

券売機システムの攻略メソッド

状態遷移とプログラムの読解プロセス（2019年春期 応用情報 午後問3）

【ALERT】 ※本スライドは解説特化型です。解答そのものは記載していません。
自力で正解を導き出すための「思考の設計図」としてご活用ください。

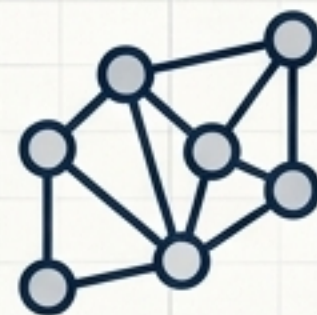
この問題を解くための「3つの神器」



【表1】商品定義

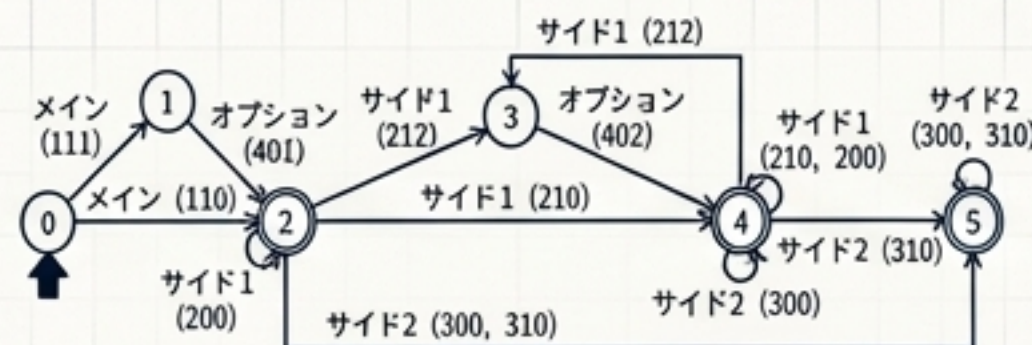
注文（商品名）をシステムが理解できる「数字（イベントコード）」に変換するための辞書。

分類名	メイン商品				
分類番号	名称	N	S	O	価格
商品	牛丼	1	1	1	380
	豚丼	1	1	1	350
	...	...	...	...	...
	...	...	...	...	...



【図2・表2】状態遷移

現在の状態と入力から、次の状態を決めるルートマップ。



【図4】プログラム

マップのルールに従って自動で状態を動かす処理エンジン。

```
status ← 0 // 状態番号
amount ← 0 // 注文金額
discount ← 0 // 割引金額
accepted ← false
for( i を 1 から order_count まで繰り返す )
  amount ← amount + [つ]
  event_code ← [エ] + order[ i ].0
  event_index ← get_event_index( event_code )
  discount ← discount + [オ]
  status ← [カ]
  if( status が E に等しい )
    エラー終了
  endif
endfor
accepted ← status_table[ status ][ ACCEPT_INDEX ].accept
if ( [キ] )
  発券ボタンを押せるようにする
else
  発券ボタンを押せないようにする
endif
```

Hint: 全ての設問は、この3つの要素間の「翻訳」と「照合」で解けます。常に3つの資料を行き来する視点を持ちましょう。

コア・メカニクス：イベントコードの錬成

システムの絶対ルール：券売機は「商品名」ではなく「3桁の数字」で動く。

例：鮭定食の場合

N: 分類番号

S: セット適用

O: オプション

1

+

1

+

0

→



生成コード = [110]

Hint: 問題を解く際は、まず注文された全商品をこの「3桁のコード」に変換する作業から始めましょう。これが全てのスタート地点です。

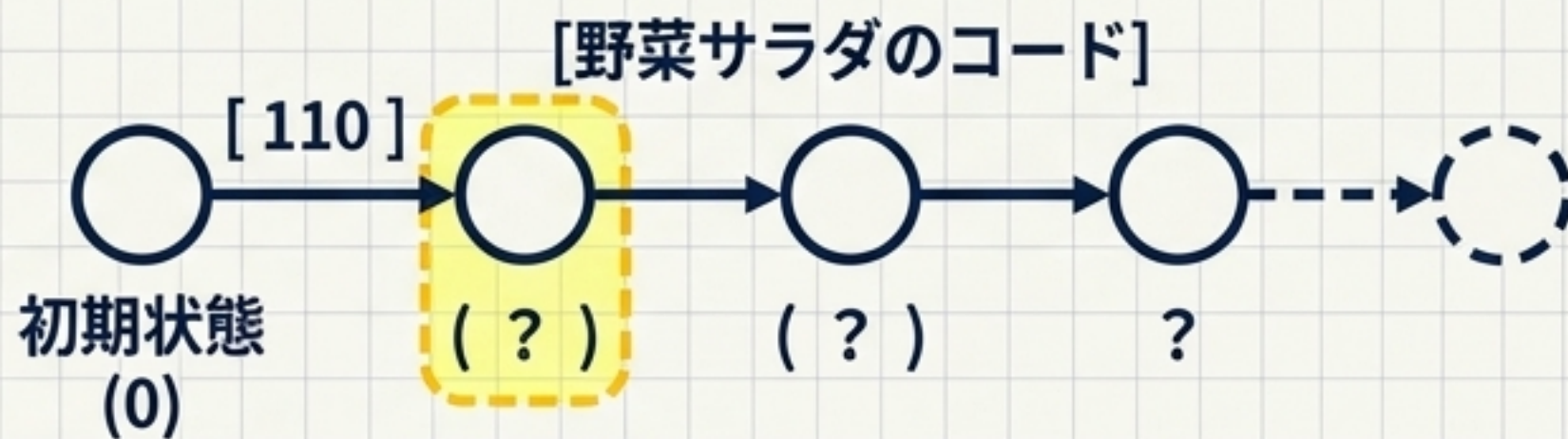
設問1 (1) のヒント：状態遷移のトレース術

Q: 鮭定食→野菜サラダ→ゴマドレ→みそ汁 の遷移順は？

Step 1: コードへ変換

1.	鮭定食	→ [110]
2.	野菜サラダ	→ [?] (表1でN,S,Oを確認)
3.	ゴマドレ	→ [?]
4.	みそ汁	→ [?]

Step 2: ルートの追跡



Hint: 図2を使います。「状態0」から「イベントコード110」の矢印を辿ると、最初の到着地（状態番号）はどこになるか？そこから次の矢印を順番に探しましょう。

設問1 (2)[ア] のヒント：発券可否のサイン

Q：表2の[ア]（状態3の発券可否）に入るのは？

図2の凡例に注目

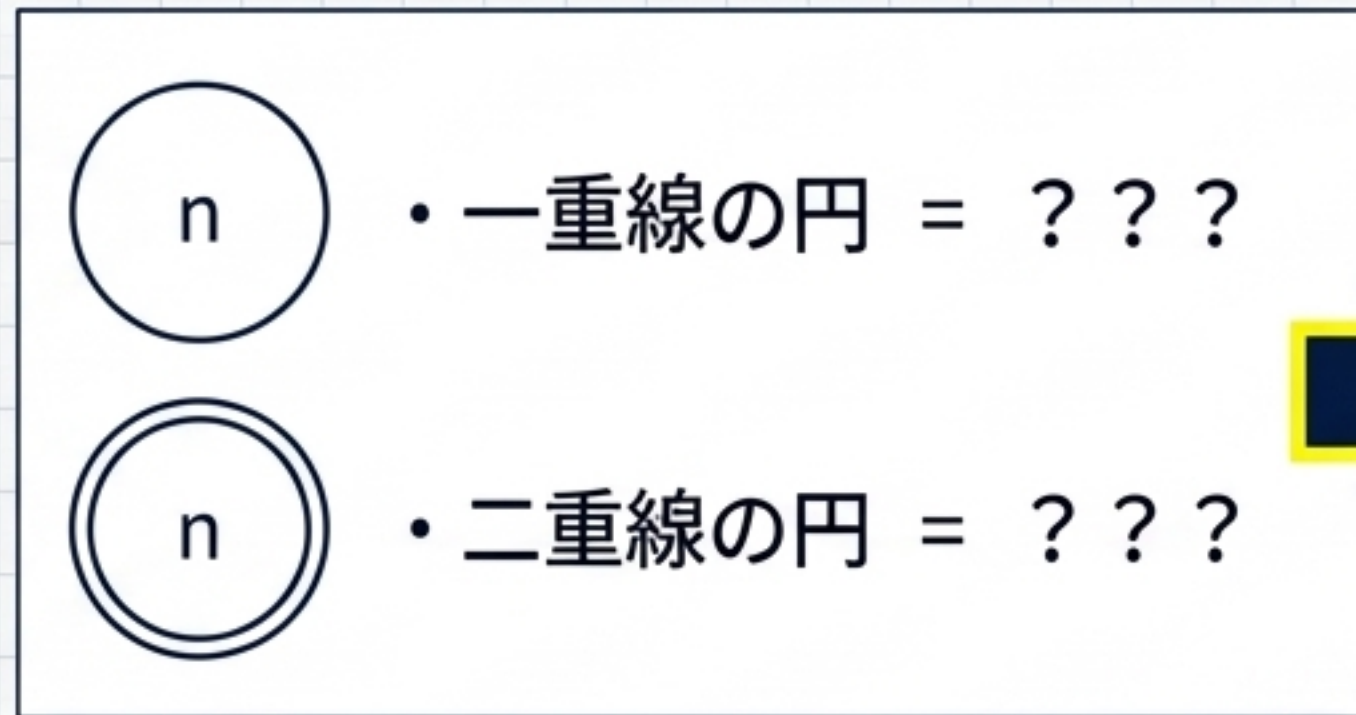


表2のデータと照合

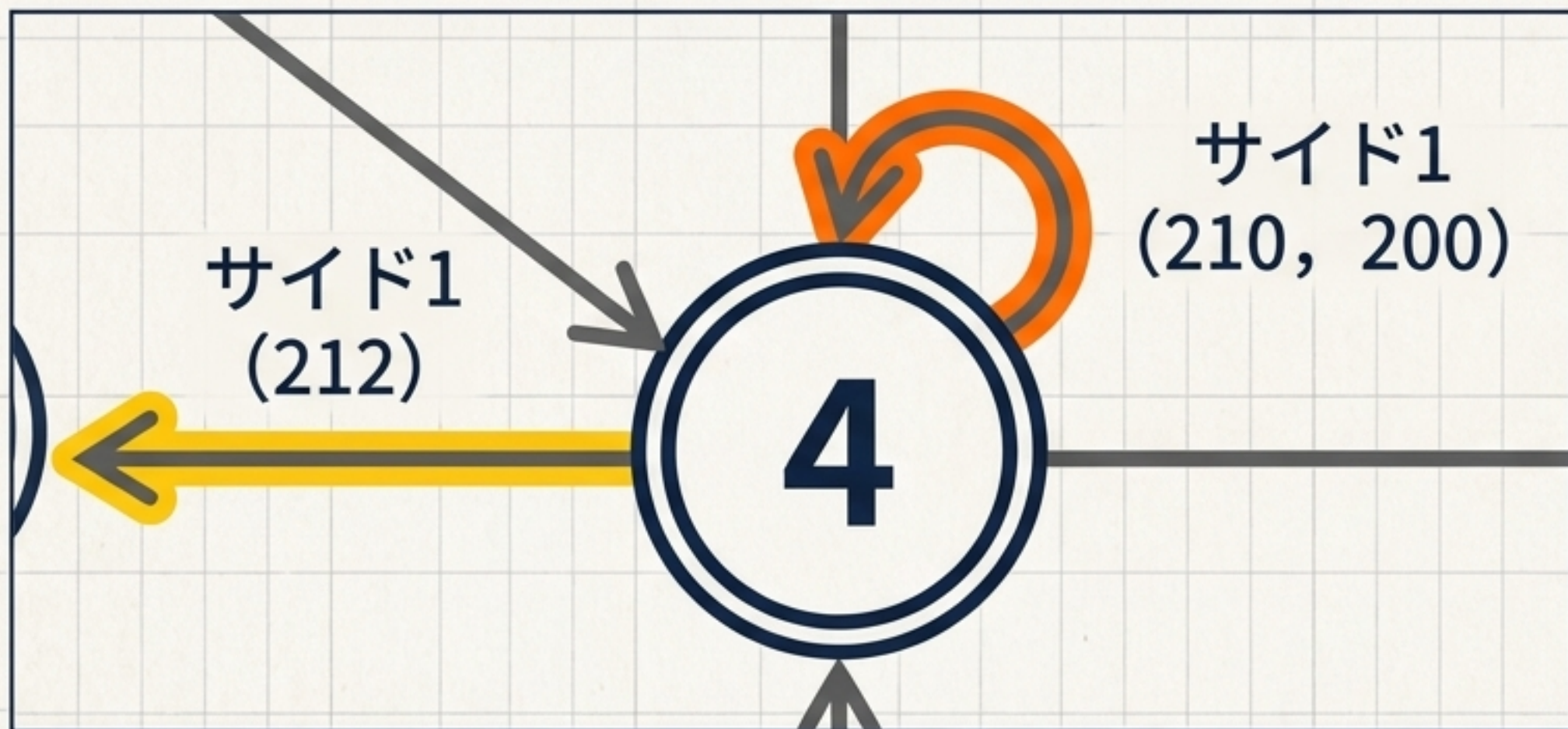
発券可
false
false
true
[ア]
true

状態3の円は、一重線か？二重線か？

Hint: 要件定義にある「オプション未指定時は押せない」という文言と、状態3の円の描き方を照らし合わせましょう。状態3の時点で注は完了していると言えますか？

設問1 (2)[イ] のヒント：自己ループと値引きのタイミング

Q: 状態4でコード210/212が来たときの「遷移先：値引額」は？



値引き額の判定ルール

- ・「メイン、サイド1、サイド2をそれぞれ1品以上選んだ場合に1回だけ50円引き」
- ・状態4の時点で、サイド2は既に揃っているか？

状態番号	...	210	212	300	...
4	...	4:0	イ	4:0	...

Hint: 値引きが発生する「瞬間」はいつか？
表2をよく見ると、値引き額が「50」になるのは
どの遷移の時か法則が見えてきます。

設問2 の全体像：プログラムループの解剖

プログラムの役割分担を理解する。このコードは「図3の判定手順」の直訳です。

図3 (2) 金額を加算

図3 (3) コード算出

図3 (4) 値引き取得

図3 (5) 状態の更新

```
amount ← amount + [ ウ ]
```

```
event_code ← [ エ ] ...
```

```
discount ← discount + [ オ ]
```

```
status ← [ カ ]
```

Hint: 穴埋め問題を解く第一歩は、「今、コードのどのステップ（目的）を処理しているか」を全体像から特定することです。

設問2 [ウ][エ] のヒント：計算と属性アクセス

order[i]	
A (金額)	
N (分類)	
S (セット)	
O (オプション)	

[ウ] 金額の加算

表3を確認。商品の「金額」データを持っている属性（プロパティ）はどれか？

[エ] イベントコードの算出

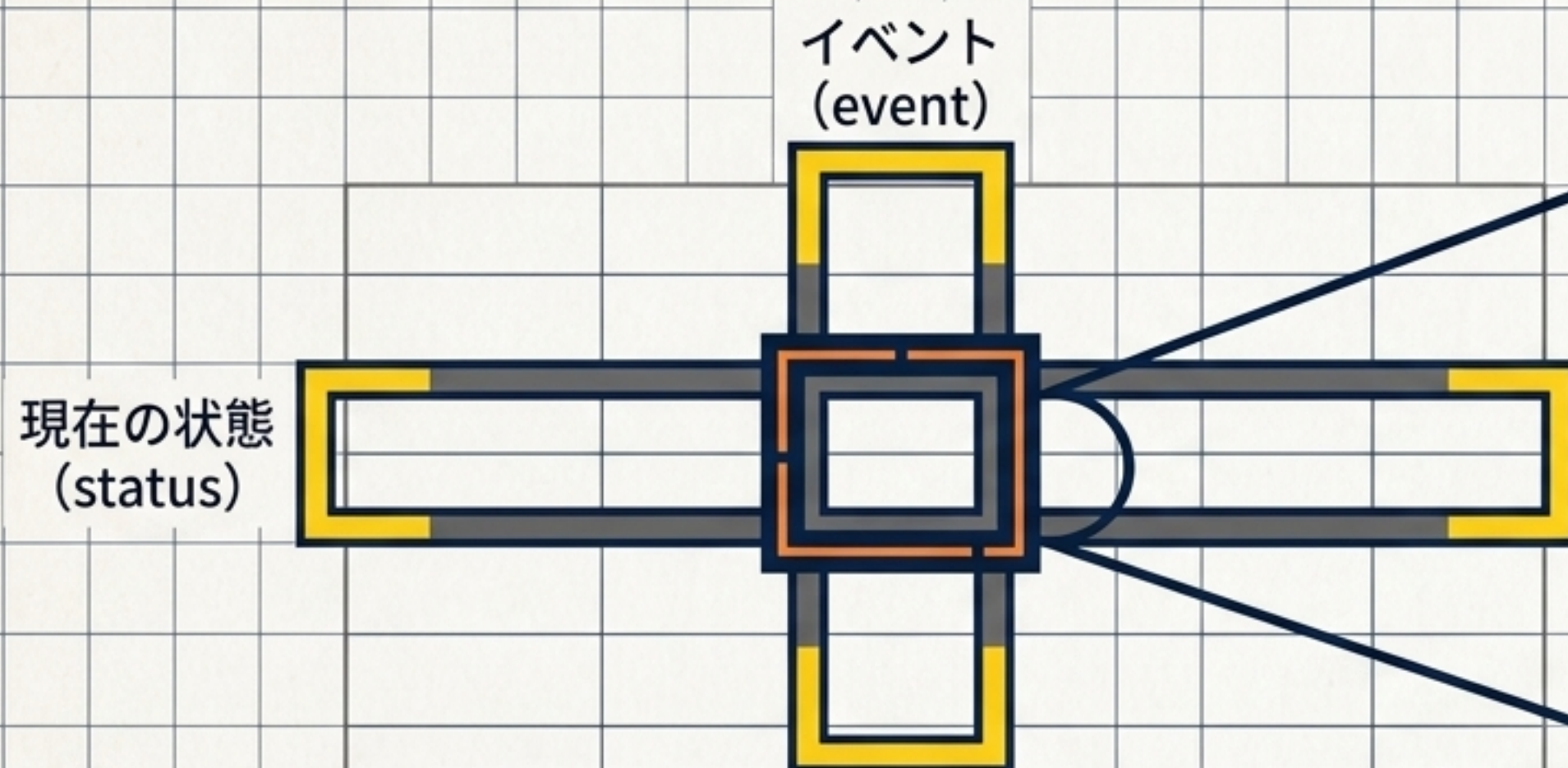
百の位	十の位	一の位
order[i].N	order[i].S	order[i].O
↓ × 100	↓ × 10	↓ × 1

例：N=1, S=1, O=0 を 110 にする算術式は？

Hint: コードの続きにある「+ order[i].O」（一の位の足し算）との整合性から、逆算して[エ]に入る掛け算の式を組み立てましょう。

設問2 [オ][カ] のヒント：2次元配列（表）からのデータ抽出

プログラム上の `status_table[行][列]` は、表2のマスを指す座標です。



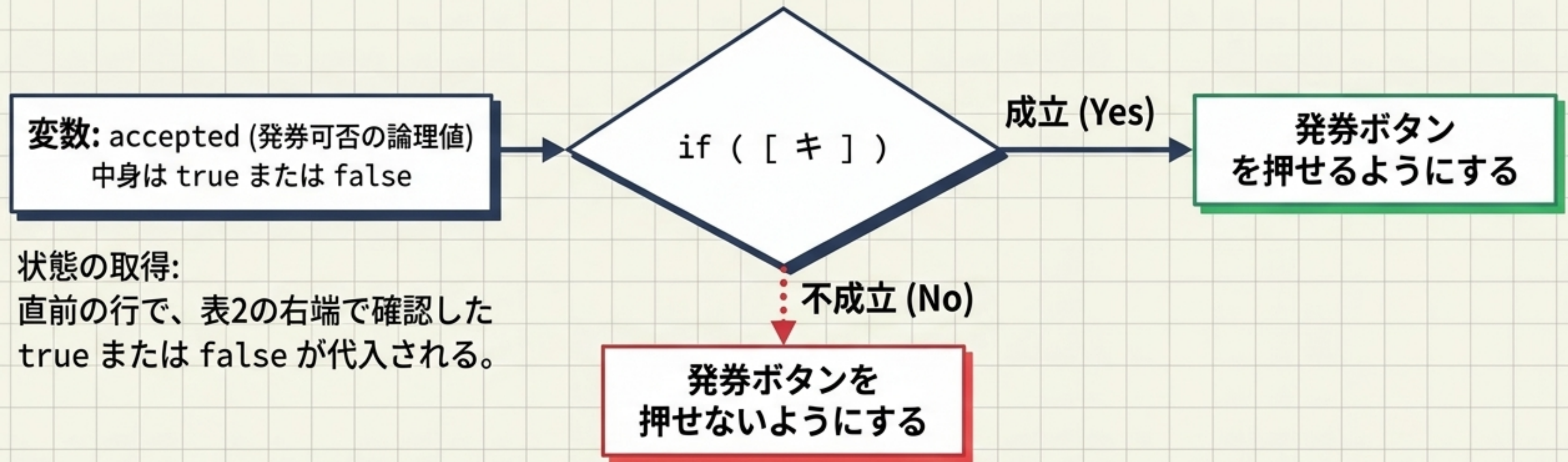
次の状態番号 = `.status`

値引き額 = `.discount`

Hint: [オ] は値引金額の加算、[カ] は状態番号の更新が目的です。
それぞれの変数名 (左辺) に合わせて、正しいプロパティ (属性) をマスの右辺から引っ張り出しましょう。

設問2 [キ] のヒント：最終判定の条件式

全てのループが終わった後、発券ボタンを押せるかどうかの最終チェック。



Hint: accepted がどのような値であるときに、ボタンを有効化すべきか？
条件式として明確に表現しましょう。(例：X が Y に等しい)

設問3 のヒント：データ駆動アプローチとは？

「プログラムに変更を加えず、表の変更だけで対応する」とはどういうことか？

ロジック / 固定



プログラム (図4)

汎用的なエンジン。ここは書き換えない (固定)。

データ / 変更可能



表 (表1・2)

新しい状態番号や商品を「マス目に追加・上書き」して拡張する。

つまり、表のマス目に追加・書き換えるだけで、
システムの挙動をコントロールできる設計になっている。

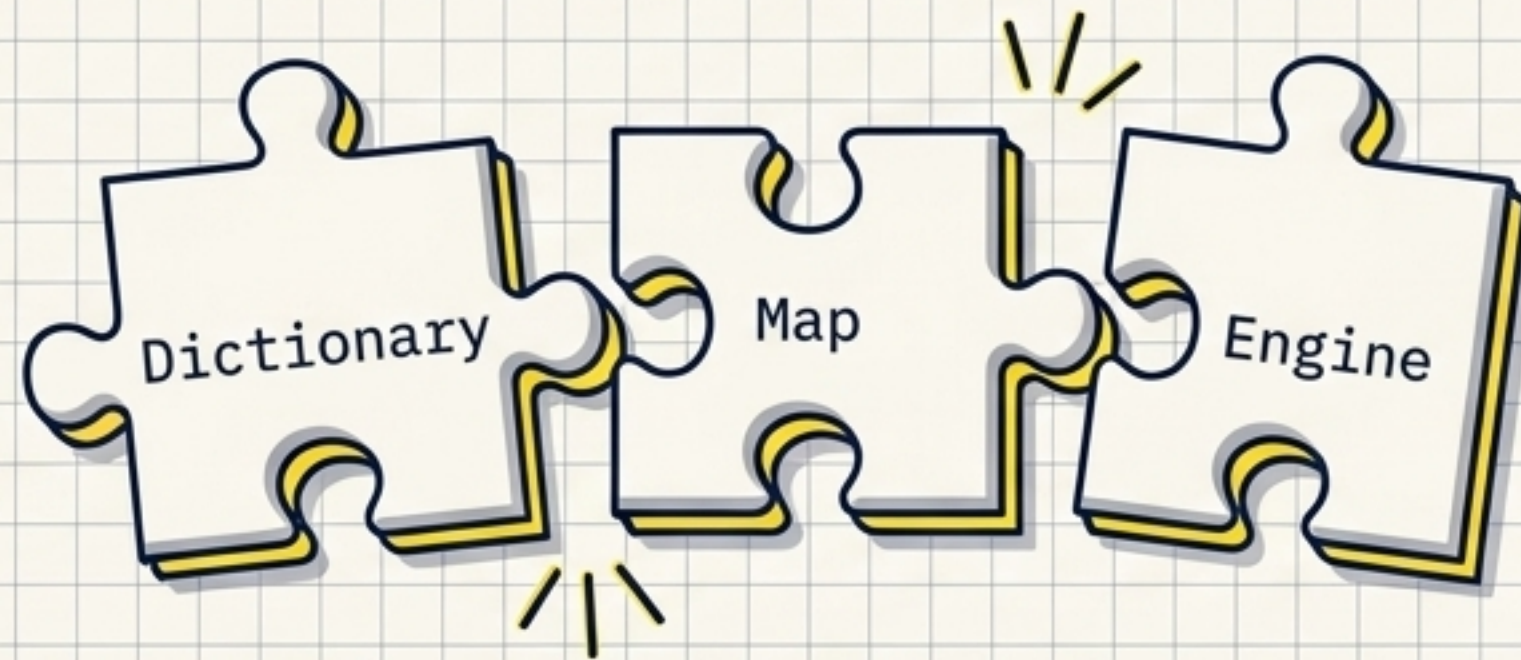
Hint: 選択肢を見る際、「表の項目を増やすだけで実現できるか？」それとも「そもそも判定手順のルール (if文など) を変えないと無理か？」という視点で仕分けします。

設問3 選択肢の診断マトリックス

選択肢	判定ポイント	アプローチの可否
ア (時間帯の制御)	表には「時間」の概念データが存在するか？ コードに時間判定を加える必要はないか？	☐
イ (二段階オプション)	既存のオプションの後に「つゆ待ち状態（新しい状態番号）」を経由させてから発券可へ繋ぐルートを、 表のマスを増やして作れないか？	☐
ウ (複数回値引き)	現状のプログラム自体が「ループ内で累積する仕組み」 を持っているか？（値引ルール自体が1回前提ではないか？）	☐
エ (商品追加)	カレーという商品を「分類1」として表1に定義し、 表2にそのコード列を追加するだけで動かないか？	☐

Hint: 「状態遷移の経路を増やす」「新しい商品データを追加する」ことで解決できる
選択肢を2つ探しましょう。

総括：システム解読の Key Takeaways



1. 抽象化 (Abstraction)

複雑なシステムは、単純な「数字（イベントコード）」への変換から始まる。人間の言葉を機械の言葉に翻訳するルールを見抜くこと。

2. 対応関係 (Mapping)

プログラム（コード）は、仕様書（図表）を忠実に直訳したものである。自然言語の手順とコードの対応関係を見抜くことが解読の最短ルート。

3. 分離 (Separation of Concerns)

「データ」と「ロジック」を分離する設計思想。優れたシステムは、コードを書き換えなくてもデータの追加だけで柔軟に拡張できる。

Now, Solve It!

さあ、これらの思考の設計図をもとに、実際の解答用紙を埋めてみましょう。