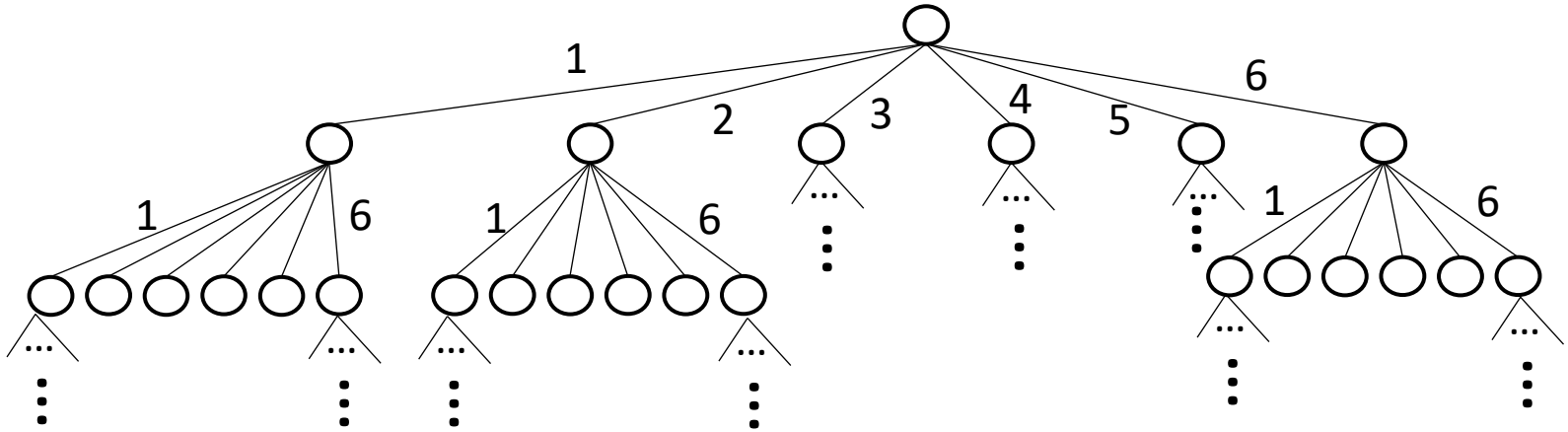
- **問題：**任意のボードに対して、任意 *T* ステップ以内でゴールにたどり着く確率は？
  - ボードサイズ *N* は，最大100
  - ステップ数 *T* は，最大100



提案１　（あまり良くない案）

- 全ての組み合わせを確認しましょう！
  - ゴールに着く組み合わせを数えながら探索

木の深さは：
ステップ数 T



$T = 5$ の場合：

11111
11112
…
11116

11121
11122
…

…

…
66664
66665
66666

時間計算量　空間計算量

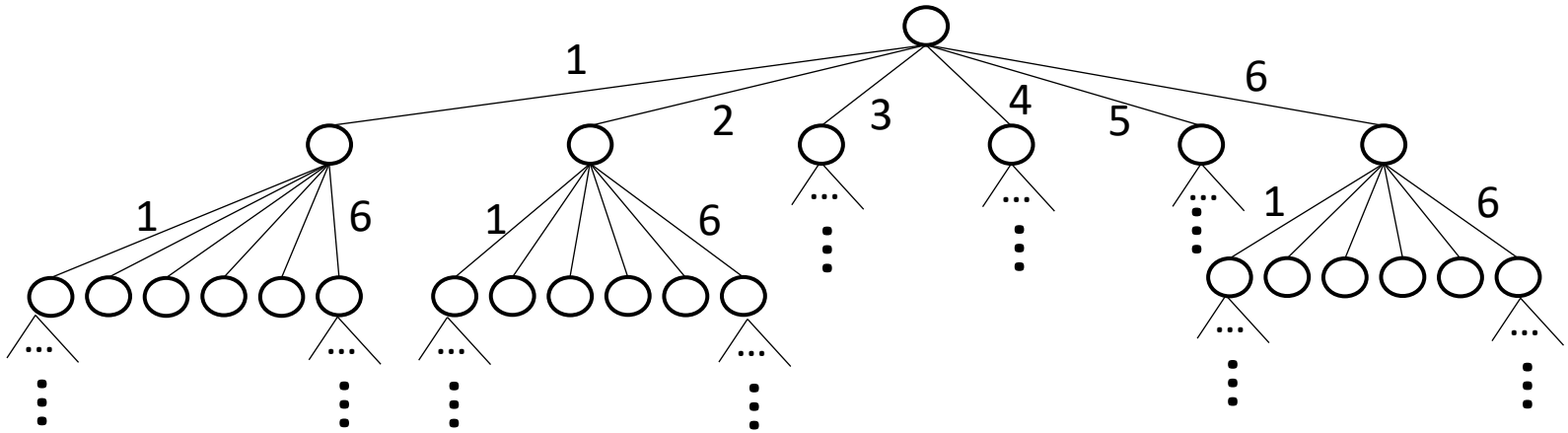
- **時間計算量**
  - 入力サイズに対して，アルゴリズムの計算時間はどうなる？

- **空間計算量**
  - 入力サイズに対して，アルゴリズムが要求するメモリがどうなる？

提案１　（あまり良くない案）

- **時間計算量**
  - 木のノード数

木の深さは：
ステップ数 T

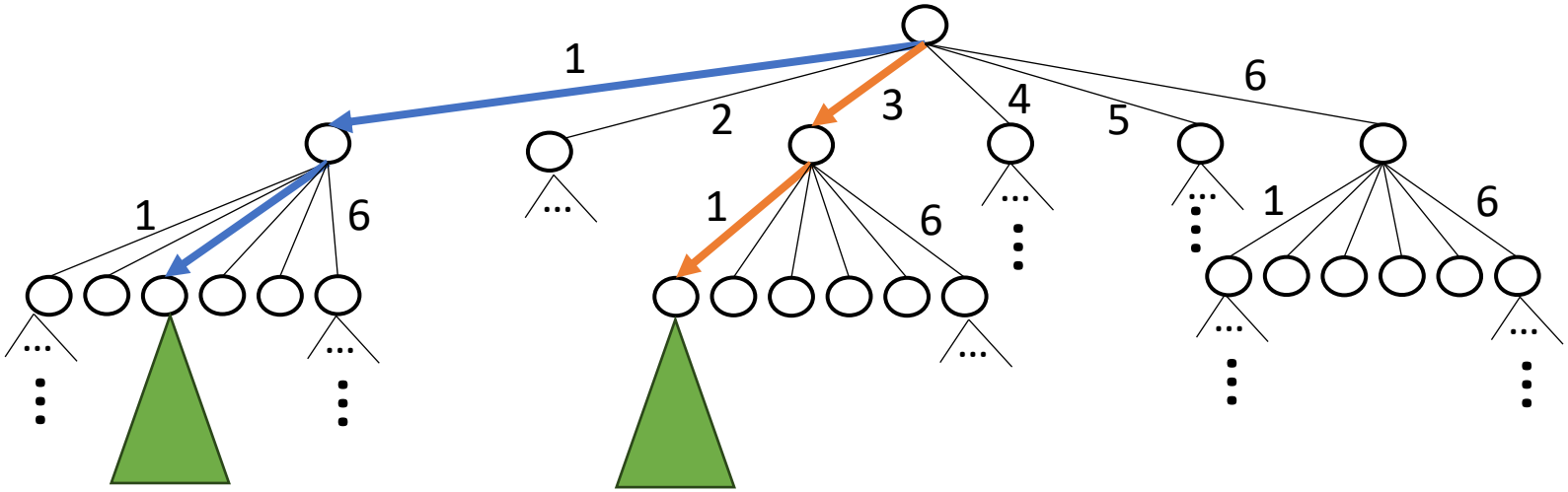


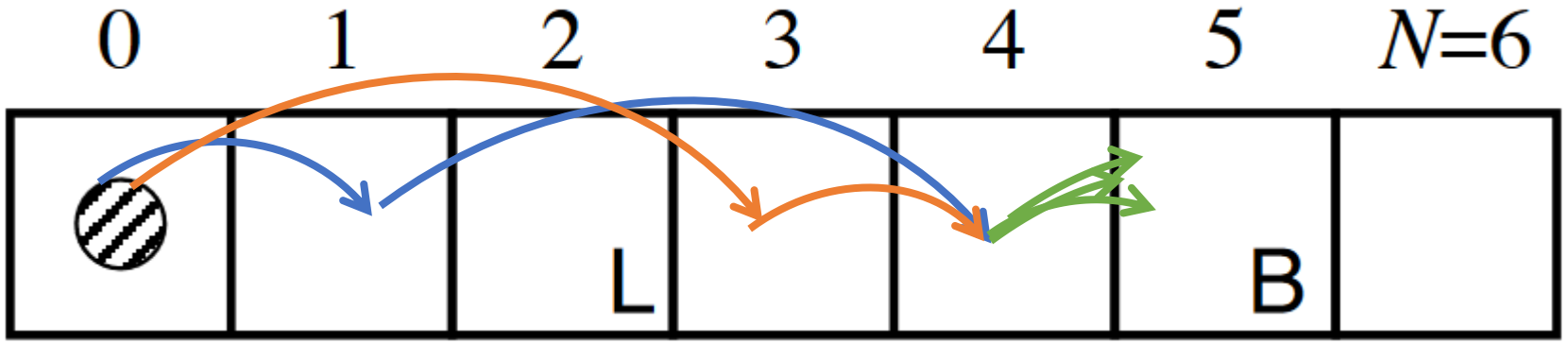
時間計算量 $O(6^T)$

- ステップ数 $T = 5$ の場合　$6^5 = 7776$
- 最大ステップ数 $T = 100$ の場合　$6^{100} \approx 6.5\ 10^{77}$

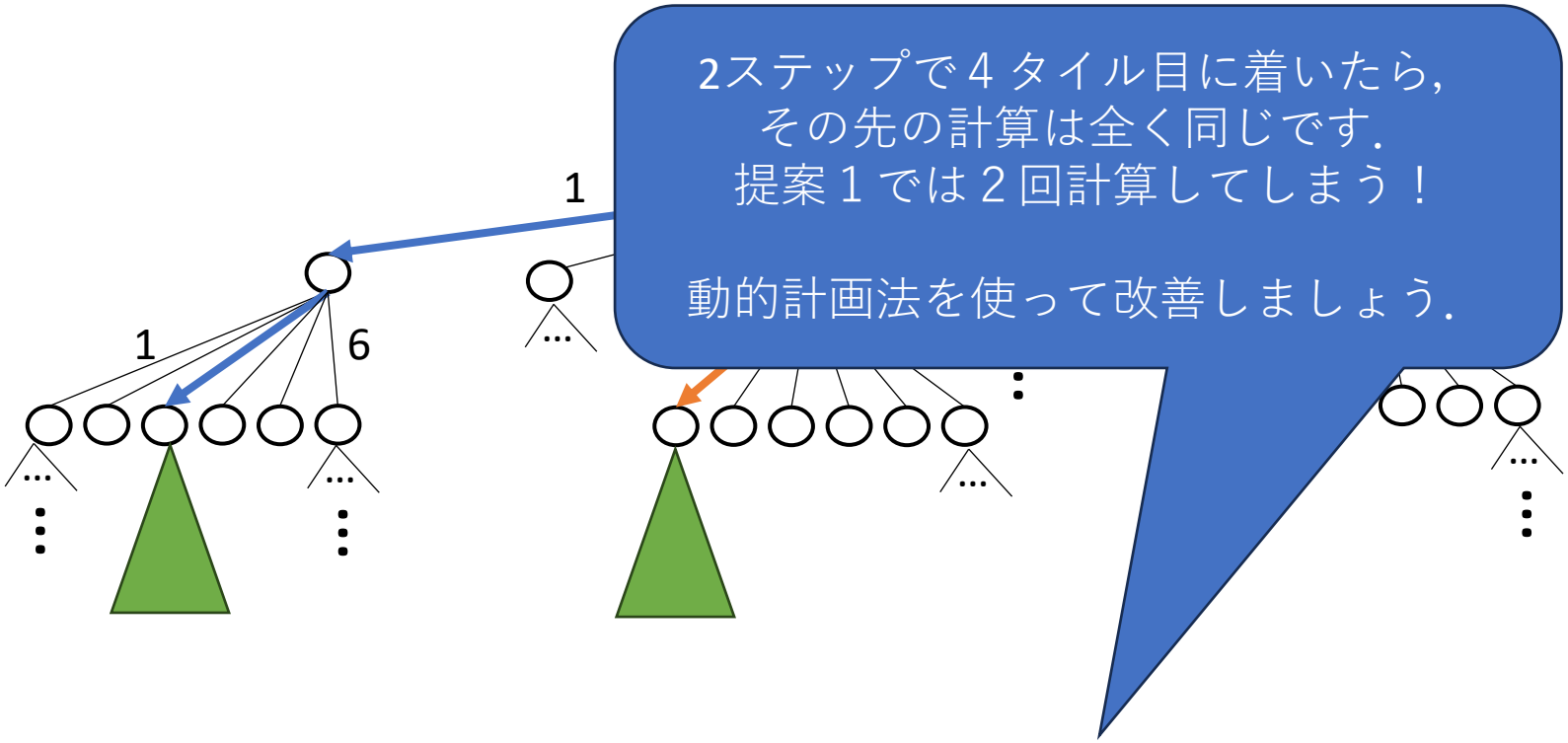
無理！

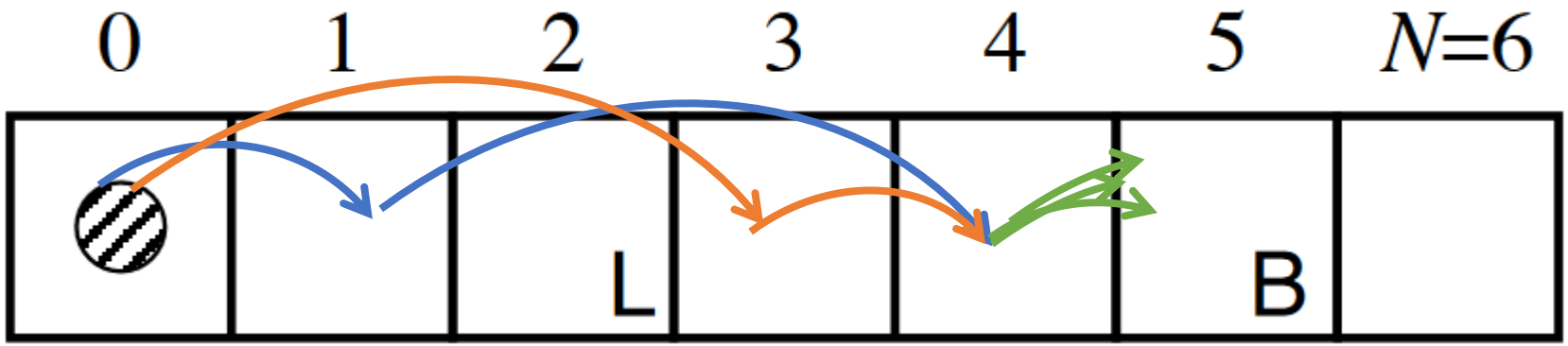
提案１（重複する部分問題）





提案１（重複する部分問題）





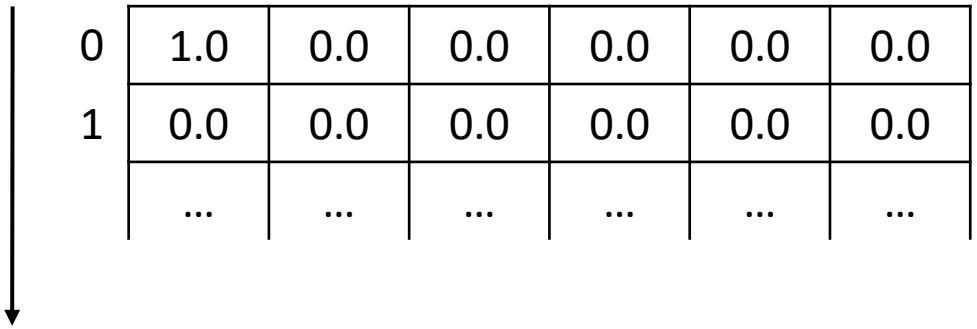
提案２

- ２次元配列 **prob**

prob[s][t] でステップ **s** でタイル **t** にいる確率を記録

prob[0][0] = 1.0; その他 **0.0**

ステップ

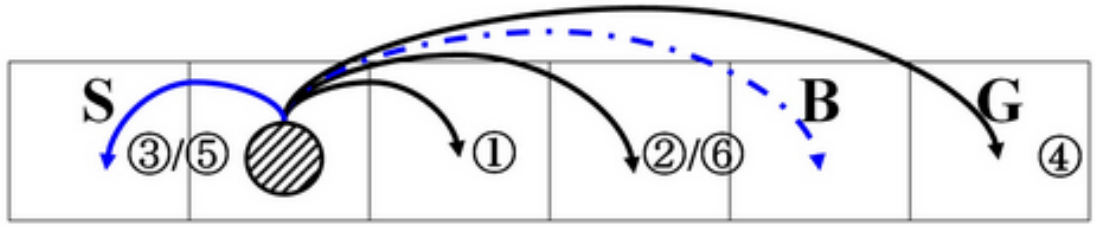


0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
	…	…	…	…	…	…

prob[s+1][0~5] を **prob[s+1][0~5]** から求める

提案２

`prob[s+1][0 ~ N-1]` を `prob[s][0 ~ N-1]` から求める

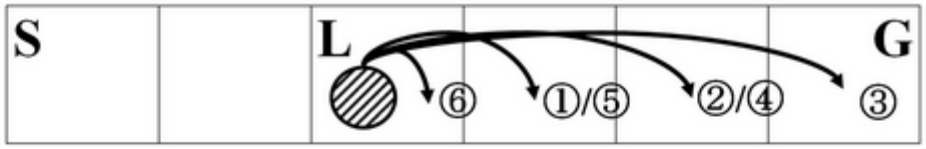


ステップ

s	.	p	.	.	.	.
s+1	+2p/6		+p/6	+2p/6		+p/6

提案２　L タイル（lose one turn）の扱い

`p[s+1][0 ~ N−1]` を `p[s][0 ~ N−1]` から求める
→ `p[s+2][0 ~ N−1]` に確率を足す



ステップ

S	.	.	p	.	.	.
S+1						
S+2			+p/6	+2p/6	+2p/6	+p/6

提案２

ソースコード

```
double solve(int n, int t) {
  /* 確率の配列を全部0.0に初期化　Initialize probability array to 0.0 */
  /* (省略) */
  /* 最初のステップに、スタートにいる確率は1.0です */
  prob[0][0] = 1.0;

  for(i=0; i<t; i++) { /* iはステップ数です i is the step number*/
    for(j=0; j<n; j++) { /* jはボードにおいているところ j is the placement on the board */
      int k; /* kはトランプの価値 k is the dice value */
      for(k=1; k<=6; k++) {
        int nPos = (j+k>n)? 2*n-k-j: j+k; /* ゴールを超えた場合 Check case we go beyong the Goal */
        int nStep = (board[nPos]==L)? i+2: i+1; /* 今Lにいるの場合は、+2のステップに動く */
        if(board[nPos]==B) {
          /* 着くタイルはBの場合は、スタート（０）に戻る*/
          nPos = 0;
        }
        prob[nStep][nPos] += prob[i][j] / 6.0;
      }
    }
  }
  /* 各ステップにゴールにたどり着く確率を合計する Sum the probability of reaching the goal at each step */
  double result = 0.0;
  for(i=0; i<=t; i++) {
    result += prob[i][n];
  }
  return result;
}
```

ステップのループ

タイルのループ

サイコロのループ

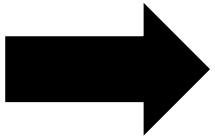
ボードのルールに従ってnPos / nStepを求める

1/6 の今にいる確率を足す

最終結果を計算

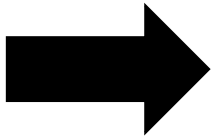
提案２

- **時間計算量**
  - 2重ループいくつか
  - **3重ループ**
    - ステップ数　T
    - ボードの幅　N
    - サイコロの面数（定数→検討しない）



$O(T \times N)$

- **空間計算量**
  - １次元配列（ボード）
  - **２次元配列**
    - ステップ数　T
    - ボードの幅　N



$O(T \times N)$

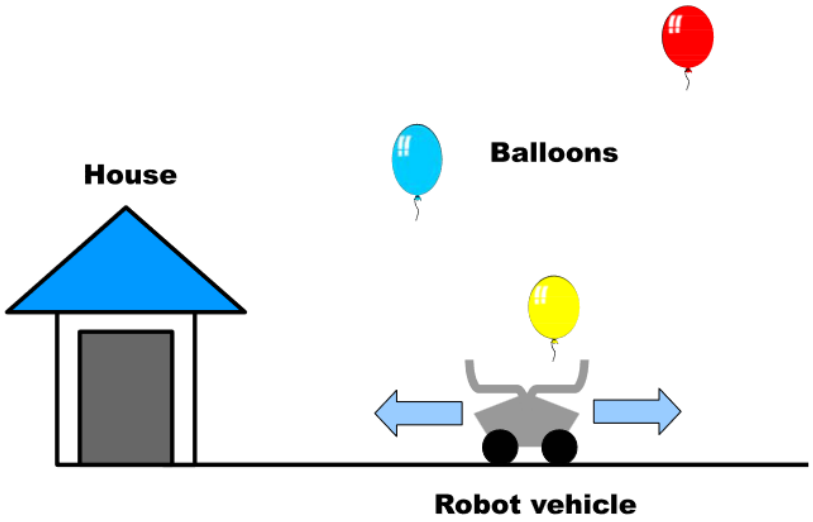
工夫すれば，過去２ステップの情報だけを保存 → $O(N)$

課題３風船回収

上から落ちて来る風船をロボットが家に集める.

- 風船が落ちる場所と時刻が与えられる.
- ロボットは左右に動ける.
- 運んでいる風船の数 k によって速度が変わる：座標を１移動するのに、k+1 時間かかる.
- ロボットは風船を最大３個運べる.

家は場所０にある.
スタート時刻０では，ロボットが場所０にいる.
風船を家に置くこと，そして風船を拾うことにも時間がかからない．つまりロボットが風船の落ちる場所に風船と同時刻に着いた場合はセーフ.



入力例

```
3 // 風船は３個落ちてくる
100 150 // 場所 100 に時刻 150
10 360  // 場所  10 に時刻 360
40 450  // 場所  40 に時刻 450
```

疑問

- 風船をすべて回収出来る場合，**最小移動距離**を答える.
- すべて回収出来なかった場合は，**何個目で回収不能になるか**を答える.

課題３風船回収　難易度は？

- 最短距離？出来るだけホームに戻らないことにすれば…?

  → … ダメです．次の風船を回収するのに，数ステップ前に持っていた風船を降ろさないといけない場合がある．

- じゃ，次の風船が落ちるまでホームへ行く余裕があれば今持っている風船を降ろすことにします！

  → …それもダメです．最短距離にならない場合がある．

上記のような戦略では全てのケースに対して正しい回答を求めない．

- じゃ，毎回風船を降ろす降ろさないケースを全部試せばいい…?

  → … それも良くないです．風船の最大数は100，よってBackgammonの提案１みたいに探索木を作れば，2^{100} ノード．時間計算量的には無理です．

課題３風船回収　難易度は？

- 解け方はBackgammonに結構似ています.
  - 具体的なヒントを次回出す

- それまでに，解け方を完全に分からなくても，問題を分割しましょう：
  - 特定の場所から，次の風船が落ちるまでに当場所に移動できるか？
    - 今持っている風船影響
    - ホームに戻った場合
  - 同様，その距離
    - 直接に行った場合
    - ホームに戻った場合

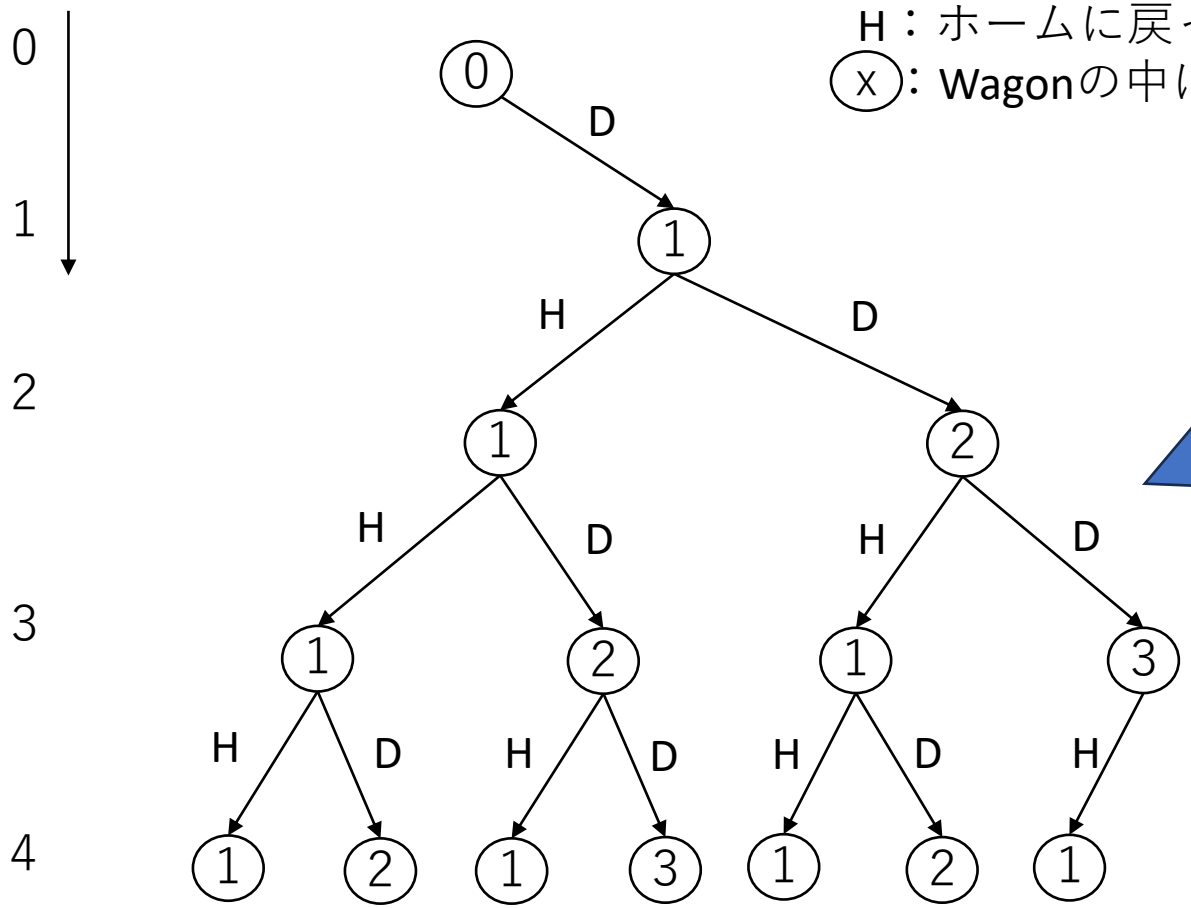
そのような関数等を次の授業までに完成しましょう！

補足

次の 4 ページ

風船回収の部分問題の重複

回収した
風船の数

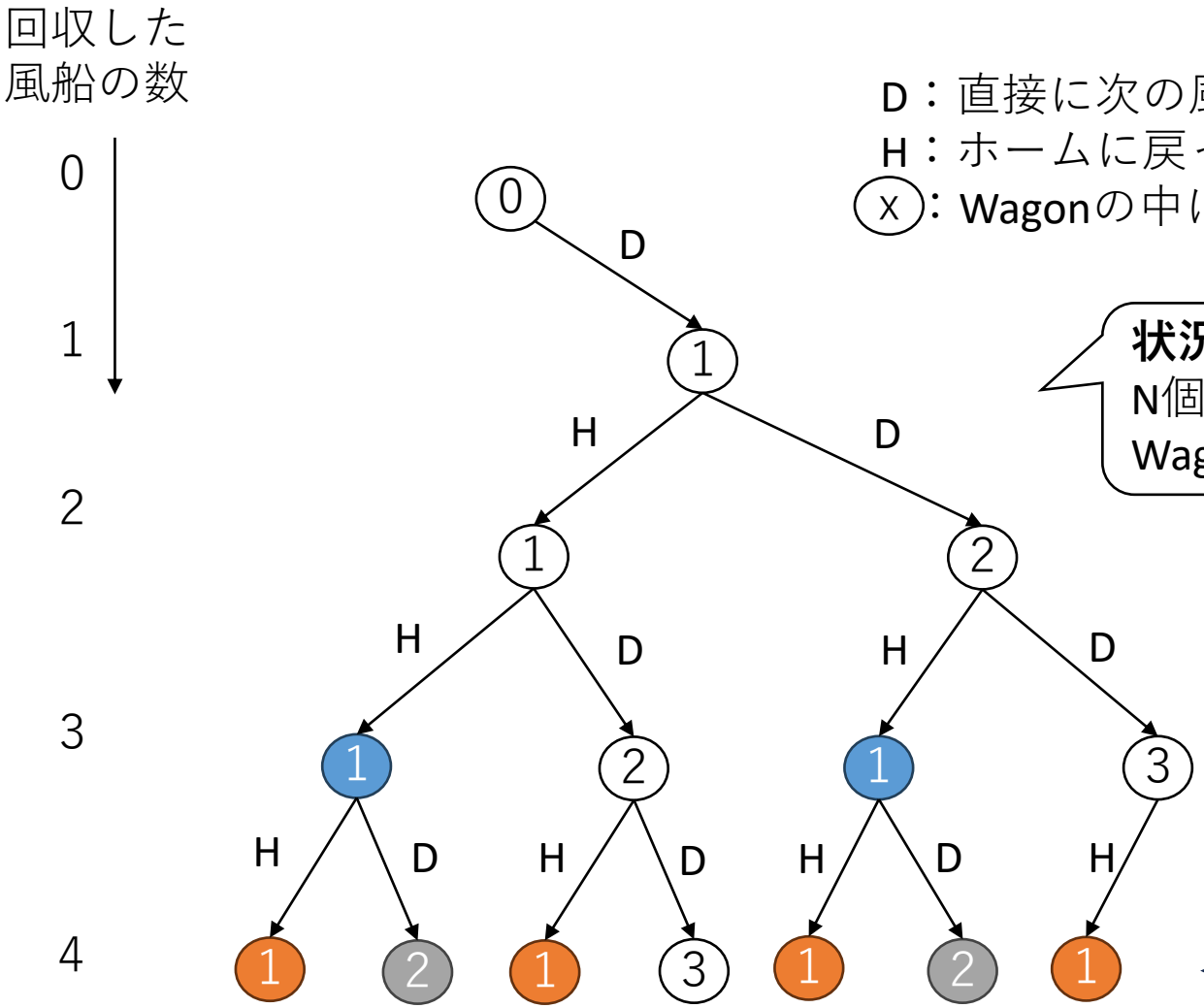


D：直接に次の風船を取りに行った
H：ホームに戻って次の風船を取りに行った
(x)：Wagonの中に風船がX個あります

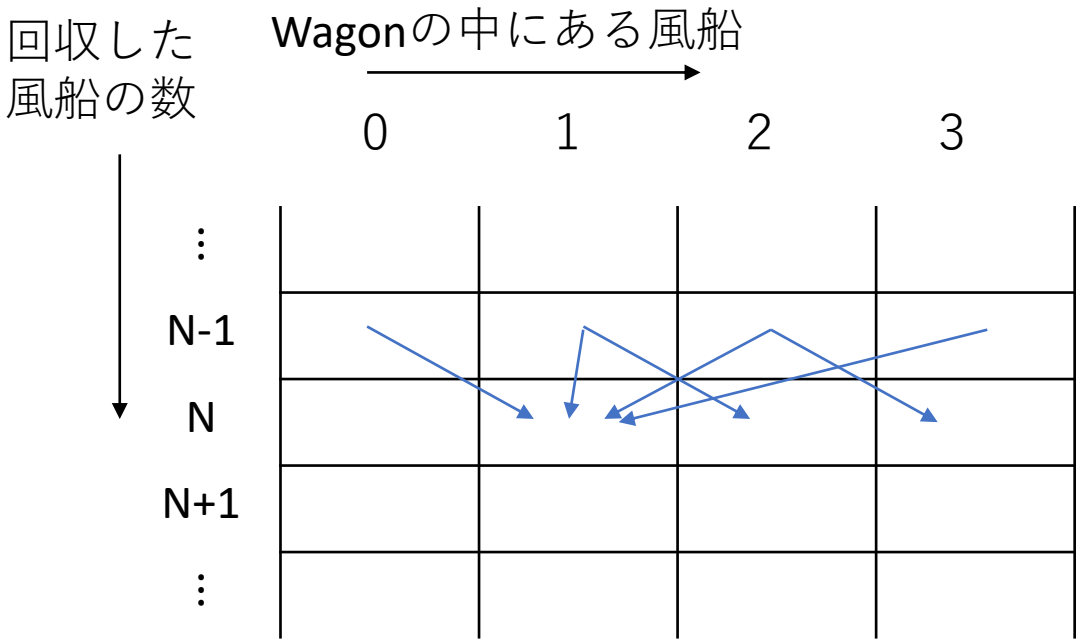
仮定：DとH両方次の風船に間に合うこと

（間に合わない＝その部分木は存在しない）

風船回収の部分問題の重複



一般的に考えると



2次元配列で最短ルートを記録しましょう！

ソースコードのイメージ

```
風船のループ {
    Wagonの中にある風船のループ {
        直接に取りに行くか

        ホームに戻って取りに行くか
    }
}
```

検討事項：

- 配列の初期値
- 状況に至らないの扱い
- 風船を回収不能な場合

レポートに向けて

- 前のページの「検討事項」をレポートの中に説明してください

- アルゴリズム自体は難易度が高くはないが，引っ掛かりそうなところが多い！
  - 距離の計算
  - 風船を回収できるかどうかの確認
  - …

→ご安心してください
- 第１締め切りでアルゴリズムの理屈を分かったことをレポートの中に示して，実装も頑張って
- 第２締め切りで回答例と比較して，誤ったところを特定しましょう

上記出来たら**ほぼ満点**として評価します

課題３

問題１

kadai3/balloon.c に実作のデータ構造を追加，solve 関数を実装

- Balloon.c を提出
- レポートの中に：
  - 実装したアルゴリズムの説明（半ページ程度）
  - コンパイル法
  - 実行結果

問題２

実装したアルゴリズムの時間計算量と空間計算量分析

- 風船の数 n （今回は n は40以下）
- 位置の取りうる値の幅 P (今回は 100 以下)
- 風船の落ちてくる時間の幅 T (今回は50000以下)

レポートの中に，根拠を含めて説明すること，
ある変数が影響はなかった場合，明確に「影響はない」と記載すること

課題３

締め切りについて

問題１ → 12月7日 12:00　**全員提出（途中でも）**

- 問題１を解けた場合は，問題２も一緒に提出していい

回答例を 12月7日12:00に公開する

問題１の再提出及び問題２ → 12月13日23:59

- 問題１：正解例と比較して，間違っていた箇所をどのように実現していたか解説すること
- 問題２：公開した回答例で解けること